Universidad de Buenos Aires

Facultad de Ciencias exactas y Naturales

Licenciatura en Ciencias de la computación

Teóricas de Teoría de las Comunicaciones

Autor:
Julián Sackmann

10 de Septiembre de 2012

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

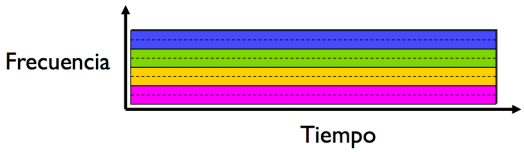
http://exactas.uba.ar

Índice

1 Introducción

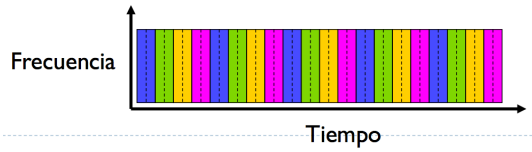
- **Internet**: la red de redes más grande del mundo, que usa los protocolos `TCP` y *packet switching*. Funciona sobre cualquier subsistema de comunicaciones.
- Nodo: tiene asociada una dirección (*bytestring* que lo identifica unívocamente)
  - *hosts*.
  - *switches*.
- Enlace:
  - Punto a punto.
  - Múltiple.
- **Red de computadoras**: se define recursivamente como:
  - Un nodo.
  - Dos o más nodos conectados por un enlace.
  - Dos o más redes conectadas por un nodo.
- **Conmutación de circuitos**: la conexión sólo se establece cuando se necesita. Una vez establecida, el ancho de banda queda reservado al usuario, lo use o no. Se subutiliza la infraestructura.
- **Conmutación de paquetes**: el ancho de banda disponible se comparte entre los diversos usuarios. Se **multiplexa** el tráfico. Se pueden generar circuitos virtuales. El ancho de banda no está reservado, pero se aprovecha mejor la infraestructura.
- **Routeo**: proceso de reenvío de un mensaje a su nodo destino basado en su dirección.
  - *unicast*: el destinatario es un nodo específico.
  - *multicast*: el destinatario es un conjunto específico de nodos.
  - *broadcast*: el destinatario son todos los nodos de la red.
- **Multiplexación**: es la combinación de dos o más canales de información en un solo medio de transmisión.
  - ...por división de tiempo (TDM).
  - ...por división de frecuencia (FDM).
  - ...por longitud de onda (WDM).
  - ...estadística. División del tiempo "bajo demanda". Los paquetes se encolan y compiten por el enlace. Si el buffer que tiene el nodo no alcanza, se denomina **congestión**.

FDM



- El espectro de frecuencias es dividido entre canales lógicos: cada usuario tiene posesión exclusiva de alguna banda de frecuencia.
- Requiere circuitería analógica.
- Provee mejor latencia[1].

TDM



- Los usuarios toman turnos (en round robin), obteniendo periódicamente cada uno el ancho de banda completo por un pequeño período de tiempo.
- Puede ser manejado enteramente por electrónica digital.
- Es más flexible (puede asignar más tiempo a señales más prioritarias dinámicamente).

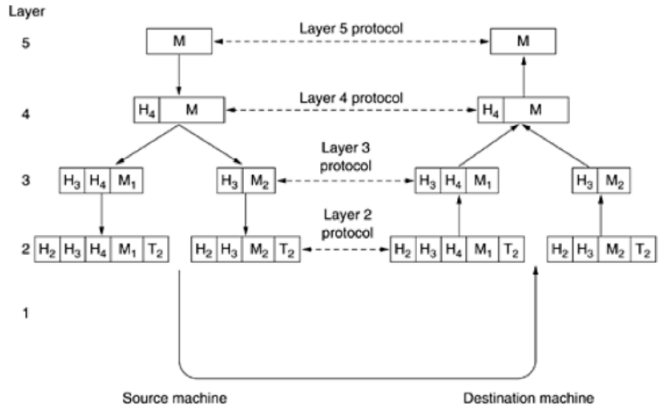
2 Modelo OSI

Es un modelo conceptual que caracteriza y estandariza las estructura de una red de comunicaciones, particionándola en **capas de abstracción**.

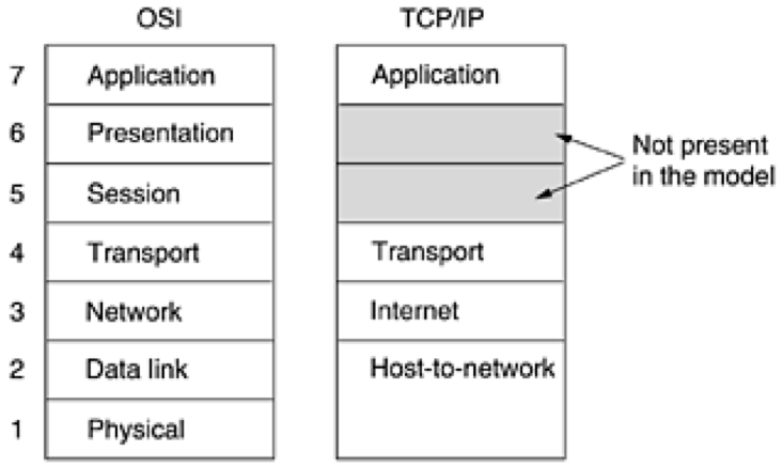
Modelo OSI			
	Unidad de dato	**Capa**	**Función**
Host	Datos	7. Aplicación	Servicio de red para aplicación
		6. Presentación	Representación de datos, cifrado, estandarización.
		5. Sesión	Comunicación entre hosts, manejo de sesiones entre aplicaciones.
	Segmento	4. Transporte	Routeo confiable de paquetes entre nodos de la red.
Medios	Paquete / Datagrama	3. Red	Direccionamiento, routeo no confiable de datagramas entre nodos.
	Bit / Frame	2. Enlace	Conexión confiable en enlace punto a punto.
	Bit	1. Físico	Conexión no confiable en enlace punto a punto.

Las comunicaciones se dan en capas que se brindan servicios entre sí. Pasar por cada capa implica el agregado de información de control en forma de encabezados:

[1] Latencia: el tiempo que le toma a los datos llegar a destino



2.1 OSI-ISO vs Internet



3 Nivel Físico

- **Señal**: es una onda electromagnética con dos campos ortogonales: uno eléctrico y uno magnético.
- **Onda electromagnética**: es un campo eléctrico magnético que se propaga por un medio, vibrando a una frecuencia determinada. TIene un comportamiento periódico en el eje longitudinal.
- **Logintud de onda**: es el periodo o repetición a longitudes constantes. Coloquialmente, es la "distancia que ocupa un ciclo". Se define como: $\lambda = \frac{v}{f}$ donde:
  - v es la velocidad de la onda en el medio de transmisión.
  - f es la frecuencia de oscilación.
- Una función f es periódica si existe un valor T tal que $f(t) = f(t+T)$. El T más chico que cumpla esto (mayor que 0) se lo llama **período fundamental**.
- La serie trigonométrica de Fourier permite expresar funciones periódicas $f(t)$ con período T de la siguiente forma:

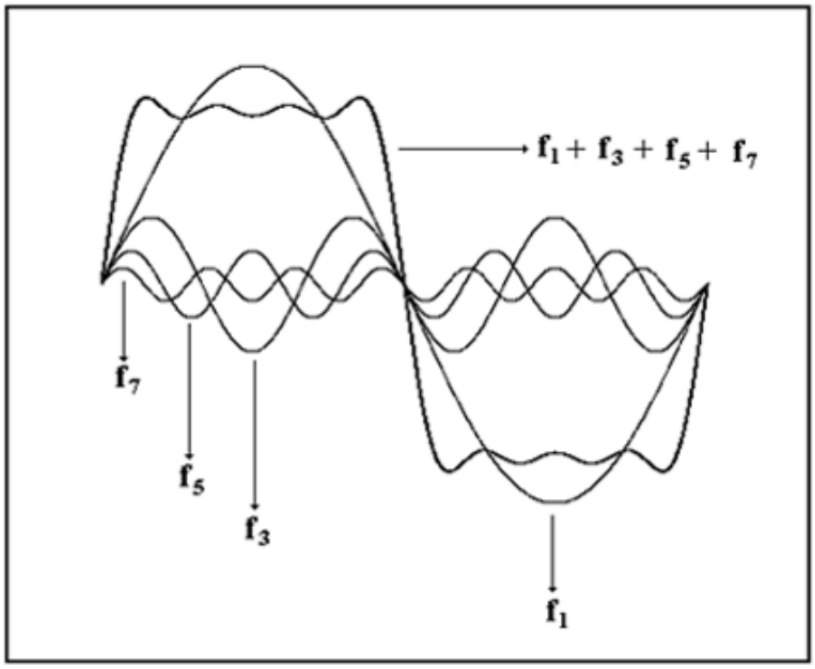
$$f(t) = \frac{1}{2}a_0 + \sum_{n=0}^{\infty}[a_n\ cos(n\omega_0\ t) + b_n\ sen(n\omega_0\ t)]$$

donde

$$\omega_0 = \frac{2\pi}{T}$$

Para encontrar los coeficientes, se usa la ortogonalidad.

Para enviar datos binarios por un canal necesitamos una onda cuadrada. Ergo, necesitamos emular una onda cuadrada con ondas senoidales:



- **Espectro**: margen de frecuencias contenidas en la señal.
- **Ancho de banda absoluto**: anchura del espectro.
- **Ancho de banda efectivo**: banda de frecuencias que contienen la mayor parte de la energía.
- **Componente continua (DC)**: componente de frecuencia cero.

3.1 Medios de transmisión

- Guiados:
  - Trenzado.
  - Coaxial.
  - Red eléctrica.
  - Fibra óptica.
- Sin Guía:

- Radio.
- Microondas.
- Infrarrojo.
- Láser.

3.2 Teorema del muestreo

Vimos que las señales periódicas se pueden descomponer como un sumatorio de senos y cosenos cada uno de una amplitud, frecuencia y fase diferente (Desarrollo en Serie de Fourier). Si dichas sinusoides las muestreamos, el caso más crítico de muestreo será aquella de mayor frecuencia (frecuencia máxima f_m que corresponde con el periodo mínimo $T_{min} = \frac{1}{f_m}$) la cual vamos a llamar:

$$f(t) = A\ sin(2\pi f_m t + \phi)$$

donde:

- A: amplitud.
- t: tiempo.
- ϕ: fase de la señal.

El **teorema de muestreo**, formulado por **Nyquist** en 1924 dice que si queremos reconstruir una señal de frecuencia máxima f_m, debemos de muestrear a $2f_m$. La frecuencia de muestreo (*sampling*) se llama f_s o también **frecuencia de modulación**.

Ejemplos:

- CDs de audio samplean 44.100 veces por segundo ⇒ captan frecuencias de 22.05KHz.
- La voz tiene un espectro de 4KHz en el teléfono ⇒ se tiene que samplear a 8000 muestras por segundo (125 μseg/muestra).

3.3 Conversión analógico digital

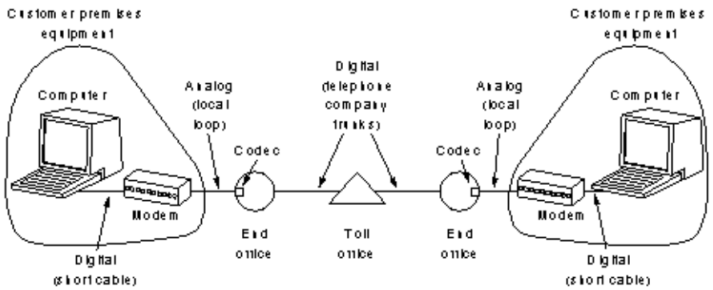


Fig. 2-17. The use of both analog and digital transmission for a computer to computer call. Conversion is done by the modems and codecs.

La conversión analógica digital consta de dos etapas.

- Se muestrea la señal al doble del ancho de banda, obteniendo una serie de **pulsos de amplitud variable** (PAM).
- Se cuantifican las muestras aproximándolas mediante un entero de n bits.

Ejemplo: el enlace `T1` consiste de 24 canales de voz de 8 bits cada uno multiplexados juntos. Son 193 bits (hay uno extra para framing) conduciendo cada 125 μseg. Si queremos calcular la velocidad:

$$\frac{1}{0.000125}\text{seg} \times 193 \text{ bits} = 1544000 \text{ bits/seg} = 1.544 \textbf{ Mbps}$$

3.4 Teoría de la información

Fue establecida por Claude Shannon. Predica sobre codificaciones usadas para canales con o sin ruido.
Uno de ellos describe la máxima eficiencia posible (teórica) de una codificación con corrección de errores frente a los niveles de ruido y corrupción de datos. Es un resultado no-constructivista (no dice nada sobre *cómo* implementar esa codificación).

Definición: Sea E un suceso que puede presentarse con probabilidad $P(E)$. Cuando E tiene lugar, decimos que hemos recibido $I(E)$ unidades de información, donde:

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

Si la probabilidad es muy alta, entonces nos da muy poca información.
Dependiendo de qué x utilicemos (la base del logaritmo), obtendremos una unidad distinta:

- $(x = 2) \Rightarrow I(E) = log_2(\frac{1}{P(E)})$ **bits** de información
- $(x = e) \Rightarrow I(E) = ln(\frac{1}{P(E)})$ **nats** de información.
- $(x = 10) \Rightarrow I(E) = log_{10}(\frac{1}{P(E)})$ **Hartleys** de información

Como caso particular de la primer unidad, notemos que si $P(E) = \frac{1}{2}$, entonces $I(E) = 1$ bit. O sea que **un bit es la cantidad de información obtenida al especificar dos posibles alternativas equiprobables**.

3.5 Fuente de memoria nula y Entropía

Supongamos que tenemos una fuente que emite una secuencia de símbolos pertenecientes a un alfabeto finito y determinado $S = \{s_1, s_2, .., s_q\}$. Los símbolos emitidos sucesivamente se eligen de acuerdo con una ley fija de probabilidad. Si los símbolos emitidos son estadísticamente independientes, se dice que la fuente S es **de memoria nula**.
Si consideramos la **información** suministrada por una fuente de memoria nula, la cantidad *media* de información por símbolo está dado por la fórmula:

$$\sum_S P(s_i)I(s_i) \text{ bits}$$

Definimos entonces a la **entropía** como la **cantidad media de información por símbolo de una fuente de memoria nula**. La notamos $H(S)$

$$H(S) = \sum_S P(s_i)\ log(\frac{1}{P(s_i)}) \text{ bits}$$

3.6 Entropía de un mensaje

La **entropía de un mensaje** X (que se representa por $H(X)$) es **el valor medio ponderado de la cantidad de información de los diversos estados del mensaje**.

$$H(X) = -\sum p(x) log\ p(x)$$

Representa una medida de la incertidumbre media acerca de una variable aleatoria y por tanto de la cantidad de información.

3.6.1 Propiedades de la entropía

- La entropía es no negativa.
- La entropía se anula $\Leftrightarrow$ un estado de la variable es 1 y el resto 0.
- La entropía es máxima (mayor incertidumbre del mensaje) cuando todos los valores posibles de X son equiprobables. Observemos que si hay n estados equiprobables ($P(i) = \frac{1}{n}$), entonces:

$$H(X) = -\sum_{i=1}^{n} P(i) * I(i) = -n\left(\frac{1}{n}\right)\ log_2\left(\frac{1}{n}\right) = -(log_2(1) - log_2(n))$$

$$H(x)_{max} = log_2(n)$$

Ejemplo: Sea una fuente S con alfabeto $S = s_1, s_2, s_3$, tal que $p(s_1) = \frac{1}{2}$ y $p(s_2) = p(s_3) = \frac{1}{4}$. Luego, los símbolos de la **fuente extendida** S^2 serán:

Simbolos	$sigma_1$	$sigma_2$	$sigma_3$	$sigma_4$	$sigma_5$	$sigma_6$	$sigma_7$	$sigma_8$	$sigma_9$
Secuencias	s_1s_1	s_1s_2	s_1s_3	s_2s_1	s_2s_2	s_2s_3	s_3s_1	s_3s_2	s_3s_3
$P(\sigma_i)$	1/4	1/8	1/8	1/8	1/16	1/16	1/8	1/16	1/16

Luego,

$$\begin{aligned} H(S^2) &= \sum_{S^2} P(\sigma_i)\ log_2\left(\frac{1}{P(\sigma_i)}\right) \\ &= \frac{1}{4}log_2(4) + 4 \times \frac{1}{8}log_2(8) + 4 \times \frac{1}{16}log_2(16) \\ &= 3 \text{ bits/símbolo} \end{aligned}$$

3.7 Fuente de Markov

Definimos una una **fuente de Markov de orden** m como una fuente en la que **la probabilidad de un símbolo cualquiera viene determinada por los** m **símbolos que lo preceden**.

En cualquier momento, por lo tanto, definiremos el *estado* de la fuente de Markov de orden m por los m símbolos precedentes. Puesto que existen q símbolos, una fuente de Markov admitirá q^m estados posibles.

Para visualizar una fuente de Markov se puede usar un diagrama de estados de q^m nodos (estados). Ejemplo:

Ejemplo 2-3. Consideremos una fuente de Markov de segundo orden con un alfabeto binario $S = \{0,1\}$. Supongamos que las probabilidades condicionales son

$$P(0/00) = P(1/11) = 0.8$$
$$P(1/00) = P(0/11) = 0.2$$
$$P(0/01) = P(0/10) = P(1/01) = P(1/10) = 0.5$$

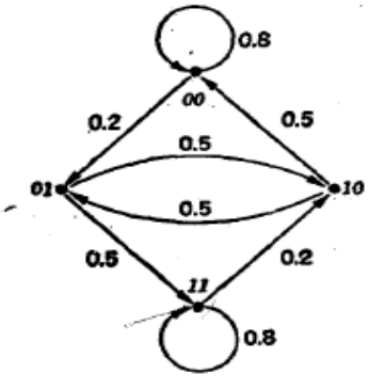


FIG. 2-4. Diagrama de estados de una fuente de Markov de segundo orden.

3.8 Codificación

Se define **codificar** como establecer una correspondencia entre los símbolos de una fuente y los símbolos del alfabeto de un código. Es un proceso mediante el cual podemos lograr una representación más eficiente de la información (eliminando redundancia).

3.8.1 Condición de prefijos

Para que un código sea **instantáneo** es condición *necesaria* y *suficiente* que sus palabras cumplan con la **condición de los prefijos**: no debe existir una palabra que sea prefijo de otra de mayor longitud.

3.8.2 Códigos eficientes

Una forma de lograr códigos eficientes es asignar las palabras más cortas a los símbolos más probables. De este modo se baja la longitud media de un código:

$$\sum p_i \; l_i$$

donde

- l_i: longitud de la palabra codificada del mensaje m_i.
- r: cantidad de símbolos del alfabeto del código.
- $log(r)$: cantidad promedio máxima de información de un símbolo.

De este modo, podemos calcular la **eficiencia** de una codificación como:

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

Observemos que $L\ log(r) \geq H(S)$, con lo cual $\eta \leq 1$.

Se dice que un codificador es **óptimo** si para codificar un mensaje X usa el menor número posible de bits.

3.9 Perturbaciones en la transmisión

Se dice que hay una **perturbación** en una transmisión cuando la señal recibida difiere de la emitida. Una perturbación puede ser analógica (degradación de la calidad de la señal) o digital (errores de bits) y puede ser causada por:

- **Atenuación y distorsión**: la intensidad de la señal disminuye con la distancia (depende del medio). La señal recibida debe ser suficiente para que se la detecte y el ruido no interfiera.
- **Distorsión de retardo**: las componentes de frecuencia llegan al receptor en distintos instantes de tiempo, originando desplazamientos de fase entre las distintas frecuencias.
  - Sólo aplica a medios guiados.
  - La velocidad de propagación varía con la frecuencia.
- **Ruido**: señales adicionales insertadas entre el transmisor y el receptor. Puede ser de varios tipos:
  - Térmico: se debe a agitación térmica de electrones.
  - Intermodulación: se produce por falta de linealidad.
  - Diafonía: una señal de una línea se mete en otra.
  - Impulsiva: impulsos irregulares o picos de corta duración pero mucha intensidad.

3.10 Capacidad del canal

- **Velocidad**: bits/segundo.
- **Ancho de banda**: ciclos/segundo (hertz).

La ley de **capacidad de Shannon** establece que para un cierto nivel de ruido, a mayor velocidad (ergo, menor período de un bit) mayor es la tasa de error (porque se pueden corromper 2 bits en el tiempo en que antes se corrompía uno). Formalmente, al relación señal/ruido se calcula como:

$$SNR_{dB} = 10 \times log(SNR) = 10 \times log\left(\frac{\text{Potencia señal}}{\text{Potencia Ruido}}\right)$$

Sin embargo, la cantidad de niveles debe mantenerse: $M \leq \sqrt{1 + SNR}$

En principio, si se aumenta el ancho de banda (B) y la potencia de la señal (S), aumenta la velocidad binaria (C). Sin embargo,

- Aumentar el ancho de banda aumenta el ruido.
- Aumentar la potencia impacta en las no linealidades y el ruido de la intermodulación.

Luego, la velocidad binaria teórica máxima está dada por la fórmula:

$$C_{max}(bps) = B(Hz) \times log_2(1 + SNR)$$

3.11 Ancho de banda de Nyquist

Definimos el **ancho de banda de Nyquist** como la **capacidad teórica máxima de un canal**:

$$\text{Para } M \text{ niveles sin ruido} \Rightarrow C(bps) = 2B(Hz)\ log_2(M)$$

3.11.1 Modulación

La **modulación** es proceso de variación de cierta característica de una señal (llamada *portadora*) de acuerdo con la señal del mensaje (llamada *moduladora*).

Existen 4 tipos de modulación:

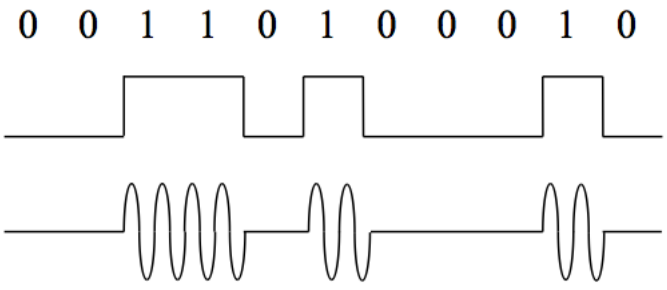
- Moduladora Analógica / Portadora Analógica: usada para las radios AM/FM.
- Moduladora Analógica / Portadora Digital: conversión de señales analógicas en digitales (`digitalización`), llevada a cabo por CODECS. Puede ser *modulación por impulsos modificados* (PCM), o *modulación delta* (se codifican sólo las diferencias).
- Moduladora Digital / Portadora Analógica: se usa para la transmisión de datos digitales a través de la red de telefonía (que fue diseñada para transmitir señales analógicas en el rango de frecuencias de voz (*300-3400Hz*)).
- Moduladora Digital / Portadora Digital: los datos binarios se transmiten codificando cada bit de datos en cada elemento de señal. Evita problema de sincronismos.

Moduladora Digital / Portadora Analógica

Hay distintas técnicas:

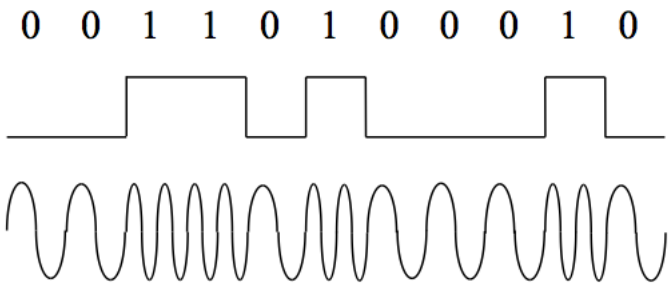
- Desplazamiento de Amplitud (`ASK`): los valores binarios se representan mediante dos amplitudes de la onda portadora.

$$S(t) = \begin{cases} A \times cos(2\pi f_c t) \\ 0 \end{cases}$$



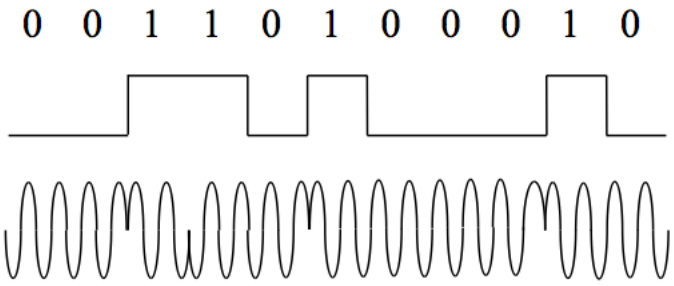
- Desplazamiento de Frecuencia (`FSK`): los valores binarios se representan mediante dos frecuencias de la onda portadora.

$$S(t) = \begin{cases} A \times cos(2\pi f_1 t) \\ A \times cos(2\pi f_2 t) \end{cases}$$



- Desplazamiento de Fase (`PSK`).

$$S(t) = \begin{cases} A \times cos(2\pi f_c t + \pi) \\ A \times cos(2\pi f_c t) \end{cases}$$



- Mixtas.

Definiciones

- **Velocidad de modulación**: se define como el número de cambios de señal por unidad de tiempo. Se expresa en *baudios* (símbolos por segundo).
- **Velocidad de transmisión**: equivale a la velocidad de modulación multiplicada por el número de bits representados por muestra.

Observemos que podemos aumentar la velocidad de transmisión representando más de un bit con cada elemento de la señal. Por ejemplo:

$$S(t) = \begin{cases} Acos(2\pi f_c t + 45) & \Rightarrow & 00 \\ Acos(2\pi f_c t + 135) & \Rightarrow & 01 \\ Acos(2\pi f_c t + 225) & \Rightarrow & 10 \\ Acos(2\pi f_c t + 315) & \Rightarrow & 11 \end{cases}$$

3.11.2 Codificación

Para la modulación digital con portadora digital, también existen distintas técnicas:

- **NRZ**: consiste en usar una tensión positiva para un 1 y una negativa para un 0. Hay pérdida de sincronización cuando un mismo símbolo se repite muchas veces seguidas (¿fueron diez 0, u once?).

- **NRZI**: cada vez que aparece un 1 en la señal original, se genera una transición. Soluciona muchos 1 seguidos, pero no con los 0.
- **Manchester**: se codifica mediante una transición en la mitad del intervalo de duración del bit: de bajo a alto representa un 1 y de alto a bajo un 0.
- **Manchester diferencial**: la codificacón de un 0 se representa por la presencia de una transición al principio del intervalo del bit y un 1 mediante la ausencia.
- **Bipolar-AMI**: un 0 se representa por la ausencia de señal y un 1 se repreesnta mediante un pulso positivo seguido de uno negativo.

Existen codificaciones **de alta densidad**, en las que se reemplazan secuencias de bits que dan lugar a niveles de tensión constante por otras que proporciones transiciones. Esto sirve para mantener la sincronización (puede ser dificil distinguir 18 intervalos de no tensión de 19. Si alterna es más fácil). El receptor debe identificar la secuencia reemplazada y sustituirla por la original.

4 Nivel de Enlace

4.1 Conceptos

- El nivel de enlace debe proveer un enlace confiable para comunicación punto a punto. Esto involucra dos aspectos:
  - **Ruido impulsivo**: lo que se recibe puede no ser lo mismo que lo que se envió. ¿Qué hacemos con los errores?
  - **Desorden**: los paquetes deben llegar en el mismo orden en el que fueron enviados.
- **Framing**: encapsular los bits en ***frames*** agregando información de control. Estos paquetes funcionan como unidades.
- ¿Cómo delimitamos los frames en la transmisión? Hay varias formas:
  - Largo fijo.
  - Largo informado en el encabezado.
  - Delimitadores con *bit-stuffing*.
  - Violación de código: mandar una señal no especificada (por ejemplo, si estoy usando 5 volt para codificar el 0 y -5 para el 1, puedo mandar 0 volt).

Existen tres tipos de servicio que el nivel de enlace provee:

- **Sin conexión y sin reconocimiento**: los datos se envían sin necesidad de saber si llegan bien.
- **Sin conexión y con reconocimiento**: los datos se envían y se asegura la correcta recepción mediante el aviso explícito (`ACK`).
- **Orientado a conexión**: además de asegurar la recepción correcta de los datos, se mantiene una conexión (*sesión*).

4.2 Corrección y detección de errores

Supongamos que tenemos un mensaje de m bits de longitud y agregamos r bits de redundancia. Sea $n = r + m$ la longitud del *codeword* resultante.

Se define la **distancia de Hamming** (d) como la cantidad de bits que deben ser flipeados para transformar una palabra en otra (cuanto mayor sea esta diferencia, menor es la posibilidad de que un código válido se transforme en otro código válido por una serie de errores).

Llamemos e a la cantidad de errores introducidos durante la transmisión.

- Si $e + 1 \leq d \Rightarrow$ se puede **detectar** el error.
- Si $2e + 1 \leq d \Rightarrow$ se puede **corregir** el error.

4.3 Retransmisiones

Cuando se detecta un error, pero no se puede corregir, es necesario que el emisor retransmita el paquete. Hay dos tipos de retransmisiones:

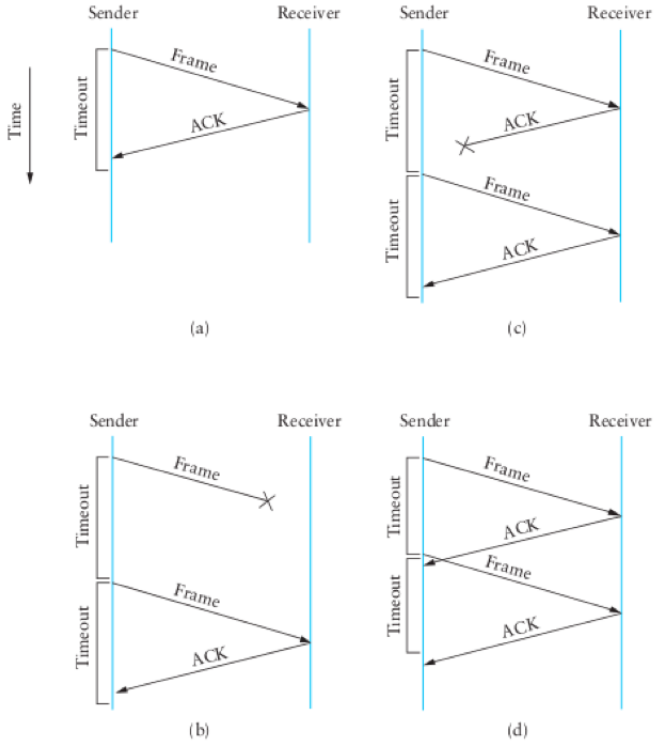
- Explícitas: involucra mensajes de control específicos para pedir un datos nuevamente.
- Implícitas: el receptor descarta el mensaje y cuando ocurra un time-out se asume que el dato se perdió y el emisor lo reenvía.

4.3.1 Stop and wait

Stop and wait es un protocolo de transmisión de nivel de enlace que garantiza que la información no se pierda (por paquetes *dropeados*) y llegue en orden. Es un caso particular del ***sliding window protocol*** (con ambas ventanas de tamaño 1): el emisor envía un paquete por vez. Luego espera hasta que:

- Le llegue el `ACK` confirmando que el receptor lo recibió bien.
- Pasa el *timeout* tras el que se considera que el mensaje se perdió y el emisor lo reenvía.

Observemos que de acá surge la necesidad de tener numerados los frames. El receptor debe poder identificar cada uno.



4.3.2 Eficiencia

Se mide la **eficiencia** de una transmisión en función de la cantidad de tiempo que uno se pasa bloqueado.

$$\text{Eficiencia } = \frac{T_{tx}}{RTT}$$

4.3.3 Capacidad

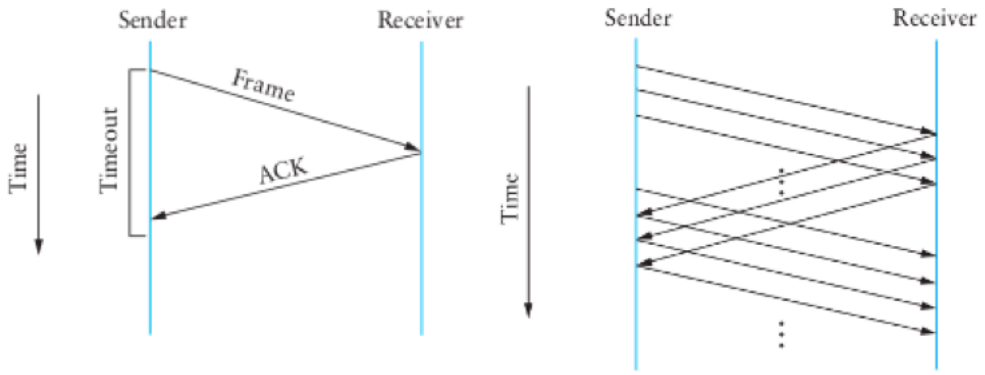
La **capacidad** de un canal se obtiene mediante el producto:

$$Bandwith \times delay$$

Este producto entre la velocidad de transmisión y el *delay* nos da **la cantidad de bits que entran en un canal**. Para aprovechar mejor, el canal debería estar siempre lo más lleno posible.

4.4 Sliding Window

Observemos que el protocolo *stop and wait* no llena demasiado el canal. Es por esto que se inventó un protocolo que, si bien es más complejo, tiene mucha mayor capacidad: **Sliding Window**.



El protocolo sliding window (*de ventana deslizante*) se utiliza para garantizar la entrega confiable y en orden de paquetes.

Conceptualmente a cada paquete se la asigna un número de secuencia, que el receptor utiliza para verificar que estén todos y reordenarlos. Se acota la cantidad de paquetes que pueden enviarse o recibirse en un momento dado. La cantidad de paquetes que el emisor puede mandar "seguidos" se conoce como **SWS** (*sender window size*) y se calcula de la siguiente manera:

$$SWS = V_{tx} \times \frac{RTT}{|Frame|}$$

donde

- $RTT = 2 * delay$ (Round trip time): el tiempo que le toma a un paquete ir y volver por el caño.
- $|Frame|$ es el tamaño del paquete.

Hay dos opciones para implementar este protocolo: con o sin **selective ack**.

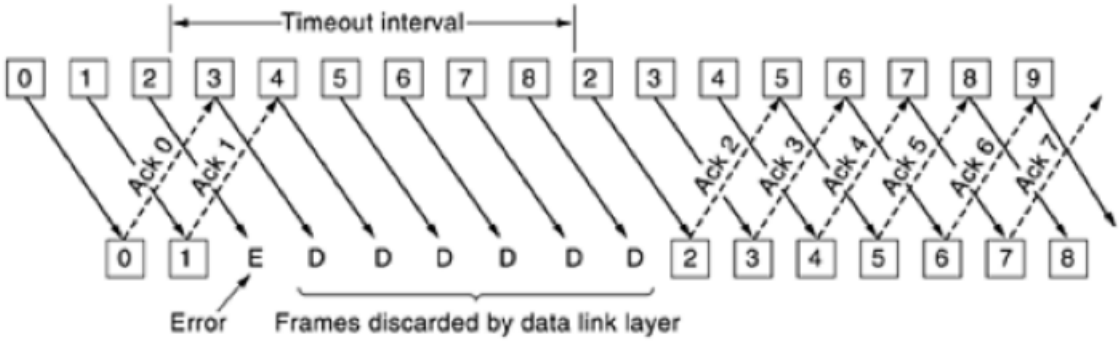
4.4.1 Sin Selective ACK

Sliding window sin selective ACK es más sencillo de implementar y requiere menos capacidad de almacenamiento en los nodos (particularmente en el receptor).

Como el receptor no hace *buffering*, simplemente manda `ACK` cuando recibe un frame correctamente. Si ocurre un error, descarta ese frame y todos los siguientes hasta que el emisor vuelva a enviar ese frame.

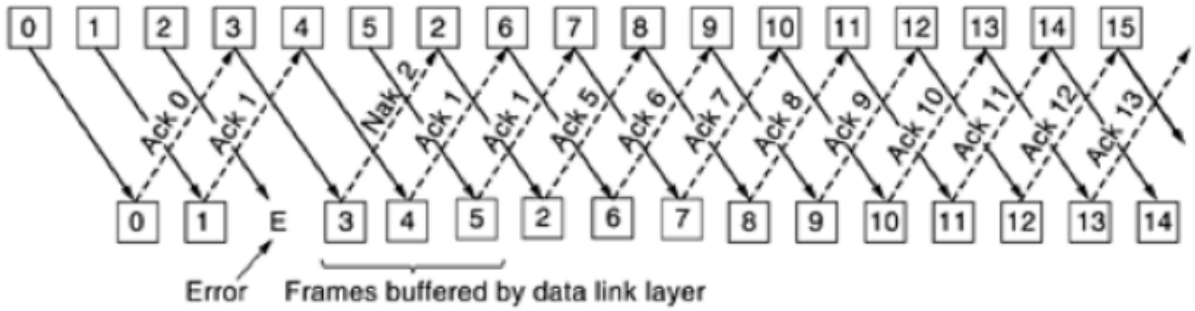
El emisor manda paquetes mientras no superen su `SWS` y va contando los `ACK` que recibe. Si en algún momento pasa el *timeout* de un paquete lo reenvía y sigue la secuencia desde ese paquete.

La desventaja enorme de esta técnica es que se deben reenviar todos los paquetes que ya se habían enviado (posiblemente en forma correcta) sólo porque se perdió uno anterior.



4.4.2 Con Selective ACK

Usando selective `ACK` se puede mitigar este problema para minimizar la cantidad de retransmisiones necesarias, pero a costa de complejizar el protocolo y necesitar más capacidad de almacenamiento en ambos nodos. Particularmente en el receptor es necesario agregar un *buffer*.



Se agrega el concepto de ventana también para el receptor **RWS** (receiver window size). No usar selective `ACK` se puede considerar como $RWS = 1$.

Si se usa selective `ACK`, entonces $RWS = SWS$ (en caso óptimo. Si hay problemas de buffering o cosas así, puede ser menor $1 \leq RWS \leq SWS$).

El emisor se mantiene las siguientes variables:

- n_t: próximo número de secuencia a enviar.
- w_t: tamaño de la ventana del emisor.
- n_a: número más alto de paquete que el receptor confirmó recibir.

El receptor se mantiene las siguientes variables:

- n_r: primer paquete que todavía no recibí.
- w_r: tamaño de la ventana del receptor.
- n_s: 1 + paquete más grande que recibí.

Observemos que:

- Todos los paquetes con número menor que n_r ya fueron recibidos.
- Todos los paquetes con número mayor que n_s todavía no fueron recibidos.
- Algunos paquetes con número entre n_r y n_s fueron recibidos y otros no.
- El emisor sabe que el receptor recibió todos los paquetes con número menor que n_a.
- Pero no está seguro los que están entre n_a y n_s.
- Siempre se mantiene el orden $n_a \leq n_r \leq n_s \leq n_t \leq n_a + w_t$.
  - $n_a \leq n_r$: el ACK más alto recibido por el transmisor no puede ser mayor al primer paquete no recibido.
  - $n_r \leq n_s$.
  - $n_s \leq n_t$: no pude haber recibido un paquete más alto que el más alto enviado.
  - $n_t \leq n_a + w_t$: el paquete más alto enviado está limitado por el ACK más alto recibido y la ventana de transmisión del emisor.

4.5 LAN Switching

En las redes **de acceso compartido** (o *LAN Switching*) se comparte un sólo medio físico para varios hosts, por lo que surge la necesidad de usar algún esquema de direccionamiento y control de acceso.

Los sistemas en los cuales varios usuarios comparten un canal común de modo tal que pueden ocurrir conflictos se conocen como **sistemas de contención**. Ejemplo de estos son:

- Ethernet (802.3).
- Wifi (802.11).
- Token Ring (802.5).

Para disminuir la cantidad de colisiones en un medio compartido se utilizan dos conceptos:

- Demora aleatoria: se asume que la pérdida de cualquier trama se debe a una colisión con otra. Luego, los emisores esperan hasta volver a enviar una cantidad aleatoria de tiempo.
- Exponential backoff: cada vez que una trama colisiona, el emisor espera el doble de tiempo de lo que esperó la última vez (para la misma trama), aunque también incluye un componente aleatorio. De esta forma se minimizan (estadísticamente) las colisiones.

4.6 CSMA

CSMA (*Carrier Sense Multiple Access*) es un método de control de acceso probabilístico en el que los nodos verifican si el canal compartido está ocupado antes de comenzar su propia transmisión.

Notemos que `CSMA` no puede evitar completamente las colisiones (el retardo de propagación significa que los dos nodos pueden no escuchar la transmisión de cada uno). Es un método probabilístico.

Viene en dos sabores: `CSMA`/**CD** (*collision detection*) y `CSMA`/**CA** (*collision avoidance*).

4.6.1 CSMA/CD

Collision detection es la estrategia utilizada por `802.3` (*Ethernet*). Esto se debe a que tiene la capacidad de escuchar el canal mientras transmite por él (por ser *full duplex*).

Utiliza dos estrategias. En primer lugar, el adaptador no transmite si detecta que algún otro adaptador está transmitiendo. A esto se le llama **detección de portadora**. Por otro lado, si el adaptador detecta que otro adaptador está transmitiendo mientras él mismo estaba transmitiendo, aborta la transmisión y envía la señal de *Jam*. A esto se le llama **detección de colisión**.

Cualquiera sea estos dos motivos, el adaptador espera un tiempo aleatorio con *exponential backoff* para volver a intentar la retransmisión.

La eficiencia de `CSMA/CD` está dada por la fórmula:

$$\text{Eficiencia} = \frac{1}{1 + 5\frac{t_{prop}}{t_{trans}}}$$

donde

- t_{prop}: tiempo máximo para propagarse entre dos nodos de la `LAN`.
- t_{trans}: tiempo de transmitir el marco de tamaño máximo.

Observemos que esta noción de eficiencia tiende a 1 cuando $t_{prop} \to 0$ o $t_{trans} \to \infty$.

Ventajas:

- La detección de colisiones en redes LAN cableadas es fácil.
- El tiempo medio necesario para detectar una colisión es relativamente bajo.
- Puede ser empleado en sistemas de control de procesos continuos si la carga de tráfico de la red es baja (inferior al 20 %).
- Ofrece un rendimiento mayor en especial cuando existen pocas colisiones.

Desventajas:

- No es posible garantizar un tiempo máximo finito para el acceso de las tramas al canal de comunicación (problemas con real-time).
- No se puede usar con redes *half-duplex*, como `802.11` (*wifi*) (mientras una estación envía información es incapaz de escuchar el tráfico existente).
- Problemática en redes inalámbricas (problema de la terminal oculta).

4.6.2 CSMA/CA

Collision Avoidance es la estrategia usada por `802.11` (*Wifi*). Debido a que la mayor parte de las redes son *half-duplex*, no tienen la capacidad de escuchar a medida que transmiten, con lo que es necesaria una nueva estrategia.

Antes de transmitir, una estación debe determinar el estado del medio. Si el medio está libre, el emisor realiza una espera adicional llamada **IFS** (*inter frame spacing*). El objetivo de este espaciado es evitar la colisión en el caso de que el medio hubiera estado siendo usado, pero todavía fuera del *threshold* de detección

Si, en cambio el canal está haciendo ocupado (o se ocupa durante el `IFS`) se procede a hacer *exponential backoff*, eligiendo una **CW** (congestion window).

4.7 Ethernet

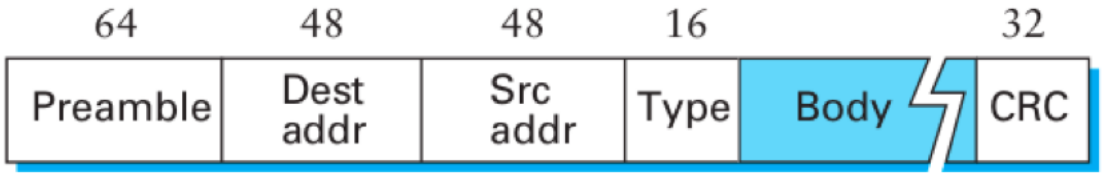
Tiene los siguientes tipos de cables:

- 10base2 ⇒ Coaxil (10Mbps, 200m)
- 10base5 ⇒ Coaxil (10Mbps, 500m)
- 10baseT ⇒ Par trensado (10Mbps, 100m)

Se intenta que los tramos no sean más de 500 metros para que disminuya el riesgo de colisiones y la atenuación de la señal.

4.7.1 Frame format

El frame de ethernet respeta el siguiente formato:



donde

- **Preámbulo**: permite a los dispositivos fácilmente detectar un frame entrante.

- **Dest addr**: dirección MAC del destinatario.
- **Src addr**: dirección MAC del emisor.
- **Type**: Tipo de paquete (`ARP`, etc).
- **Body**: Contenido del mensaje.
- **CRC**: validación del paquete.

Dado que se comparte el medio, todos los nodos van a recibir los paquetes que todos envíen. Es necesario establecer un criterio de filtrado para que cada host pueda determinar que paquetes le corresponden. Luego, un host de ethernet sólo va a procesar paquetes en caso de que:

- El destino del paquete sea ese host.
- El destino del paquete sea "broadcast" (FF:FF:FF:FF:FF:FF).
- El destino del paquete sea una dirección "multicast" a la que está suscripto.
- El host esté en modo promiscuo (agarra todos los frames).

4.8 WiFi

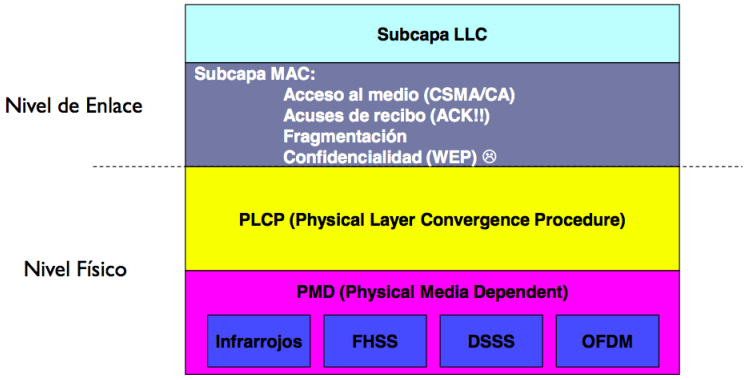
Las tecnologías inalámbricas tienen numerosos inconvenientes:

- Pierden intensidad con la distancia.
- Tienen más fuentes de ruido que las guiadas (y más impredecibles).
- Privacidad de los datos.
- Licenciado de bandas.

Wi-Fi (*Wireless-Friendly*) es un nombre para una serie de estándares definidos por la `IEEE` para comunicación por redes wireless:

- 802.11.**b** $\Rightarrow$ 11 Mbps.
- 802.11.**a** $\Rightarrow$ 54 Mbps.
- 802.11.**g** $\Rightarrow$ 54 Mbps.
- 802.11.**n** $\Rightarrow$ 100 Mbps (600 Mbps con `MIMO` (*multiple input multiple output antennas*)).
- 802.11.**ac** $\Rightarrow$ 1 Gbps.

El modelo de referencia de 802.11 es:



Como se ve, en su nivel físico hay 4 alternativas de transmisión:

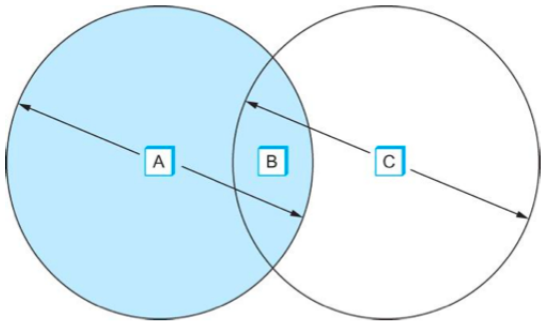
- **Infrarrojo**: hoy en día muy poco usado por lento y poca distancia
- **FHSS** (*Frequency Hopping Spread Spectrum*): sistema de bajo rendimiento, también muy poco usado.
- **DSSS** (*Direct Sequence Spread Spectrum*).
- **OFDM** (*Orthogonal Frequency Division Multiplexing*).

Configuraciones típicas de la infraestructura

- **BSS** (*Basic Service Set*): un *access point* provee la funcion de un puente (*bridge*) local para `BSS`. Todas las estaciones se comunican con con el *access point* y no directamente entre ellas. Las tramas son retransmitidas entre las estaciones Wi-Fi por el *access point*.
- **ESS** (*Extended Service Set*): un `ESS` es un conjunto de `BSS`s, donde los access points se comunican entre ellos para forwardear el tráfico desde una `BSS` a otra.
- **AdHoc**: las estaciones inalámbricas se comunican directamente entre sí. Cada estación puede o no ser capaz de comunicarse con otra debido a las limitaciones rango.

Problema de la estación oculta

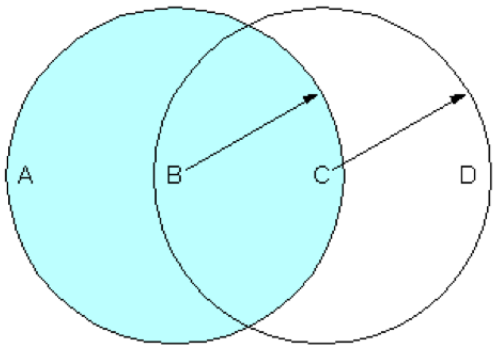
Se conoce al **problema de la estación oculta** a la situación en la que una estación no puede detectar a un competidor potencial por el medio porque está demasiado lejos.



Supongamos que ocurre cuando A transmite a B. Si C detecta el medio no escuchará a A porque está fuera de su alcance, y por lo tanto deducirá erróneamente que puede transmitir. Si C comienza a transmitir, interferirá en B eliminando la trama de A.

Problema de la estación expuesta

Ahora consideremos la situación opuesta: B transmite a A. Si C detecta el medio, escuchará una transmisión y concluirá que no puede enviar a D. Cuando de hecho tal transmisión causaría una mala recepción solo en la zona entre B y C, en la que no está localizado ninguno de los receptores pretendidos. Esta situación se conoce como **problema de estación expuesta**.



Ambos dos problemas se dan porque antes de comenzar una transmisión lo que se quiere saber si hay o no actividad en las cercanías **del receptor** y no alrededor del transmisor. Estos dos problemas son parte de la causa por la que no es posible implementar *collision detection* es una red wireless. El otro motivo es económico, se necesitan nodos *full-duplex*, que los hace más caro.

4.8.1 MACA y MACAW

Dado que las técnicas de `CSMA` no funcionan bien en medios inalámbricos, se desarrollaron otras técnicas.

MACA

(*Múltiple Access Collision Avoidance*) se usó como base para el 802.11. El concepto en que se basa es que, el transmisor estimula al receptor a enviar una trama corta, de manera que las estaciones cercanas puedan detectar esta transmisión y eviten ellas mismas de hacerlo durante la trama siguiente de datos.

Ejemplo: A comienza por enviar una trama `RTS` (*Request to Send*) a B. Esta trama corta (30 bytes) contiene la longitud de trama de datos que seguirá posteriormente. Entonces B contesta con una trama `CTS` (*Clear to send*). La trama contiene la longitud de los datos (copiado de la trama `RTS`). A la recepción de la trama `CTS`, A comienza a transmitir.

Cualquier estación que escuche el `RTS` está lo suficientemente cerca de A y debe permanecer en silencio durante el tiempo suficiente para que el `CTS` se transmita de regreso a A sin conflicto. Cualquier estación que escuche el `CTS`evidentemente está lo suficientemente cerca de B y debe permanecer en silencio durante el siguiente tiempo de transmisión de datos, cuya longitud puede determinar examinando la trama `CTS`.

Nuevamente esto no garantiza la ausencia de colisiones, pero reduce su probabilidad (por ejemplo B y C pueden intentar enviar una trama `RTS` al mismo tiempo).

MACAW

es una mejora de `MACA` que agrega los siguientes cambios:

- Agrega un ACK tras cada trama de datos exitosa.
- Agrega detección de portadora (CSMA/CA).
- Ejecuta el algoritmo de *exponential backoff* por separado para cada flujo de datos (no por estación).
- Agrega un mecanismo para que las estaciones intercambien información de congestionamiento.

4.8.2 802.11 MAC

En una wireless LAN no podemos usar los mecanismos vistos de *collision detection* porque:

- Se necesita una radio *full duplex*.
- *Hidden terminal problem.*
- *Exposed terminal problem.*

Es por esto que la IEEE 802.11 MAC define su propio método de acceso al medio. Involucra dos métodos:

- **DCF** (*Distributed Coordination Function*): es el mecanismo base.
- **PCF** (*Point Coordination Function*): opcional. Permite agregar prioridades (`PIFS` en vez de `DIFS`)

DCF

DCF (*Distributed Coordination Function*) está basado en `CSMA/CA` con rotación de *backoff window*. Consiste en la política de *"listen before talk"*.

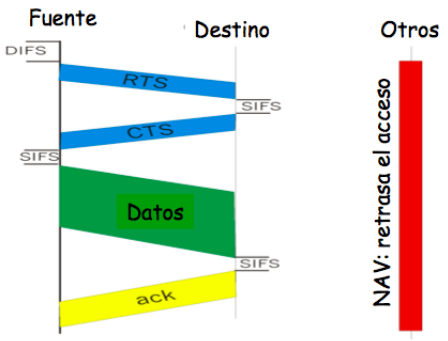
Cuando el emisor quiere transmitir, escucha el canal. Si lo detecta vacio por DIFS segundos, transmite la trama completa. Si detecta que el canal está ocupado, entonces hace *backoff* binario. El receptor, una vez que recibe la trama hace lo mismo para enviar el `ACK`.

Esta técnica tiene el problema de que desperdicia demasiado ancho de banda y no resuelve el problema de la terminal oculta.

Para solucionar eso se agrega el intercambio **RTS-CTS**:

- Un nodo que quiere transmitir escucha el canal durante un determinado período de tiempo (**DIFS**).
- Si está ocupado, hace *backoff* binario.
- Si está libre, envía un **RTS** (*Request To Send*).
- El nodo de destino cuando recibe un **RTS** espera otro período de tiempo (**SIFS**).
- Si el canal sigue libre, le responde con un **CTS** (*Clear To Send*).
- Si el canal no está libre (porque, por ejemplo, está recibiendo datos de una estación oculta para el emisor) le envía un **RxBusy**.
- Una vez que el emisor recibe la **CTS**, también debe esperar **SIFS** para poder comenzar a transmitir.
- El receptor también debe esperar **SIFS** para enviar el `ACK`.

Los paquetes `RTS` pueden involucrar el tamaño de la transmisión, que el receptor devuelve en el `CTS` (informando a las terminales ocultas que no manden por ese tiempo).



4.8.3 Frame Format en 802.11

- Utiliza direcciones de 48 bits.
- Utiliza un CRC de 32 bits para validar.
- Dentro del campo de control:
  - Hay un campo que indica si es traza, `CTS` o `RTS`.

4.8.4 Redes de espectro disperso

La idea es expandir la señal a través de un ancho de banda mayor que el mínimo requerido para transmitir con éxito. Esto evita concentrar la potencia sobre una única y estrecha banda de frecuencia, lo que puede dar lugar a interferencia catastrófica (accidental o deliberada), además de que se puede interceptar fácilmente la señal si usa una frecuencia constante.

Existen distintos métodos de transmisión:

FHSS

FHSS (*Frequency Hopping Spread Spectrum*) es un método de transmisión de señales que cambia rápidamente entre canales de frecuencia, usando una secuencia pseudoaleatoria compartida entre emisor y receptor.

Ventajas:

- Es resistente a interferencias de onda angosta.
- Es segura: es difícil de detectar porque aparenta ser un incremento en el ruido de fondo. Si no se conoce la secuencia pseudoaleatoria es difícil de interpretar aún si se pincharon todos los canales.
- Puede compartir bandas de frecuencia con otras transmisiones ofreciendo mínima interferencia.

Desventajas:

- Complicado de implementar.
- Complicado de mantener sincronizado emisor con receptor.

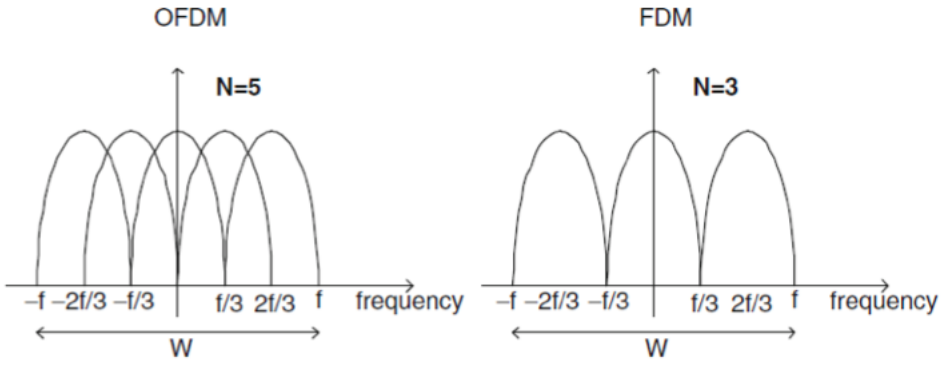
OFDM

OFDM (*Orthogonal Frequency Division Multiplexing*) es un método de multiplexación que consiste en enviar un conjunto de ondas portadoras de diferentes frecuencias.

Usa un número de señales ortogonales para que envíen datos en forma paralela por varios canales. Cada carrier se modula con una técnica convencional (`QAM` o `PSA`) y no necesariamente la misma.

`LTE` usa `OFDM`.

Su principal ventaja es la resistencia a condiciones adversas en el canal (atenuaciones, etc) así como `ISI` (*intersymbol interference*). Pero también tiene problemas de sincronización emisor-receptor.



5 Nivel de Red

Un **switch** de nivel de red es un "appliance" que conecta enlaces para formar redes más grandes. Su objetivo es lograr que la mayor cantidad de paquetes que entren al *switch* vayan a la salida apropiada.

Para eso cuenta con múltiples entradas y múltiples salidas. Cuando recibe un frame por un puerto de entrada, selecciona un puerto de salida utilizando una dirección que trae el *header* del paquete. Los frames pueden ser de longitud variable o fija.

Lo importante de los switches es que permiten construir redes **escalables**: agregar nuevos hosts a un switch no necesariamente impacta negativamente en la performance.

5.1 Conmutación de datagramas

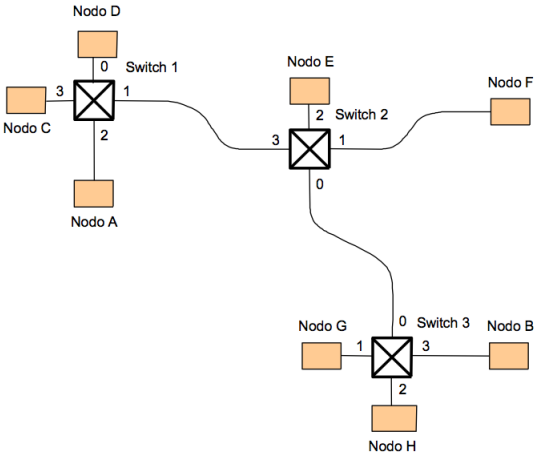
La **conmutación de datagramas** es un modelo **no orientado a conexión** en el que cada nodo puede enviar un paquete en cualquier momento. Los paquetes se envían en forma independiente y deben llevar toda la información necesaria para ser routeados hasta destino (con lo cual el *overhead* puede ser notable).

Para esto, cada switch tiene una **tabla de *forwarding*** en la que se indica, para cada destino, por qué puerto se debe reenviar el datagrama.

Analogía: sistema postal

Cada switch mantiene una tabla de *forwarding* (*routing*)

Tabla de conmutación para el switch 2	
Destino	Puerto
A	3
B	0
C	3
D	3
E	2
F	1
G	0
H	0



El modelo de datagrama es *best-effort*. No se ocupa de que los paquetes lleguen a destino correctamente, ni que lleguen en orden. Se pueden perder o desordenar paquetes. Incluso

pueden entregarse paquetes duplicados. Si pasa, habrá alguna abstracción de nivel superior que se ocupe de manejarlo.

No se debe esperar un `RTT` para establecer la conexión, con lo que el inicio es más rápido (un nodo puede mandar datos en cuanto esté listo). Además, como los paquetes son independientes, pueden tomar diferentes caminos para evitar sobrecargar los enlaces y nodos que estén fallutos.

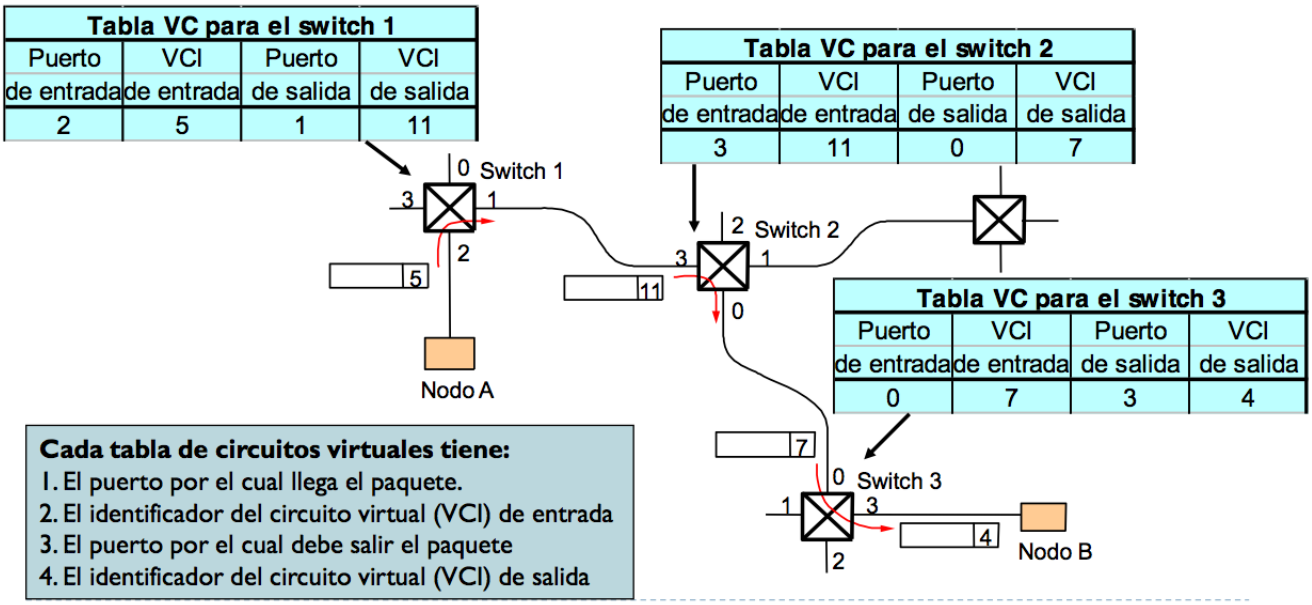
5.2 Conmutación orientada a conexión

La **conmutación orientada a conexión** requiere una fase para establecer la conexión antes de poder comenzar a transmitir datos y luego una de finalización de conexión.

El establecimiento del a conexión define un camino por el que van a circular los paquetes de la conexión, por lo que los paquetes no necesitan tener toda la información de cómo llegar a destino. Todos los paquetes utilizan siempre el mismo circuito, lo que garantiza que lleguen en orden.

Cada *switch* mantiene una tabla de `VC` (*virtual circuit*) que consta de:

- El puerto por el cual llega el paquete.
- El identificador del circuito virtual (`VCI`) de entrada.
- El puerto por el cual debe salir el paquete.
- El identificador del circuito virtual (`VCI`) de salida.



5.2.1 Tipos de conexiones

- Las conexiones permanentes las define y finaliza el administrador de la red. Después de creado el circuito virtual ya pueden mandar datos.
- Cuando un nodo desea enviar datos a otro, envía un mensaje solicitando una conexión. Este mensaje sí contiene toda la información necesaria para llegar a destino. Cuando el mensaje llega a destino, el destinatario le responde aceptando la conexión. A medida que va pasando por los switches, estos van "generando estado" para la conexión. O sea, creando las entradas en sus respectivas tablas de circuitos virtuales. Algo análogo ocurre para terminar la conexión.

5.3 Datagramas vs. Circuitos

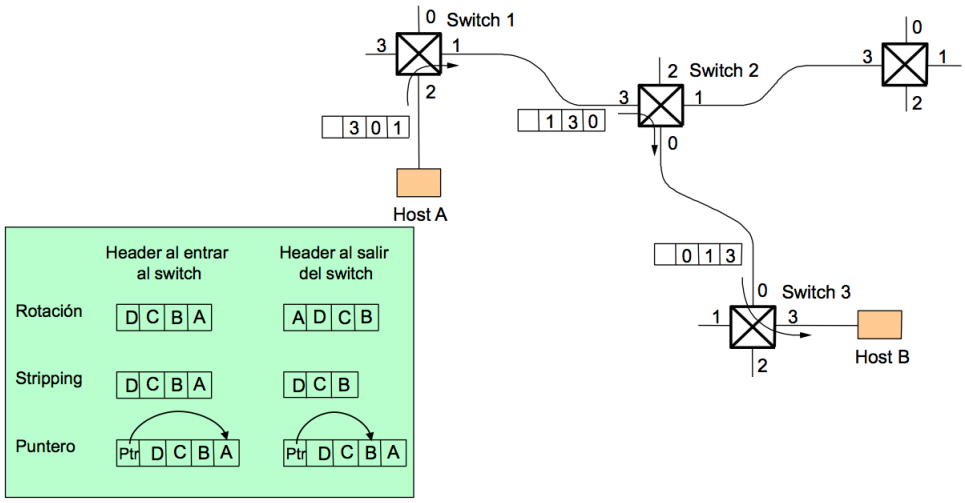
Datagramas	Conexiones
Los paquetes van por donde quieren	Los paquetes de la conexión van todos por el mismo camino
Comienzo inmediato	Esperar 1 `RTT` para comenzar a transmitir
Paquetes pesados	Paquetes livianos
Paquetes llegan desordenados	Paquetes llegan ordenados
Se banca que se caiga algún enlace/nodo	Si se cae un nodo de la conexión, se cae la conexión
No se permite reservar recursos	Permite reservar recursos en los switches

5.3.1 Source Routing

En la conmutación ***source routing***, toda la información sobre la topología de la red que se necesita para conmutar los paquetes es proporcionada por el nodo origen.

Existen varias formas de implementarlo:

- Rotación.
- Stripping.
- Pointer.



La conmutación ***source routing*** se puede utilizar en redes orientadas a datagramas para especificar un camino por el que se quiere que circule un paquete. Esto puede ser importante por motivos de seguridad. Pero también puede ser usado en redes orientadas a conexión.

5.4 Internet Protocol

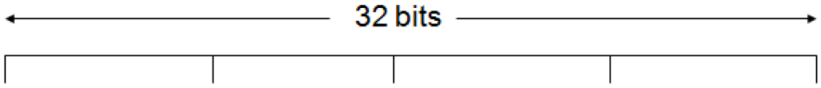
El ***internet protocol*** (`IP`) es un protocolo **de capa 3** basado en datagramas. Esto implica que comparte muchas de sus características:

- Best-effort.

- Los paquetes se pueden perder.
- Los paquetes pueden llegar fuera de orden.
- No hay cota para el tiempo de entrega.

5.4.1 Header IP

La cabecera de un datagrama IP contiene información que deben interpretar los routers. El tamaño de la cabecera es normalmente de 20 bytes, pudiendo llegar a 60 si se utilizan los campos opcionales.



Versión	Lon.Cab.	DS	Longitud total			
Identificación			□	DF	MF	Desplaz.fragmento
Tiempo de vida		Protocolo	Checksum			
Dirección de origen						
Dirección de destino						
Opciones (de 0 a 40 bytes)						

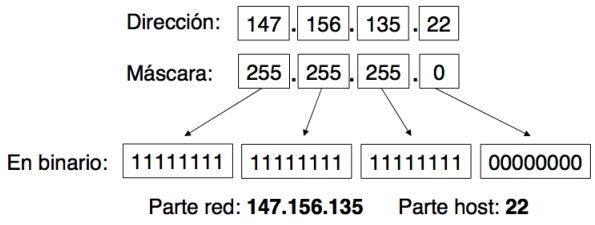
Notas:

- El checksum es de la cabecera, no de todo el datagrama.
- Un datagrama IP (incluyendo la cabecera) tiene un tamaño máximo de 65.535 bytes.
- El campo protocolo determina el uso que tiene el datagrama. Hay muchas opciones:
  - 1: ICMP.
  - 6: TCP.
  - 17: UDP.
  - 89: OSPF.

5.4.2 Direcciones IP

Las direcciones IP tienen 32 bits y están compuestas de dos partes, la parte **red** (parte alta) y la parte **host** (parte baja). La longitud de cada una de las partes se indica mediante un parámetro denominado **máscara de red**.

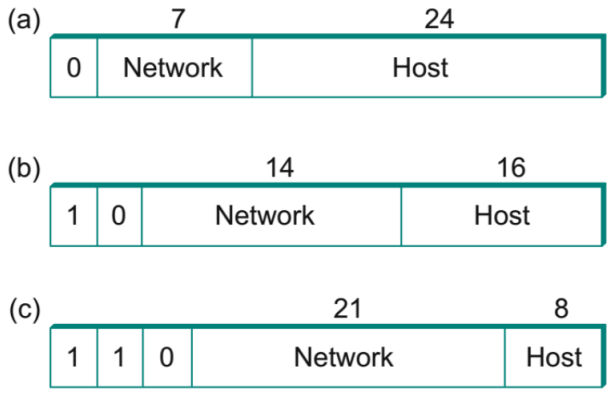
La máscara también tiene una longitud de 32 bits y no aparece en los paquetes sino que sólo se especifica en las interfaces y rutas.



Hay tres tipos de direcciones IP:

- Clase **A** ⇒ 16 millones de IPs.
- Clase **B** ⇒ 65534 IPs.
- Clase **C** ⇒ 254 IPs.

(En realidad hay más, pero estos 3 son los más usados).



Direcciones especiales

La primer y última dirección de red de una máscara se reservan para uso especial. No pueden ser asignadas a ningún host.

- La primera es para la **dirección de la red**.
- La segunda es para la **dirección de broadcast**.

Rangos reservados

Existen tres rangos de IP que no pueden ser usados para IPs públicas (RFC 1918). O sea las direcciones incluidas en esos rangos no pueden aparecer en Internet

- 10.0.0.0 – 10.255.255.255/**8**
- 172.16.0.0 – 172.31.255.255/**12**
- 192.168.0.0 – 192.168.255.255/**16**

5.4.3 Fragmentación

`IP` debe poder funcionar sobre cualquier red subyacente. Como cada tecnología tiene distintos `MTU` (*maximum transmission unit*), es necesario que los datagramas IP se adapten para respetar esto.

Es por esto que IP define el concepto de **fragmentación**: si un router recibe un datagrama que debe enviar a una red en la que $MTU <$ tamaño(*datagrama*), entonces se fracciona el datagrama en varios, colocando valores particulares en el *header* para su posterior reensamblado del lado del destinatario.

Todos los fragmentos que componen un mismo datagrama partido tienen el mismo identificador. De hecho, comparten toda la cabecera excepto por los campos *MF (more fragments)* y *desplazamiento del fragmento*.

La unidad básica de fragmentación es 8 bytes. Los datos se reparten en tantos fragmentos como haga falta, todos múltiplos de 8 bytes (excepto el último).

5.4.4 IP Forwarding

Dada una dirección de destino P y un prefijo de red N, el algoritmo de forwardeo de los routers IP es:

if *(N está concetado directamente a mi)* **then**
  Reenviar a esa interfaz
else
  if *(N está en mi forwarding table)* **then**
    Reenviar datagrama al nextHop que figura en la tabla
  else
    Reenviar datagrama al default router
  end
end

Algorithm 1: Routeo IP

5.4.5 VLANs

Una **VLAN** (*virtual LAN*) es un método para crear redes lógicas independientes dentro de una misma red física. Varias `VLAN` pueden coexistir en un único conmutador físico o en una única red física.

Existen dos tipos de `VLAN`s:

Las `VLAN`s pueden ser **estáticas** o **basadas en puerto** se crean mediante la asignación de puertos de un *switch* a la `VLAN`.

En las `VLAN`s dinámicas, la asignación se realiza dinámicamente en función de propiedades tales como la dirección `MAC` o autenticación de usuario.

¿Para qué usar VLANs?

- Para reducir el tráfico *broadcast* en una LAN muy grande.
- Para conectar redes locales remotas de una misma organización.
- Para poder dividir la red en zonas con distinto nivel de seguridad con un firewall.
- Para separar redes de servicio, backbone, interna, etc. (por ejemplo, un ISP).

5.5 X.25

Norma de la `ITU-T` para la interface entre los host y la red de conmutación de paquetes. Define 3 capas:

- Física: interface en el enlace entre el nodo y la estación. Ambos extremos son distintos.
- Enlace: provee transferencia confiable de datos sobre el enlace enviando secuencias de tramas.
- Paquete: provee conexiones lógicas (*virtual circuits*) entre los usuarios.

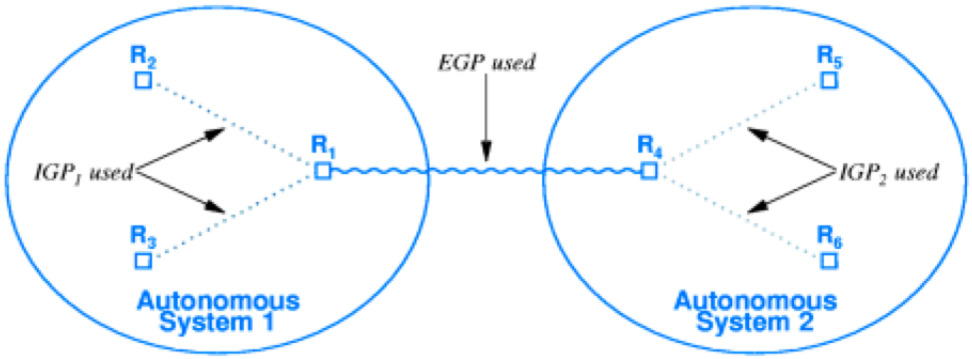
Aspectos claves incluyen:

- Paquetes de control de llamadas, señalización dentro de banda.
- Multiplexaje de circuitos virtuales en el nivel 3.
- Las capas 2 y 3 incluyen control de flujo y de errores.

5.5.1 Algoritmos de ruteo

Los algoritmos de routeo se pueden separar en dos categorías:

- Ruteo **interno** (`IGP`, *Internal Gateway Protocol*): tienen como dominio a un sistema autónomo.
- Ruteo **externo** (`EGP`, *External Gateway Protocol*): routean entre sistemas autónomos.



Forwarding vs Routeo

- **Reenvío/Forwaridng**: proceso de seleccionar un puerto de salida basado en la dirección de destino y las tablas de ruteo.
- **Ruteo**: proceso mediante el cual se construyen las tablas de ruteo.

Distance Vector

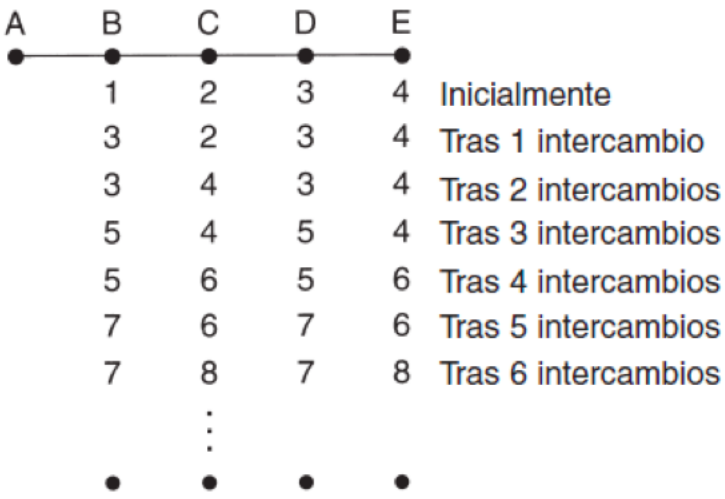
Los algoritmos de ruteo basados en *distance vector* están basados en el algoritmo de *Bellman-Ford* distribuido. Cada nodo mantiene una tabla de tuplas `(Destination, Cost, NextHop)`.

Para mantener actualizada la información de ruteo cada nodo intercambia información con sus vecinos directamente conectados. Estas actualizaciones ocurren periódicamente, pero también se fuerzan cuando hay algún cambio.

Las actualizaciones consisten de un par `(Destination, Cost)`. Cuando un nodo recibe una tupla de actualización, computa qué es más costoso: si mantener el camino que él tiene en su tabla de ruteo para el destino o si sumarle 1 hop al valor que me acaba de pasar mi vecino e ir a través de él. Si ocurren varios *time-out* se borran las rutas en los routers.

Count to infinity

El **problema de conteo a infinito** es una situación que puede darse con los algoritmos de *distance vector*. El problema tiene su origen en que si un nodo A le informa a otro nodo B que tiene un camino a C de costo n, B no tiene forma de saber si él pertenece a ese camino.



Si A se cae, B tiene que borrar esa información de su tabla. Como ahora B no sabe llegar a A, cuando le llega un mensaje de C diciendo "si vas a través mío, podés llegara a A en dos pasos", actualiza tu tabla a 3. Esta información es incorrecta, porque la forma de llegar de C a A **es a través de** B. Luego empieza un ciclo en el que continuamente se van aumentando las distancias sin parar.

Soluciones

Para solucionar el *count to infinity problem* se pueden usar dos heurísticas:

- **Split horizon**: cuando un nodo A recibe información de un nodo B y compute la nueva ruta, envía la información a todos sus vecinos **excepto a** B.
- **Poison reverse**: cuando un nodo A detecta que una de sus rutas conectadas se cayó, va a "envenenar" la ruta asignándole ∞ a su distancia y avisándolo a sus vecinos. Cuando uno de sus vecinos reciba esto, va a romper la regla de *split horizon* y anunciarle a todos sus vecinos (incluido A) de la ruta caida.

Estas heurísticas no funcionan demasiado bien en la práctica.

Link State

A diferencia de *distance vector*, donde cada nodo conoce sólo la información de sus vecinos, en ***link state*** todos los nodos conocen toda la topología de la red, con sus respectivos costos. Esto permite que se utilice una versión modificada del algoritmo de *Dijkstra* para calcular el camino mínimo.

Para que todos los nodos conozcan la topología entera, los nodos comparten su conocimiento no con los vecinos sino con todos los nodos de la red mediante una técnica llamda **reliable flooding**. No se envía la tabla completa con la que cuenta cada nodo sino sólo sobre los enlaces directamente conectados.

Para transmitir, cada nodo hace un paquete del estado del enlace **LSP** (*Link State Packet*). Este paquete contiene:

- *id* del nodo que creó el `LSP`.
- Costo del enlace a cada vecino directamente conectado al nodo.
- Número de secuencia del paquete.
- `TTL` (*Time To Live*) del paquete.

Reliable Flooding

La técnica de **reliable flooding** permite asegurar que la información que envía un nodo llegue a todos los nodos de la red.

- Cada nodo almacena el `LSP` más reciente (número de secuencia más alto) de cada uno de los otros nodos.
- Cuando un nodo recibe un `LSP` se lo reenvía a todos sus vecinos excepto al que me lo mandó.
- Se decrementa periódicamente el `TTL` de cada LSP en la tabla.
  - Si el `LSP` que recibe tiene `TTL`=0, lo descarta.
- Periódicamente cada nodo genera y envía un nuevo `LSP` (aumentando el número de secuencia).
- Cuando se reinicia el router, se setea el número de secuencia a 0.

Shortest Path Routing

Forward Search Algorithm es una versión modificada del algoritmo de Dijkstra que se usa para calcular los caminos mínimos en *link state*. Difiere del algoritmo de Dijkstra en que permite ir calculando el camino paulatinamente conforme lleguen los `LSP` de los nodos.

El algoritmo mantiene dos listas de nodos: la de entradas **confirmadas** y la de **tentativas**, cada una de las cuales consta de una tupla `(Destination, Cost, NextHop)`.

Distance Vector vs Link State

	DISTANCE-VECTOR	LINK-STATE
Qué informa cada nodo?	* Su Tabla de Ruteo	* Estado de sus Enlaces
A quién pasa la infomación?	* Sólo a sus vecinos	* Inunda a toda la red
Algoritmo utilizado	* Bellman-Ford Distribuido	* Dijkstra
Datos utilizados	* Información de los vecinos	* Estado de Enlaces de cada nodo
Estructuras de Datos	* Tabla de Distancias * Tabla de Ruteo	* Tabla de Estado de Enlaces * Tabla de Ruteo
Caracteríticas	* Ciclos de Ruteo * Gran variedad de Algoritmos: * Merlin-Segall * Jaffe-Moss * Esquema OP * Diffusing Comp * Cheng * Cálculo Distribuido	* Visión Consistente de la Red * Gran uso de CPU y Memoria * Algoritmo Básico único * Cálculo Centralizado
Ejemplo de Protocolos de Internet	* RIP	* OSPF

Distance Vector	Link State
Más liviano (no hay algoritmos de grafos complejos)	Más pesado (calcular dijkstra en cada nodo)
Nodos bobos y baratos	Necesitás mucho almacenamiento y poder de cómputo en los nodos
Cada nodo transmite a sus vecinos lo que sabe respecto de toda la red (distancia a todos los nodos)	Cada nodo transmite a toda la red lo que sabe de sus enlaces vecinos (el estado de sus vecinos)
Más propenso a errores	Más estable
Mucho overhead	No genera tanto tráfico
Si se me cae un nodo puede entrar en count to infinity	Responde más rápido a cambios de topología
Escala peor	Escala mejor

Métricas

Las métricas `ARPANET` fueron diseñadas para medir la performance de un sistema de ruteo. En su versión original, `ARPANET` mide el número de paquetes encolados en cada enlace. No toma en cuenta la **latencia** ni el **ancho de banda**.

Más tarde apareció una segunda versión que *taggea* cada paquete entrante con su tiempo de llegada (AT) y registra su tiempo de salida (DT). Luego, cuando llega el `ACK` de ese paquete, realiza el cálculo:

```
Delay = (TiempoSalida - TiempoLlegada) + Transmit + Latency
```

donde `Transmit` y `Latency` son parámetros de la red.

RIP

RIP (*Routing Information Protocol*) es un protocolo de ruteo para redes `IP` basado en *vector-distance*. Utiliza cantida de hops de salto como medida de proximidad y heurísticas de *poison reverse* y *split horizon* para evitar el problema de *count to infinity*. Además, setea el número máximo de hops permitido como 15 (el 16 lo considera ∞).

Se dejó de usar por sus notables problemas de escalabilidad y *overhead*: los nodos `RIP` enviaban cada 30 segundos actualizaciones a sus vecinos. Aún si los *clocks* de los nodos comenzaban aleatoriamente, después de un tiempo se sincronizaban, generando un pico muy alto de uso de de la red (con su correspondiente saturación) cada 30 segundos.

OSPF

OSPF (*Open Shortest Path First*) es un protocolo de ruteo abierto para redes IP (disponible públicamente) basado en *link state*.

Al ser un protocolo de tipo *link state*, `OSPF` utiliza *reliable flooding* para comunicar a todos los nodos de la red los enlaces conectados a cada nodo. Lo particular de OSPF es que utiliza sus propios mensajes `IP`, que es mejor que comunicar por `TCP` o `UDP`.

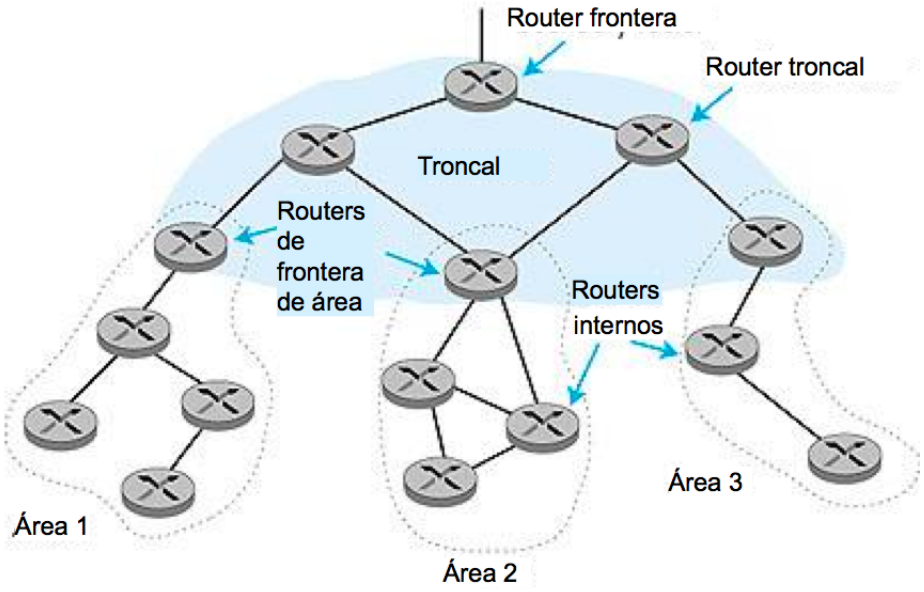
Todos los mensajes en `OSPF` van autentificados, evitando inserciones no autorizadas de terceros.

Además, permite que se mantengan dos caminos entre dos nodos si ambos tienen igual costo (cosa que no se puede en `RIP`).

Si bien no es usado en la práctica, `OSPF` en teoría soporta distintos `TOS` (*type of service*), que pueden utilizarse para distinguir enlaces de alta prioridad, de bajo coste, etc.

Jerarquía

La principal ventaja que provee `OSPF` consiste en su estructura jerárquica, lo que lo dota de una inmensa **escalabilidad**.



Como se ve en la imagen, los nodos se dividen en tres categorías:

- **Frontera de área**: "resumen" las distancias a las redes del mismo área, anuncian a otros routers de Frontera de área.
- **Troncal**: ejecutan ruteados de `OSPF` limitados al troncal.
- **Interno**:

Utilizando estas categorizaciones jerárquicas y algoritmos inteligentes, se puede reducir mucho la cantidad de mensajes necesarios para hacer el *reliable flooding*.

Además, reduce la cantidad de información que tiene que estar almacenada en un nodo, dado que cada nodo detalla la topología del área; sólo conoce la dirección (el camino más corto) a las redes de otras áreas.

Tipos de mensaje

`OSPF` soporta los siguientes tipos de mensaje:

- **Hello**: descubre quienes son los vecinos.
- **Link State Update**: proporciona los costos del emisor a su vecino.
- **Link State Ack**: confirma la actualización del estado del enlace.
- **Database Description**: anuncia qué actualizaciones tiene el emisor.
- **Link State Request**: solicita información de otro nodo.

EGP

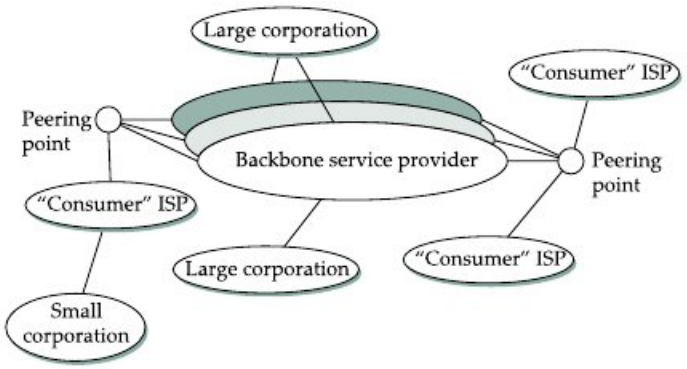
EGP (*Exterior Gateway Protocol*) era el algoritmo de ruteo usado en internet para conectar sistemas autónomos en su primer época, cuando aún estaba estructurada como árbol y no como grafo.

BGP

BGP (*Border Gateway Protocol*) es un protocolo de ruteo externo (entre sistemas autónomos) pasado en *distance vector*.

Hoy en día se usa **BGP-v4** que también sirve para interconectar sistemas autónomos pero más eficientemente.

Cada sistema autónomo tiene uno o más routers de borde y un portavoz que publica las redes locales, otras redes alcanzables e información de rutas.



Los autonomous systems se dividen en categorías:

- **Stub AS** (tiene una única conexión a otro AS): transporta sólo tráfico local. *Small corporation*, en el ejemplo.
- **Multihomed AS** (tiene conexiones a más de un AS): no transporta tráfico en tránsito. *Large corporation*.
- **Transit AS** (tiene conexiones a más de un AS): transporta ambos tráfico local y en tránsito. *Backbone providers*.

6 Nivel de Transporte

6.1 Transporte vs Red

El **nivel de transporte** se apoya sobre la capa de red, con lo que tiene que ocuparse de las consecuencias de que sea *best effort*:

- Llegan paquetes desordenados.
- Se pierden paquetes.
- Llegan paquetes con demoras arbitrariamente largas.
- Llegan paquetes duplicados.

El nivel de transporte debe lidiar con estos problemas porque debe proveer:

- Garantía de entrega de mensajes.
- Entrega de mensajes en el mismo orden que son enviados.
- Una copia por mensaje.
- Soporte para mensajes arbitrariamente largos.

- Control al receptor el flujo de datos del transmisor.
- Soporte para múltiples procesos de nivel de aplicación en cada máquina.
- Soporte para diferentes `RTT`.
- Soporte para potencialmente largos retardos en la red subyacente.
- Soporte para destinos de diferentes capacidades.

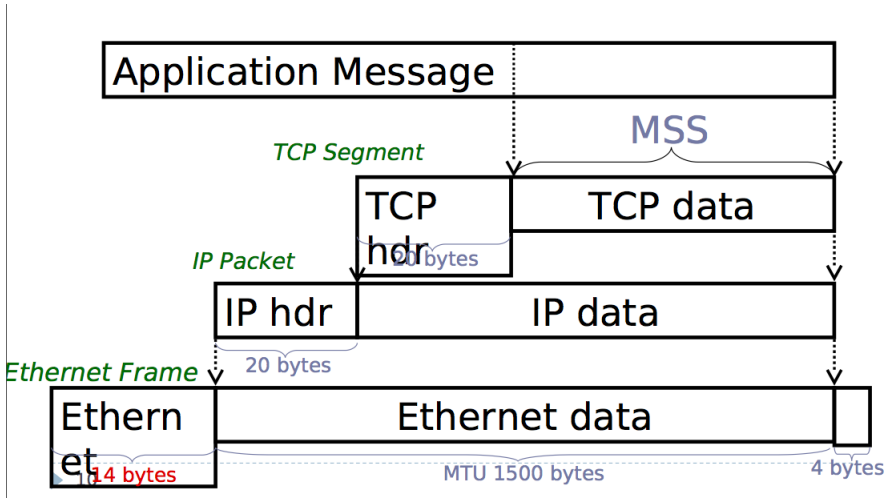
6.2 TCP

TCP (*Transmission Control Protocol*) es el protocolo de comunicación más usado en la actualidad y uno de los pilares de Internet.

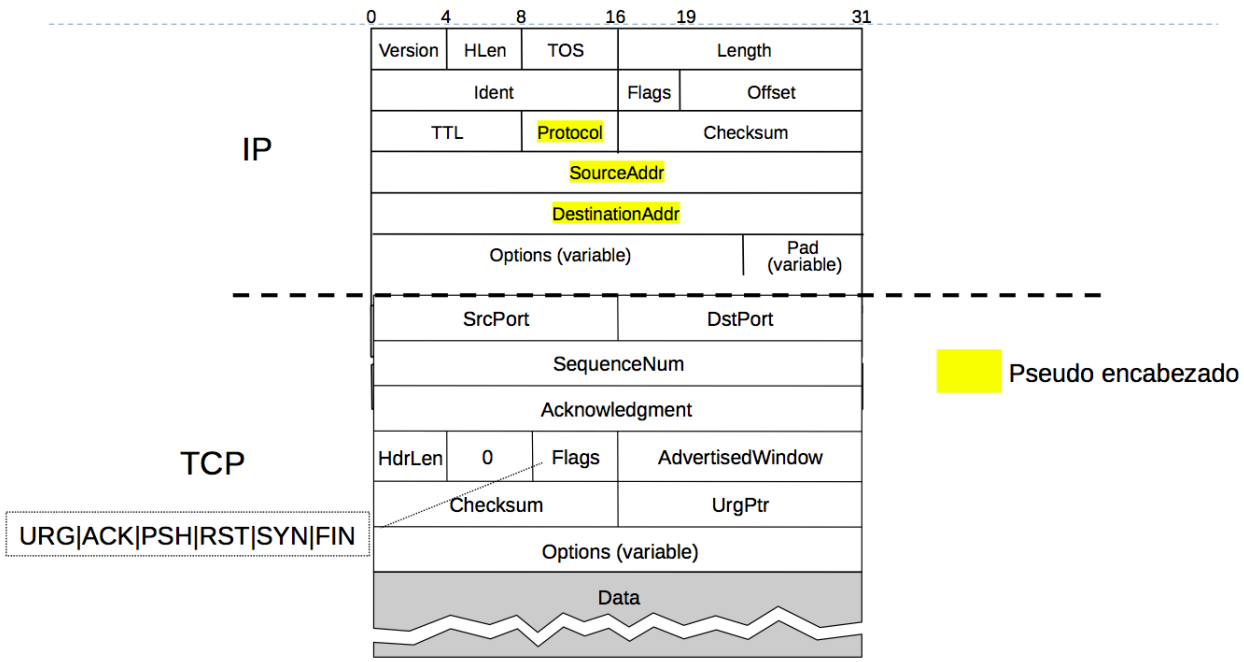
Algunas características generales son:

- **Orientado a conexión**: *3-way handshake* para establecer una conexión y *4-way handshake* para liberarlo.
- **Confiable**: establece una conexión lógica entre los *sockets*. Para eso usa:
  - `ACK`.
  - *Checksum*.
  - Números de secuencia.
  - Timeout.
  - **Control de flujo**: evita que un transmisor inunde a un receptor.
  - **Control de congestión**: evita que un transmisor sobrecargue a la red.
- Provee flujo de *bytes*:
  - La aplicación escribe *bytes*.
  - Bajo algún criterio, se compactan estos *bytes* en un segmento y los envía. Este criterio puede ser:
    - Se llega a `MSS` (*Maximum Segment Size*) *bytes*.
    - Ocurre un timeout.
    - La aplicación pide explícitamente pushear los datos.
  - La aplicación receptora lee *bytes*
- **Full duplex**: puede haber comunicación simultánea y bidireccional entre emisor y receptor.

6.2.1 Encapsulamiento



6.2.2 Frame TCP



donde

- `srcPort` y `dstPort` (16 bits): indican los puntos finales locales de la conexión.
- `sequenceNum` (32 bits): identifica el primer *byte* de los datos de aplicación que contiene el segmento `TCP`.
- `ACK` (32 bits): indica el siguiente byte que esperar del emisor. Implica la confirmación de todos los bytes de menor `sequenceNum`.

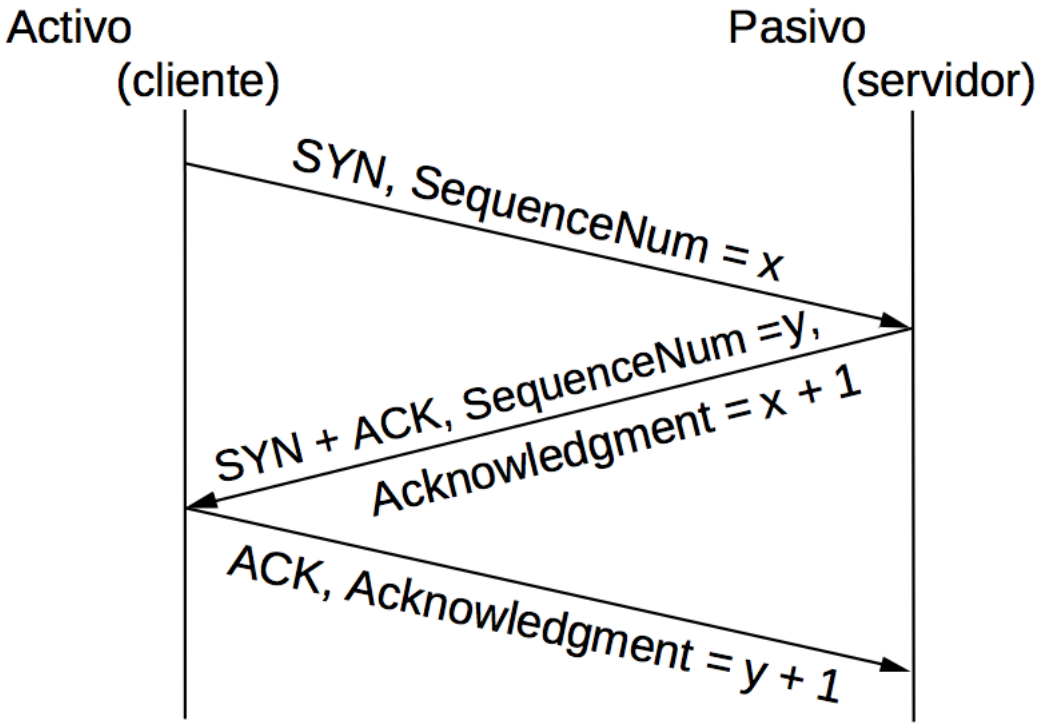
- `HdrLen` (4 bits): indica cuántas palabras de 32 bits están contenidas en el header. Es necesario porque el campo `options` tiene longitud variable.
- `Flags`:
  - `URG`: es 1 si el campo "puntero a urgente" está en uso.
  - `ACK`: es 1 si el número de reconocimiento es válido. Si vale 0, el paquete no contiene un reconocimiento, y entonces el campo número de reconocimiento es ignorado.
  - `PSH`: indica al receptor que debe entregar los datos a la aplicación inmediatamente y no bufferear.
  - `RST`: sirve resetear la conexión (por ejemplo si se cayó un host).
  - `SYN`: usado para establecer conexiones.
  - `FIN`: usado para liberar conexiones.
- `AdvertisedWindow`: se usa en control de flujo. Indica cuantos bytes pueden ser enviados comenzando desde el último byte reconocido.
- `Checksum`: se computa sobre el *pseudoheader* (`protocol + srcIpAddr + dstIpAddr`), todo el header de `TCP` y todos los datos del segmento.
- `UrgPtr`: se utiliza para indicar un desplazamiento en bytes a partir del número de secuencia actual en el que se encuentran los datos urgentes. Esta facilidad se brinda en lugar de los mensajes de interrupción.
- `Options`: diseñado para proveer una manera de adicionar facilidades extras no cubiertas por la cabecera regular. Se usan para extensiones de `TCP`.

6.2.3 Estableciendo la conexión

Cada conexión en `TCP` se identifica con la tupla `(srcIpAddr, srcPort, dstIpAddr, dstPort)`. El protocolo usa *sliding window* y control de flujo. El emisor transmite los datos con un cierto `seqNum` y el receptor devuelve `ACK` + `avertisedWindow`.

Para establecer una conexión, `TCP` utiliza un mecanismo llamado **three-way handshake**: antes de que un cliente intente conectarse con un servidor, deben primero establecer una conexión enviando tres mensajes:

- `SYN`: el cliente le envía al servidor un mensaje `SYN` y un determinado número de secuencia aleatorio.
- `SYN+ACK`: en respuesta, el servidor envía un paquete con `SYN+ACK` que tiene el `ACK` seteado en 1 + el número de secuencia recibido. En su número de secuencia envía otro número aleatorio.
- `ACK`: Finalmente el cliente envía un `ACK` de nuevo al servidor. El `ACK` debe ser el 1 + número de secuencia recibido en el paquete anterior y el número de secuencia 1 + el número generado por el cliente para el primer paquete.



6.2.4 Terminando la conexión

La terminación de una conexión de `TCP` puede realizarse de dos formas: *four-way handshake* o `three-way handshake`

6.2.5 Terminar con four-way handshake

Para finalizar una conexión con un *four-way handshake* cada nodo de la conexión debe terminarla independientemente: los 4 mensajes requeridos pueden verse como 2 grupos de 2 en los cuales cada uno cierra "su lado" de la conexión.

Cuando uno de los nodos (A) desea terminar su conexión con el otro nodo (B) se intercambian los siguientes mensajes:

- El nodo que desea cerrar (A) envía `FIN`.
- El nodo receptor (B) acusa recibo del mensaje enviando `ACK`.

En este punto la conexión se considera *half-open*. El nodo que envió el `FIN` no puede seguir enviando información por la conexión, pero sí debe seguir recibiendo, puesto que el otro lado no desea terminar su conexión todavía. Cuando B desea terminar, realizan el intercambio inverso:

- B envía `FIN`.
- A responde con `ACK`.

Luego, de que ambos intercambios hayan terminado, el nodo A debe esperar un cierto tiempo antes de cerrar la conexión y permitir reasignar su puerto. Esto está pensado para prevenir que si se reasigna el puerto demasiado rápido puedan llegar paquetes demorados de la conexión anterior y generar confusión.

6.2.6 Terminar con three-way handshake

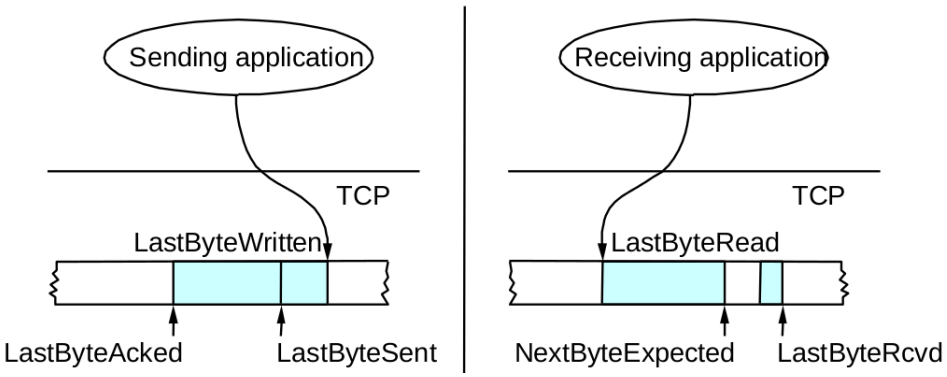
Finalizar una conexión con *three-way handhsake* es igual que con 4 mensajes, pero se combinan los dos mensajes consecutivos del mismo host en uno sólo (cuando a ese host no le interesa dejar la conexión en estado *half-open*). Luego, la secuencia de mensajes es:

- Una de las partes envía `FIN`.
- El otro nodo envía `FIN+ACK`.
- El nodo original envía `ACK`.

6.2.7 Sliding Window

Se implementa una variante de *sliding window* que garantiza **confiabilidad**, **orden** y fuerza el **control del flujo**.

- **Emisor**: `LastByteAcked` $\leq$ `LastByteSent` $\leq$ `LastByteWritten`
- **Receptor**: `LastByteRead` $<$ `NextByteExpected` $\leq$ `LastByteRcvd` $+ 1$



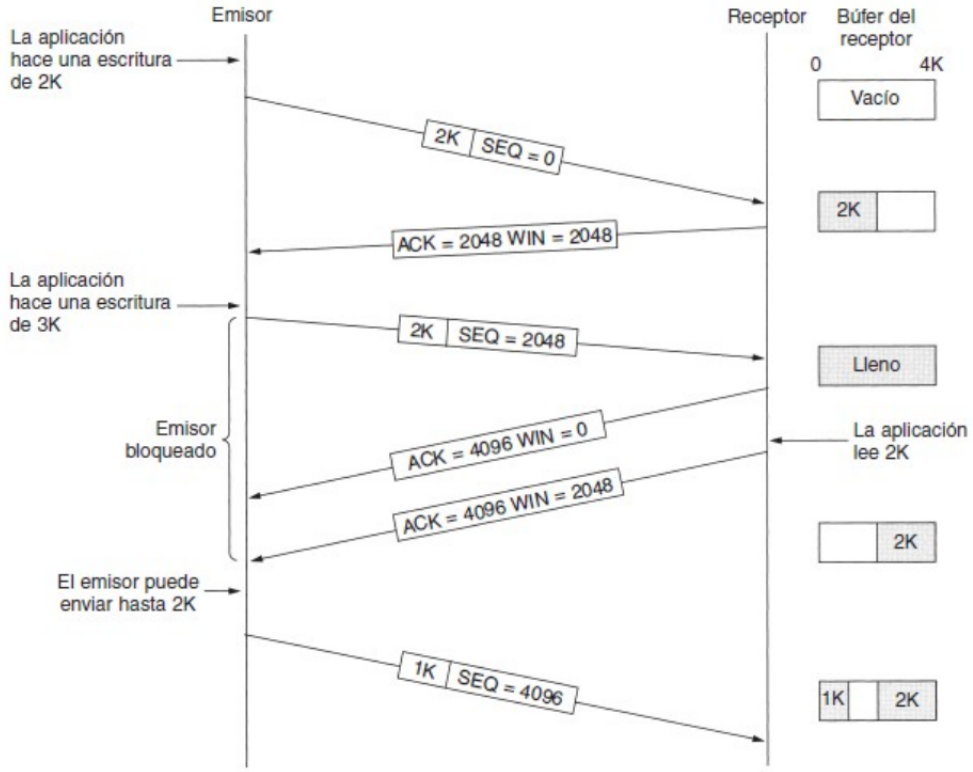
Para la parte de **control de flujo**, tanto el emisor como el receptor tienen *buffers* que utilizan para almacenar los mensajes de tamaño `MaxSendBuffer` y `MaxRcvBuffer` respectivamente.

Del lado del receptor, se debe cumplir que:

- `LastByteRcvd` - `LastByteRead` $\leq$ `MaxRcvBuffer`.
- `AdvertisedWindow` $=$ `MaxRcvBuffer` - ((`NextByteExpected` - 1) - `LastByteRead`)

Del lado del transmisor,

- `LastByteSent` - `LastByteAcked` $\leq$ `AdvertisedWindow`
- `EffectiveWindow` $=$ `AdvertisedWindow` - (`LastByteSent` - `LastByteAcked`)
- `LastByteWritten` - `LastByteAcked` $\leq$ `MaxSendBuffer`
- Bloquear al transmisor si (`LastByteWritten` - `LastByteAcked`) + #bytesQueElTransmisorQuiereEscribir $>$ `MaxSenderBuffer`



6.2.8 Adaptative Retransmission

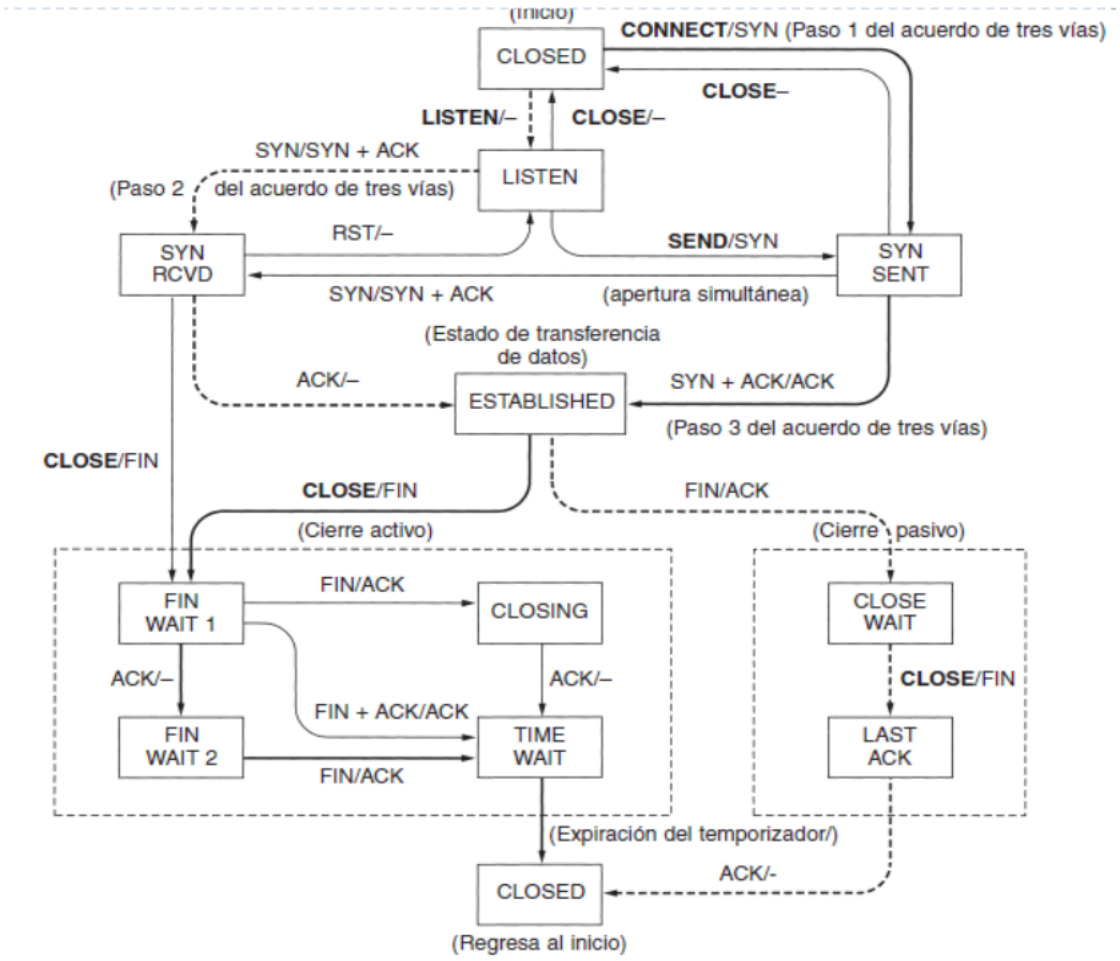
`TCP` garantiza que los mensajes llegan a destino. Para esto, define un `RTO` (*retransmission timeout*) luego del cual si no se recibió el `ACK` para un determinado paquete, se reenvía.

Lo novedoso de `TCP` es que ese *timeout* no es un valor fijo sino que se calcula en funciión del `RTT` esperado entre dos nodos de la conexión.

Hubo 4 fórmulas:

- La histórica: medir el RTT por paquete y hacer un promedio ponderado.
- Karn/Partridge: no cuentan los RTT de las retransmisiones, pero después de cada retransmisión, duplican el *timeout*.
- Jacobson/Karels: considera la desviación estándar.
- RFC6298: define cotas para cuán agresivo puede ser un algoritmo de cálculo de `RTO` (aunque se pueden programar más conservadores). El emisor mantiene dos variables `SRTT` (*smoothed rtt*) y `RTTVAR` (*rtt variation*). A medida que van llegando valores se computa:
  - `RTTVAR` $= (1 - \beta) \times$ `RTTVAR` $+ \beta \times \|$`SRTT` - R'$\|$
  - `SRTT` $= (1 - \alpha) \times$ `SRTT` $+ \alpha \times$ R'

6.2.9 Estados de TCP



Máquina de estados finitos de administración de conexiones TCP. La línea continua gruesa es la trayectoria normal de un cliente. La línea punteada gruesa es la trayectoria normal de un servidor. Las líneas delgadas son eventos poco comunes. Cada transición está indicada por el evento que la ocasiona y la acción resultante, separada por una diagonal.

Como se ve en la imagen, una conexión de `TCP` puede estar en muchos estados:

- **Closed**: no hay conexión activa ni pendiente.
- **Listen**: el servidor espera una llamada.
- **SYN Received**: llegó solicitud de conexión, espera `ACK`.
- **SYN Sent**: la aplicación comenzó a abrir una conexión.
- **Established**: estado normal de transferencia.
- **FIN Wait 1**: la aplicación dijo que ya terminó.
- **FIN Wait 2**: el servidor aceptó liberar.
- **Timed Wait**: el primero espera a que mueran todos los paquetes.
- **Closing**: ambos lados intentaron cerrar simultáneamente.

- **Close Wait**: el otro lado inició una liberación.
- **Last ACK**: espera a que todos los paquetes mueran.

6.2.10 Algoritmo de Nagle

El **algoritmo de Nagle** intenta reducir la cantidad de paquetes `IP` que deben ser enviados a la red.

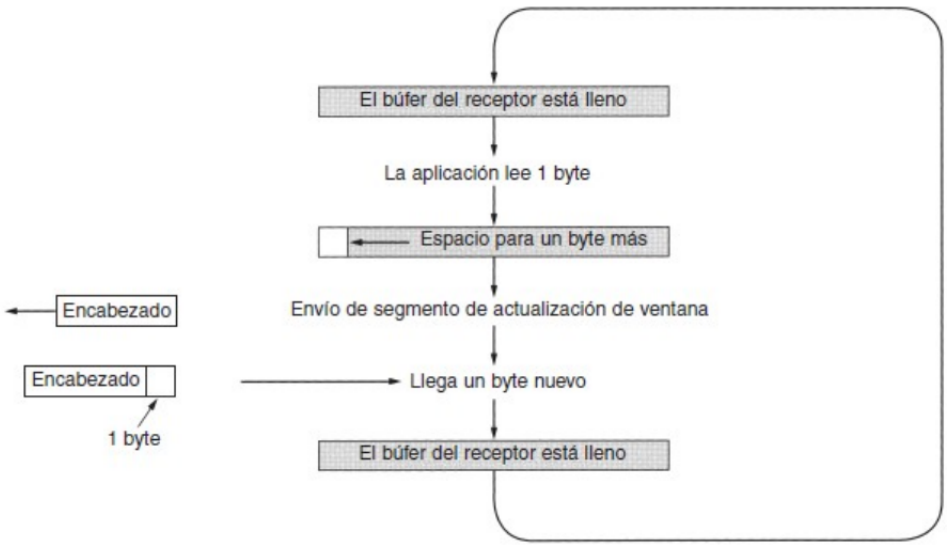
if *(tamaño de ventana y datos disponibles ≥ MSS)* **then**
  Enviar un segmento lleno
else
  if *(Los datos en vuelo están in reconocer)* **then**
    Bufferear el dato nuevo hasta que llegue el `ACK`
  else
    Enviar datos nuevos ahora
  end
end

Algorithm 2: Algoritmo de Nagle

Si la aplicación receptora lee muy despacio, el buffer se llena y anuncia un tamaño de ventana de lectura = 0. Luego, el emisor deja de transmitir. Cuando el receptor lea vuelve a aumentar la ventana y manda un `ACK` anunciándolo. ¿Qué pasa si se pierde ese `ACK`? el emisor no puede enviar datos y el receptor no sabe que se perdió el `ACK`.

Para solucionar esto, `TCP` implementa ***persistent timer***: periódicamente se envían *window probes* con 1 byte de dato.

Esto puede llevar al problema conocido como ***Silly Window Sindrome***: si la aplicación receptora lee de a 1 byte, el window probe puede estar llenándome el espacio de ese byte mientras el receptor lee.



La solución de Clark al problema del *silly window* consiste en no enviar un aviso de ventana para 1 byte, sino en esperar a tener una capacidad de buffer considerable (el mínimo entre el MSS y la mitad del buffer).

6.2.11 Análisis de TCP

$$T = \text{throughout} = \frac{\frac{paquetes\ transmitidos}{ciclo}}{\frac{tiempo}{ciclo}}$$

$$BW = \text{ bandwidth } = \frac{\frac{data}{ciclo}}{\frac{tiempo}{ciclo}}$$

6.3 Congestión

Decimos que hay **congestión** cuando hay un estado sostenido de sobrecarga de una red donde la demanda de recursos (*buffers* y enlaces) se encuentra al límite o excede la capacidad de los mismos. Las manifestaciones de la congestión son: pérdida de paquetes (por saturación de *buffers*) y retardos muy largos. Esto da idea de algunas métricas que pueden usarse:

- Porcentaje de paquetes descartados por sobrecarga de buffers.
- Longitud media de una cola.
- Cantidad de paquetes que generan timeout y son retransmitidos.
- Promedio de demora de un paquete.
- Desviación estandar de la demora de un paquete.

> **OBSERVACIÓN**: **Control de congestión $\neq$ control de flujo**. El control de congestión es una cuestión global que involucra a todos los hosts y routers. Evita que los transmisores sobrecarguen el interior de una red.
> El control de flujo contra tráfico punto a punto entre un receptor y un transmisor y evita que los transmisores sobrecarguen a receptores lentos.

6.3.1 Causas de congestión

Se pueden nombrar muchas:

- Procesadores lentos.
- Problemas con software de ruteo.
- Partes del sistema (una linea vieja y muchas nuevas).
- Congestión (la congestión tiende a realimentarse).
- Políticas de Transporte (retransmisión, almacenamiento fuera de orden, control de flujo confirmación de recepción).
- Políticas de Red (circuitos vs datagramas, encolamiento, política de descarte, TTL).
- Políticas de Enlace de datos (retransmisión, confirmación de recepción).

6.3.2 Control de congestión

Definimos entonces al **control de congestión** como el esfuerzo hecho por los nodos de la red para prevenir o responder a sobrecargas de la red que conducen a pérdidas de paquetes.

Se pueden hacer 4 cosas para evitar la congestión:

- Sobredimensionar.
- Diseñar.
- Controlar, evitar.
- Pre-asignar recursos (cuando se pueda, ej. redes orientadas a conexión). Problema: subutilización de recursos.

Queremos que la red sea usada eficientemente ($\frac{throughput}{retardo}$) y equitativamente (fórmula de Jain).

6.3.3 Congestion control vs Congestion avoidance

Congestion control es reactivo. Detecta la presencia de congestión y hace algo al respecto.
Congestion avoidance es proactivo.

6.3.4 Lazo abierto vs Lazo cerrado

De acuerdo con la taxonomía de Yang y Reddy, los algoritmos de control de congestión se categorizan en:

- Lazo abierto: no hay retroalimentación hacia el controlador para que éste pueda ajustar la acción de control.
- Lazo cerrado: usan la retroalimentación desde un resultado final para ajustar la acción de control en consecuencia.
  - Realimentación implícita: la red *dropea* paquetes cuando ocurre la congestión. Las fuentes pueden detectar esto (por *timeouts*, `ACK`s duplicados, etc) y hacer algo al respecto. Su implementación es relativamente simple, porque sólo involucra complejidad en los emisores.
  - Realimentación explícita: los distintos componentes de la red deben proveer indicación explícita de congestión a las fuentes. Se usa *packet marking*. Provee información más precisa. Su implementación es más complicada porque involucra cambios en las fuentes además de en los nodos y hay cuestiones de compatibilidad.

RED y FRED

`RED` (*Random Early Detection*) es un algoritmo de lazo cerrado con realimentación implícita que tiene como objetivo evitar la congestión y mantener el tamaño medio de las colas en niveles bajos. No requiere que los routers mantengan ninguna información del estado de las conexiones.

En lugar de que los nodos descarten paquetes cuando se les llena la cola, si la cola está más de un cierto porcentaje (umbral) llena, descartan un nuevo paquete con una probabilidad p.

`RED` tiene problemas de imparcialidad con las conexiones de baja velocidad: cuando se alcanza el umbral, `RED` descarta paquetes aleatoriamente sin considerar si se trata de una conexión que está usando más de su cuota de recursos o menos. Es por esto que se implementó una mejora que se convirtió en `FRED` (*Flow Random Early Detection*) en el que se controla a los usuarios "bandidos": mantiene umbrales y tasas de ocupación del buffer para cada flujo activo. La desventaja notoria de esto es que necesita guardar información por cada flujo, produciendo un alto costo en los routers (debe mirar la fuente, destino, puertos y protocolo del paquete por cada paquete que llega).

Traffic shaping

Traffic shaping es un concepto de lazo abierto que intenta guiar la congestión forzando a los paquetes a transmitirse a una velocidad más predecible. Intenta evitar el problema que causa el tráfico de tipo *burst traffic*. Hay varios métodos que lo usan:

- Leaky bucket.
- Token bucket.

> Se define **marshalling** como el proceso de transformar la representación en memoria de un objeto a una forma apropiada para almacenamiento o transmisión.

7 Fuentes

- Slides de la cátedra de Teoría de las comunicaciones de Claudio Righetti.
- Apuntes de clase de Julián Sackmann.