

Direcciones IPv6

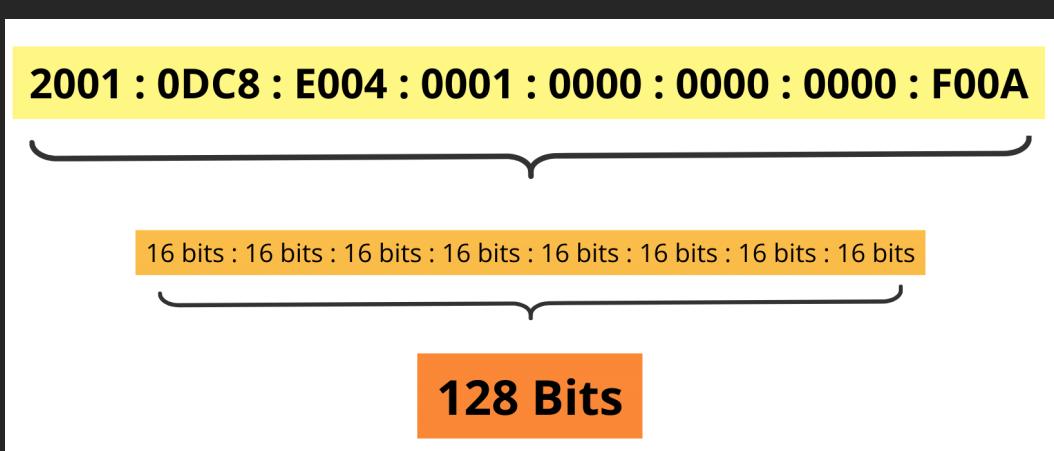
- El tamaño y el formato de la dirección IPv6 amplían la capacidad del espacio para direcciones.

El tamaño de las direcciones de IPv6 es de 128 bits.

- La representación preferente de direcciones de IPv6 es x:x:x:x:x:x:x:x

donde cada x es el valor hexadecimal de las 8 partes de 16 bits de la dirección.

Las direcciones IPv6 abarcan desde 0000:0000:0000:0000:0000:0000:0000:0000 a ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff.



Direcciones IPv6

- Además de en este formato preferente, las direcciones IPv6 pueden especificarse en otros dos formatos abreviados:

Omitir ceros iniciales

- Por ejemplo, la dirección IPv6 1050:0000:0000:0005:0600:300c:326b puede escribirse como 1050:0:0:0:5:600:300c:326b.

Dos signos de dos puntos

- Dos signos de dos puntos (::) en lugar de una serie de ceros.
- Por ejemplo, la dirección IPv6 ff06:0:0:0:0:0:c3 puede escribirse como ff06::c3.
- Los dos signos de dos puntos sólo pueden utilizarse una vez en una dirección IP

IPv4	vs.	IPv6
Deployed 1981		Deployed 1998
32-bit IP address		128-bit IP address
4.3 billion addresses Addresses must be reused and masked		7.9x10²⁸ addresses Every device can have a unique address
Numeric dot-decimal notation 192.168.5.18		Alphanumeric hexadecimal notation 50b2:6400:0000:0000:6c3a:b17d:0:10a9 (Simplified - 50b2:6400::6c3a:b17d:0:10a9)
DHCP or manual configuration		Supports autoconfiguration

Una dirección IPv6 (en hexadecimal)

2001 :0DB8 :AC10 :FE01 :0000 :0000 :0000 :0000

↓ ↓ ↓ ↓ ──────────────────

2001 :0DB8 :AC10 :FE01 :: Se pueden omitir los ceros

10000000000001:0000110110111000:1010110000010000:11111100000000:
00000000000000:00000000000000:00000000000000:00000000000000

Direcciones IPv6

- Un formato alternativo para las direcciones IPv6 combina la notación de dos puntos y un punto, de forma que la dirección IPv4 puede incrustarse en la dirección IPv6.
- Se especifican valores hexadecimales para los 96 bits de más a la izquierda, y valores decimales para los 32 bits de más a la derecha que indican la dirección IPv4 incrustada.

Este formato garantiza la compatibilidad entre los nodos IPv6 y los nodos IPv4 cuando se está trabajando en un entorno de red mixto.

- La dirección IPv6 correlacionada con IPv4 utiliza este formato alternativo.

Este tipo de dirección se utiliza para representar los nodos IPv4 como direcciones IPv6. Permite que las aplicaciones de IPv6 se comuniquen directamente con las aplicaciones de IPv4. Por ejemplo, 0:0:0:0:ffff:192.1.56.10 y ::ffff:192.1.56.10/96 (formato abreviado).

Direcciones IPv6

- Las direcciones IPv6 se agrupan en estos tipos básicos de categorías:

Dirección de difusión simple

- La dirección de difusión simple especifica una interfaz única. Un paquete enviado a un destino de dirección de difusión simple viaja de un sistema principal al sistema principal de destino.

Dirección de difusión indiferente

- Una dirección de difusión indiferente especifica un conjunto de interfaces, posiblemente en ubicaciones diferentes, que comparten una única dirección. Un paquete enviado a una dirección de difusión indiferente se transmite sólo al miembro más próximo del grupo de difusión indiferente.

Dirección de difusión múltiple

- La dirección de difusión múltiple especifica un conjunto de interfaces, posiblemente en ubicaciones múltiples.
- El prefijo que se utiliza para una dirección de difusión múltiple es ff.
- Cuando se envía un paquete a una dirección de difusión múltiple, se envía una copia del paquete a cada miembro del grupo.

Direcciones IPv6

- Dirección de difusión simple

Los dos tipos habituales de direcciones de difusión simple son:

Dirección de enlace local

- Las direcciones de enlace local han sido diseñadas para su uso en un solo enlace local (red local). Las direcciones de enlace local se configuran automáticamente en todas las interfaces. El prefijo que se utiliza para una dirección de enlace local es fe80::/10. Los direccionadores no reenvían paquetes cuya dirección de destino o de origen contenga una dirección de enlace local.

Dirección global

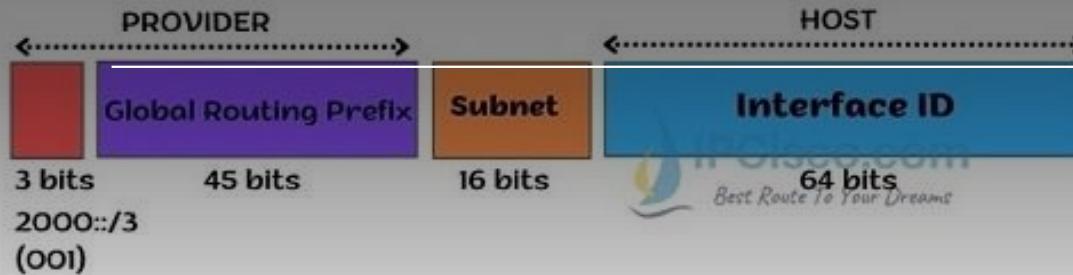
- Las direcciones globales están diseñadas para su uso en cualquier red. El prefijo que se utiliza para una dirección global empieza por un 001 binario

Dirección sin especificar 0:0:0:0:0:0:0:0 ::

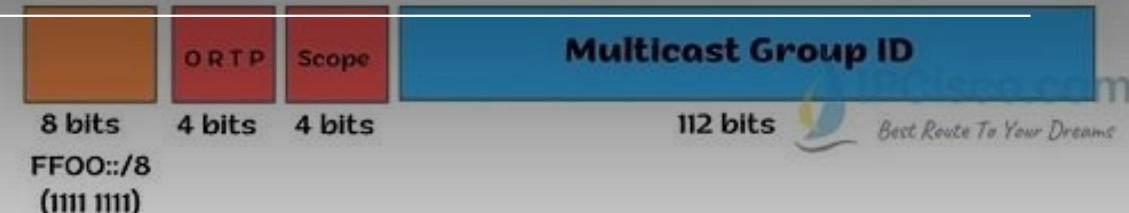
Dirección de bucle de retorno 0:0:0:0:0:0:0:1 ::1

IPv6 Address Types

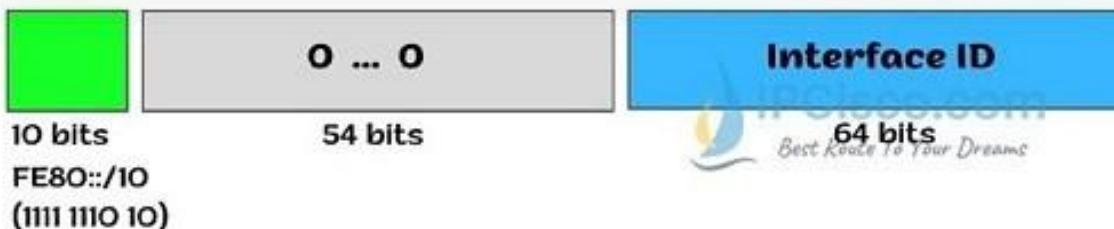
IPv6 Global Unicast Address



IPv6 Multicast Address



IPv6 Link-Local Address



IPv6 Solicited-Node Multicast Address



IPv6 Unique Local Address



IPv6 Anycast Address



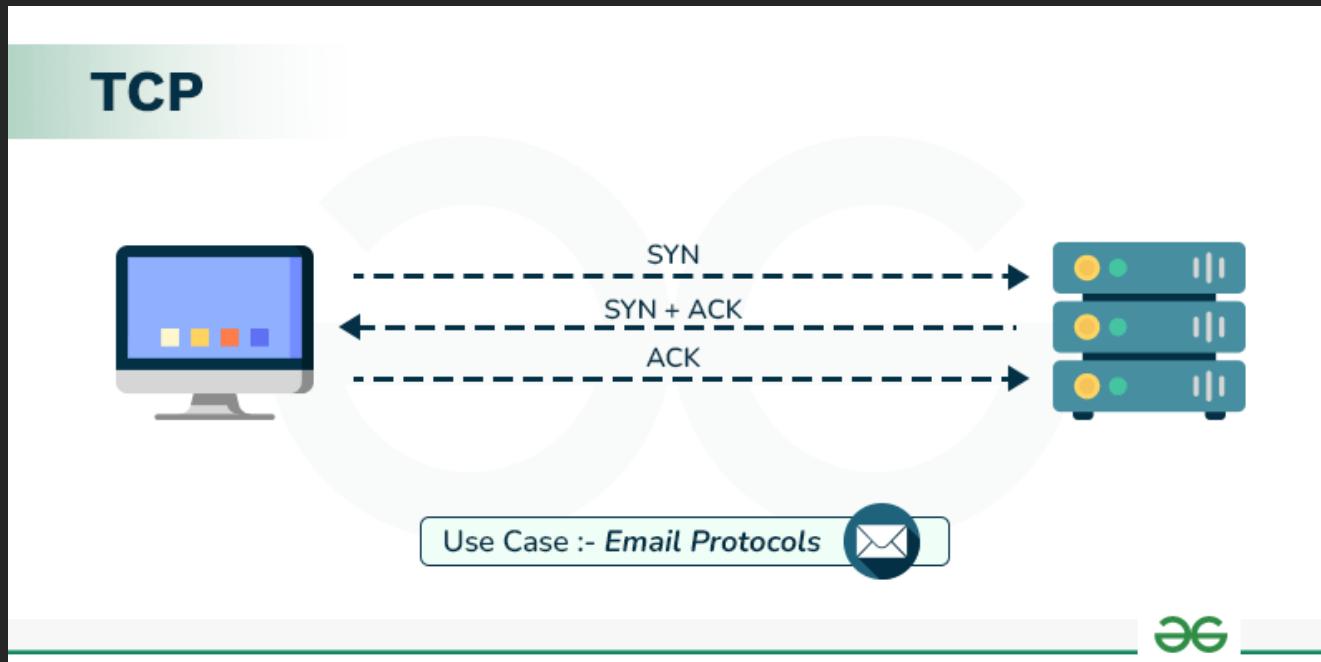
Capas de TCP/IP

- Capa 3 o de transporte
- Es donde tienen protagonismo los puertos lógicos de un router

Capa que establece los canales básicos que utilizarán las aplicaciones para intercambiar información entre dos puntos.

 - Define como será la conexión de un host a otro
 - Asociación a un puerto a un tipo de aplicación a su vez con un tipo de datos.
- En esta capa hay protocolos que se encargan de la segmentación de paquetes, del control de errores y el control de flujo de estos. El protocolo TCP opera en esta capa, siendo un de tipo orientado a conexión, mientras que el otro quizás más importante es el UDP que no es orientado a conexión. Veremos luego sus diferencias.
- Esta capa de transporte hace más o menos las mismas funciones que la capa de transporte del modelo OSI.

Capas de TCP/IP

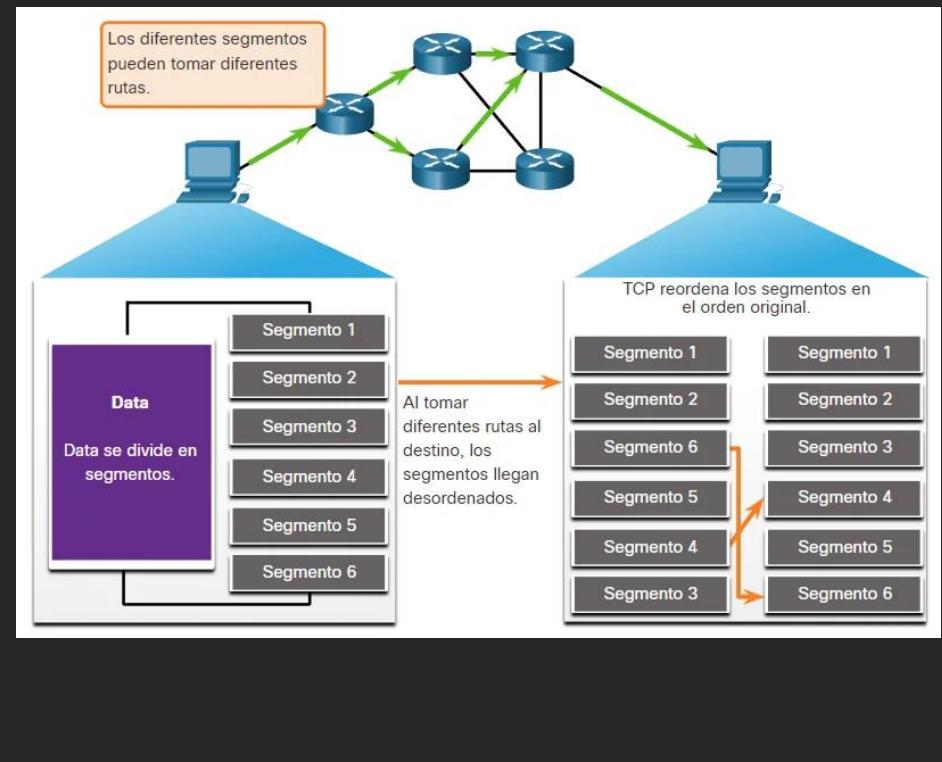


- Capa de transporte

TCP:

- Las aplicaciones pueden interactuar entre sí mediante TCP como si estuvieran conectadas físicamente por un circuito.
- TCP transmite datos de una manera que se asemeja a la transmisión carácter por carácter en lugar de paquetes separados.
- Un punto de inicio que establece la conexión, toda la transmisión en orden de bytes y un punto final que cierra la conexión conforman esta transmisión.

Capas de TCP/IP



- El modelo de Protocolo de Control de Transmisión (TCP):
Divide los datos en pequeños segmentos en el emisor
Los vuelve a ensamblar en el mensaje original en el extremo opuesto para asegurarse de que cada mensaje llegue intacto a su destino.
Enviar la información en pequeños segmentos de información hace que sea más sencillo mantener la eficiencia en comparación con enviar todo de una sola vez.
Después de dividir un mensaje en particular en segmentos, estos segmentos pueden viajar por múltiples rutas si una ruta está bloqueada, pero el destino sigue siendo el mismo.

Capas de TCP/IP

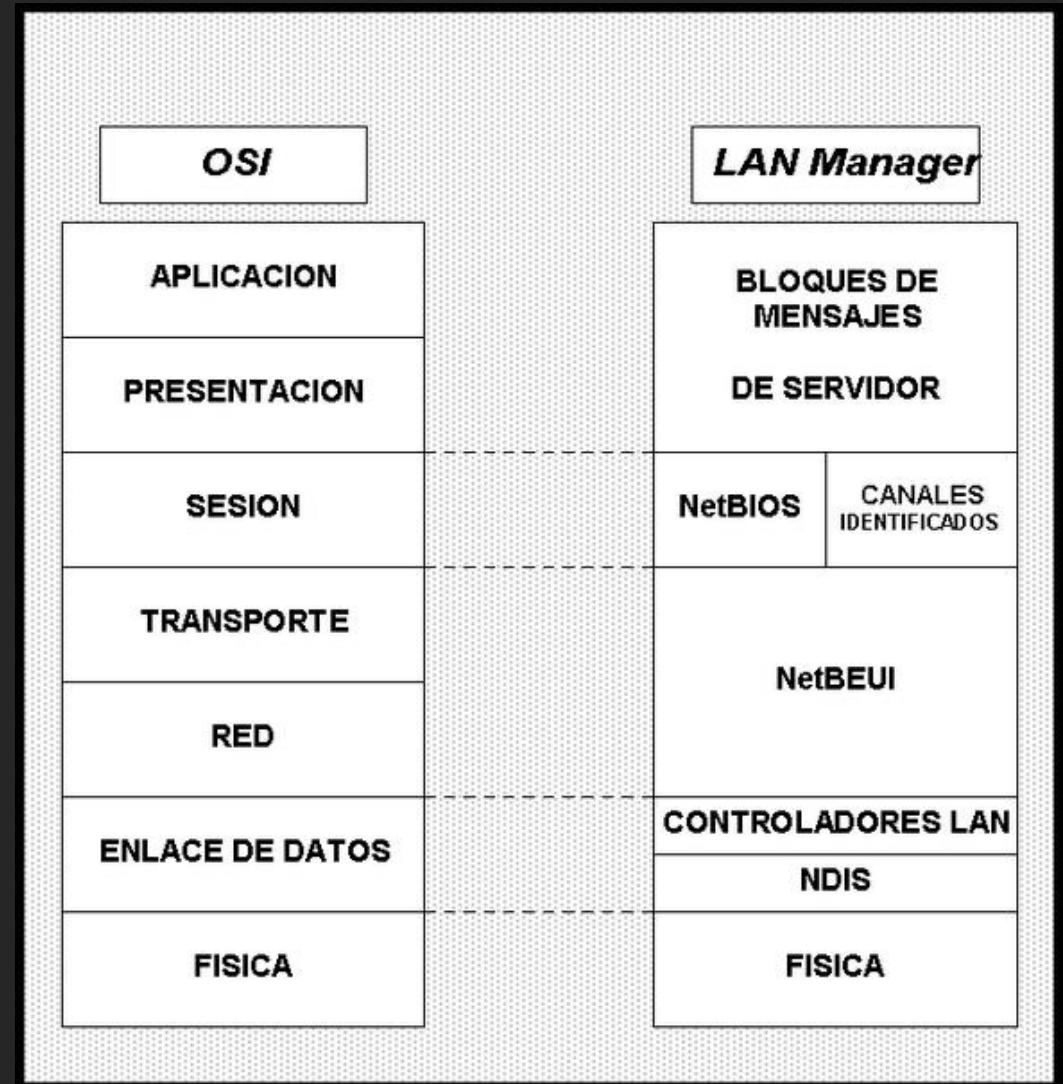
- Ejemplo de funcionamiento de TCP

Cuando un usuario solicita una página web en Internet, en algún lugar del mundo, el servidor procesa esa solicitud y envía una página HTML a ese usuario.

El servidor utiliza un protocolo llamado Protocolo HTTP.

- A continuación, el protocolo HTTP solicita a la capa TCP que establezca la conexión necesaria y envíe el archivo HTML.
- Ahora, el protocolo TCP divide los datos en pequeños segmentos y los reenvía hacia la capa de Protocolo de Internet (IP).
- A continuación, los paquetes se envían al destino a través de diferentes rutas.
- La capa TCP del sistema del usuario espera a que finalice la transmisión y confirma la recepción de todos los paquetes.

Otros protocolos habituales



Otros protocolos habituales

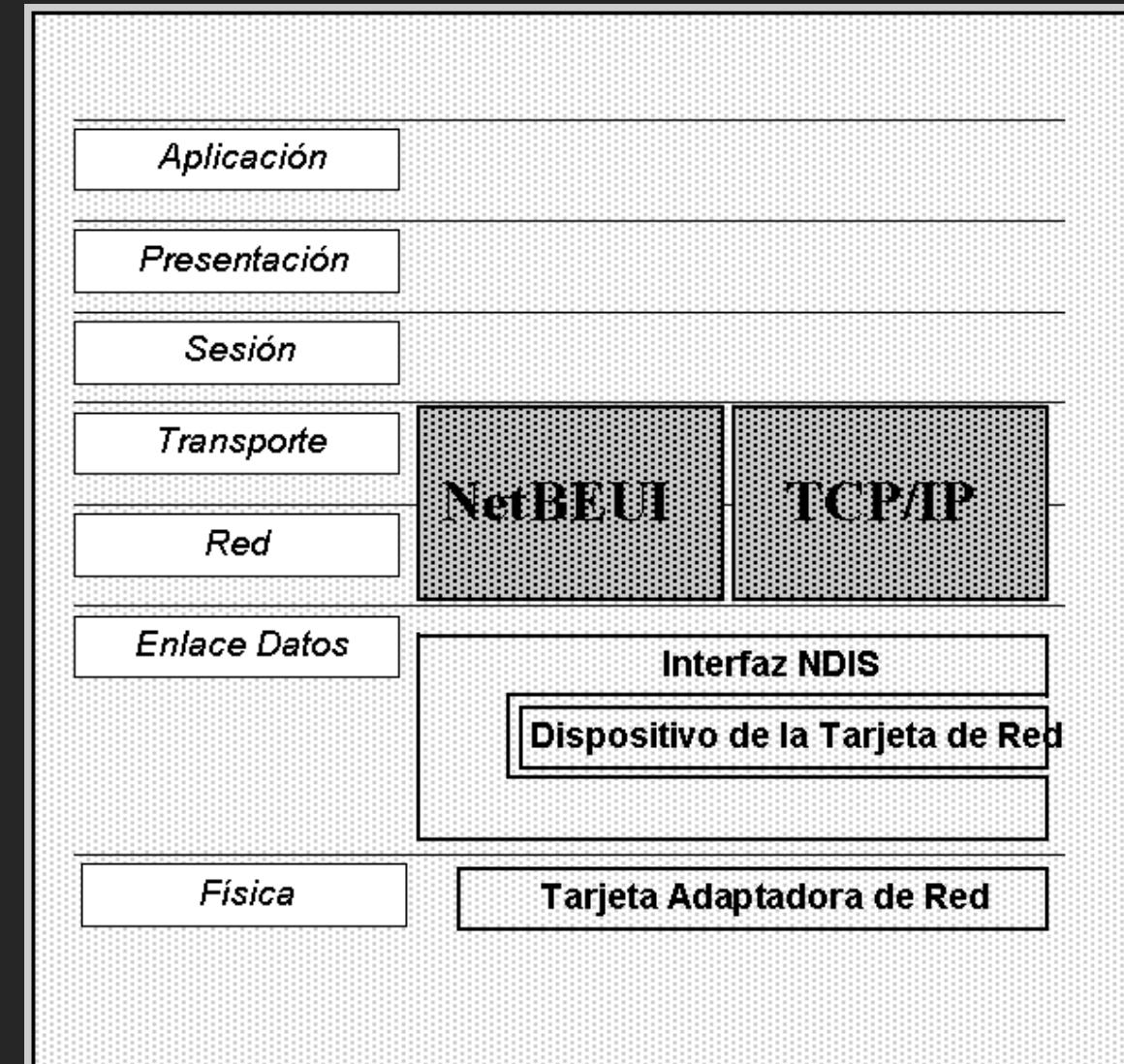
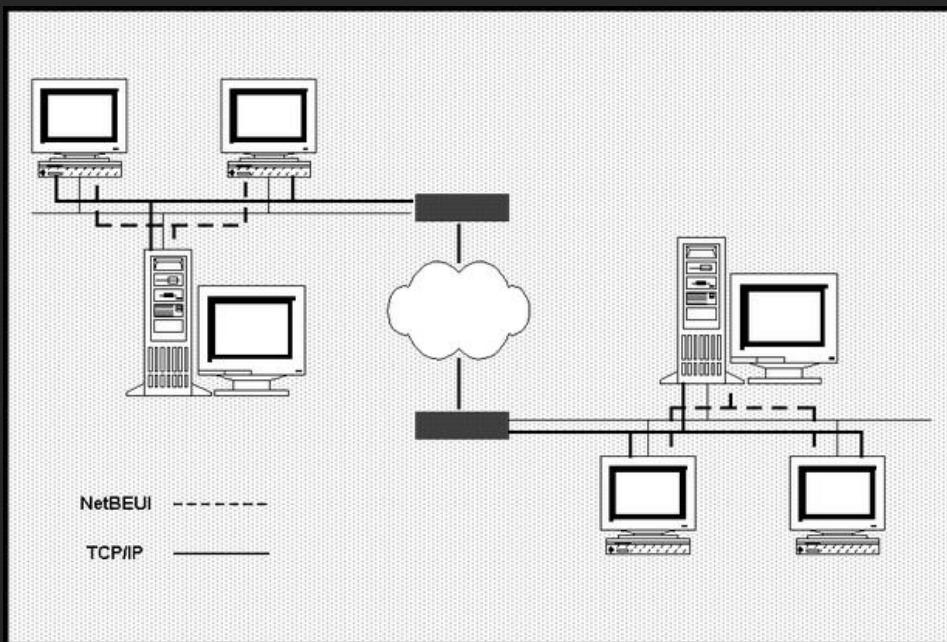
- NetBEUI, Interfaz de Usuarios Extendida NetBIOS, es como su nombre lo indica una versión extendida de NetBIOS.

Fue introducido por IBM en 1.985.

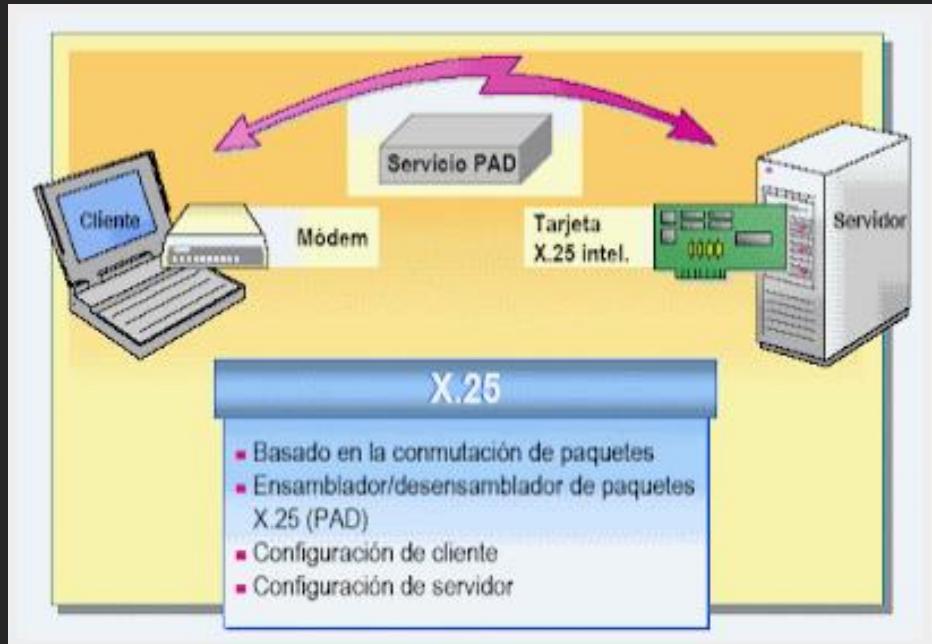
Es un protocolo eficiente y rápido que permite que las computadoras se comuniquen en un ambiente LAN

- de 20 a 200 estaciones de trabajo o en un segmento de LAN.
- Es más conocido en el ambiente Microsoft por NBF.
- NetBEUI provee los servicios de transporte de datos descriptos en las capas 3 y 4 del modelo OSI.
- NetBEUI emplea la interfaz NetBIOS como una interfaz de nivel superior y a su vez proporciona al mismo el formato necesario para la transmisión de los datos.
- El inconveniente que posee este protocolo es que no es *ruteable*.

Otros protocolos habituales

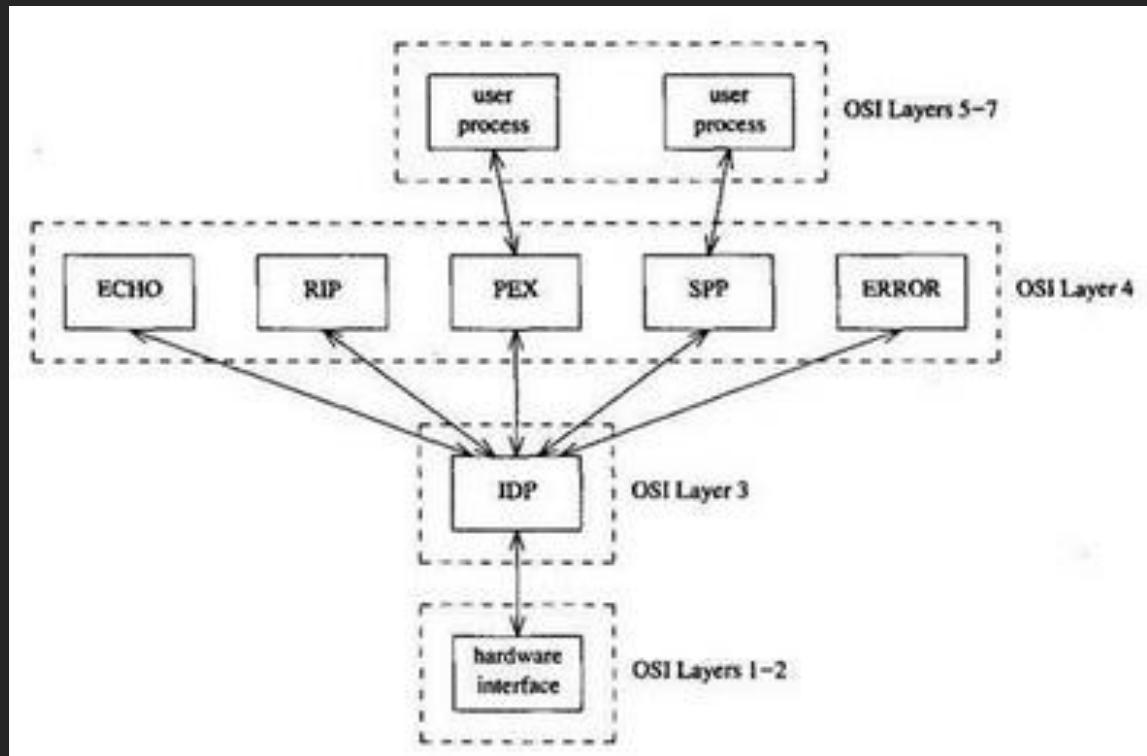


Otros protocolos habituales



- X.25 es un estándar ITU-T para redes de área amplia de conmutación de paquetes
- Su protocolo de enlace, LAPB (Link Access Procedure, Balanced), está basado en HDLC (High-Level Data Link Control) que a su vez se basa en SDLC (Synchronous Data Link Control)
- Establece mecanismos de:
 - Direccionamiento entre usuarios*
 - Negociación de características de comunicación*
 - Técnicas de recuperación de errores*

Otros protocolos habituales



- XNS

Xerox Network Systems (también llamado Xerox NS o XNS) es la arquitectura de red desarrollada por Xerox Corporation a fines de la década de 1970

- Integra sus productos de oficina y sistemas informáticos.
- XNS es un sistema abierto
- Otros proveedores también proporcionan hardware y software que admiten los protocolos XNS.
- La mayoría de los sistemas 4.3BSD admiten el conjunto de protocolos XNS.
- XNS es similar en estructura al conjunto de protocolos TCP/IP

Otros protocolos habituales

- XNS

ECHO Protocolo de eco. Un protocolo simple que hace que un host repita el paquete que recibe. La mayoría de las implementaciones de XNS admiten este protocolo.

RIP Protocolo de información de enrutamiento. Un protocolo utilizado para mantener una base de datos de enrutamiento para su uso en un host en el reenvío de paquetes IDP a otro host. Normalmente existe un proceso de enrutamiento en el host y este proceso utiliza RIP para mantener la base de datos.

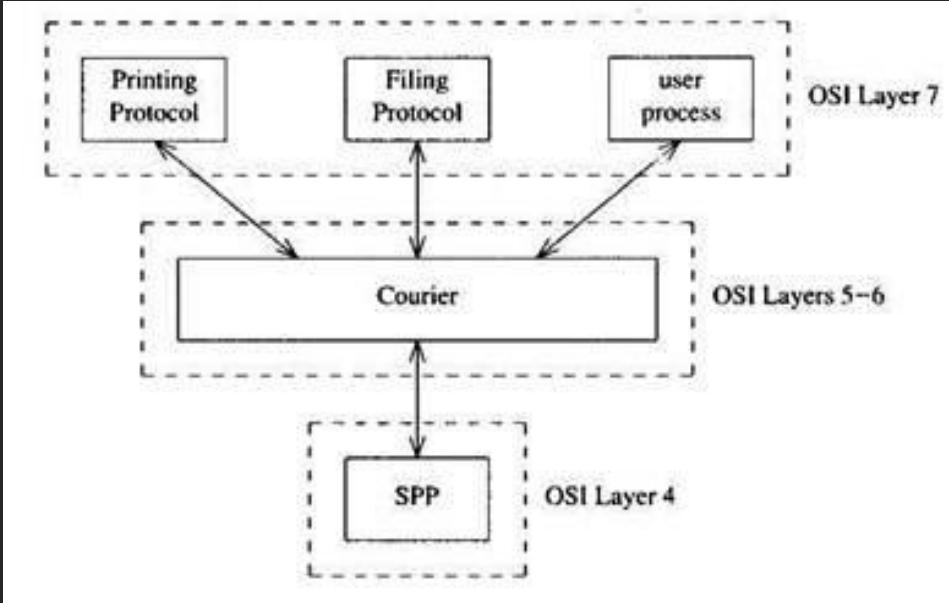
PEX Protocolo de intercambio de paquetes. Un protocolo de datagramas no confiable y sin conexión para procesos de usuario. Aunque PEX no es un protocolo confiable, realiza retransmisiones pero no realiza detección de duplicados.

SPP Protocolo de paquetes secuenciados. Un protocolo confiable y orientado a la conexión para procesos de usuario. Proporciona un flujo de bytes para el proceso de usuario con límites de mensajes opcionales. SPP es el protocolo más comúnmente utilizado en el conjunto XNS, similar a TCP en el conjunto de Internet.

ERROR Protocolo de error. Protocolo que puede utilizar cualquier proceso para informar que ha descubierto un error y, por lo tanto, ha descartado un paquete.

IDP Protocolo de datagramas de Internet. IDP es el protocolo de datagramas no confiable y sin conexión que proporciona el servicio de entrega de paquetes para todos los protocolos anteriores.

Otros protocolos habituales



- Xerox denomina a los cinco protocolos que se muestran en la capa 4 de OSI Protocolos de transporte de "Internet".
- La mayoría de las aplicaciones de Xerox se crean utilizando el protocolo de llamada a procedimiento remoto Courier.

Courier, a su vez, se crea utilizando SPP.



Otros protocolos habituales

- APPC (Advanced Program-to Program Communications)
APPC, también conocido como LU 6.2, fue introducido por IBM en 1982
 - Intercambio de datos entre dos programas pares que se encuentran en la misma computadora o en dos sistemas conectados por la red.
Arquitectura que define un conjunto de protocolos de red y una interfaz de programación de aplicaciones (API).
 - Debido a que APPC es un protocolo de red, no aborda la sintaxis de lenguaje de programación individual.
 - La API de APPC se describe como una presentación abstracta de las diversas funciones de API llamadas verbos.
 - Los verbos definen la secuencia y el orden que debe seguir un programa de aplicación para comunicarse con programas de aplicación pares.
 - La implementación de APPC en los diversos lenguajes de programación convierte la API abstracta en funciones o subrutinas invocables que se ajustan a la sintaxis del lenguaje de programación.

Programación con Sockets

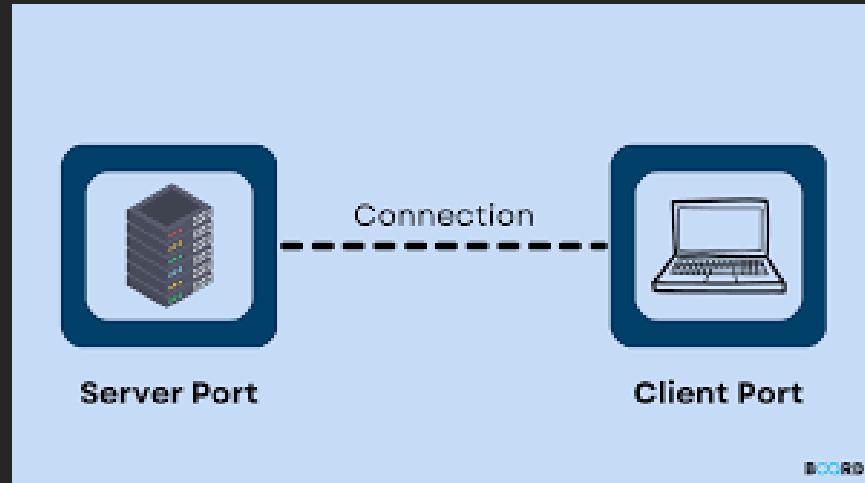
```
    _for object to mirror
    mirror_mod.mirror_object

    operation = "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
    mirror_mod.operation = "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
    mirror_mod.operation = "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

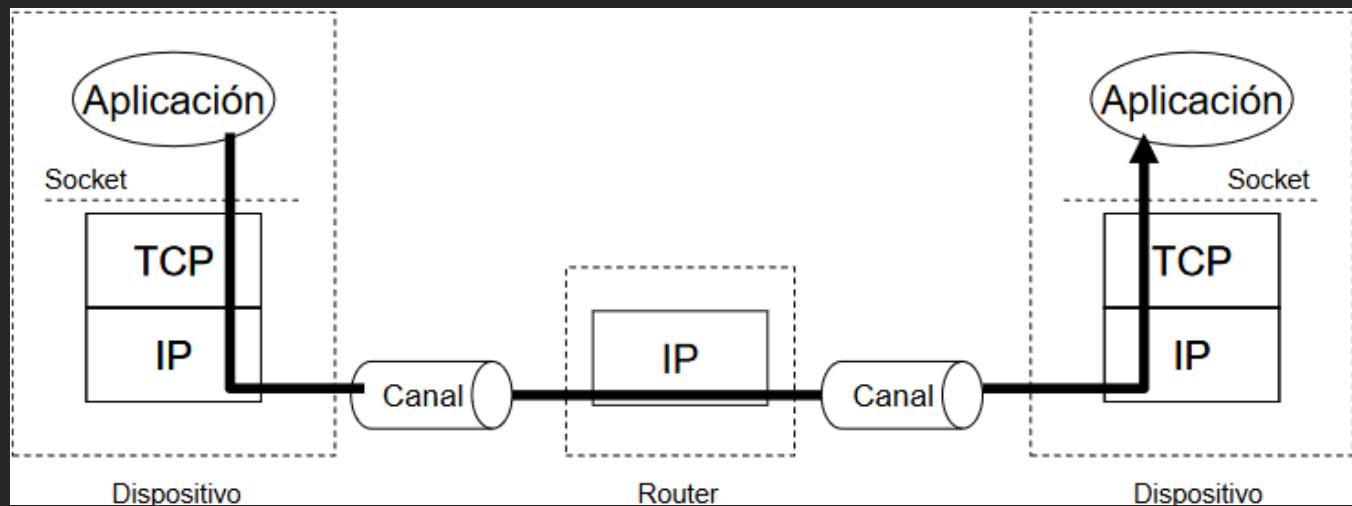
    selection at the end -add
    mirror_ob.select= 1
    mirror_ob.select=1
    context.scene.objects.active = one
    ("Selected" + str(modifier))
    mirror_ob.select = 0
    bpy.context.selected_objects = []
    data.objects[one.name].select = 1
    print("please select exactly one object")
    - OPERATOR CLASSES -
types.Operator:
    X mirror to the selected
    object.mirror_mirror_x"
    for X"
```

Definición

- Los sockets son una de las herramientas que ofrecen los Sistemas Operativos para la comunicación entre diferentes procesos.
- La particularidad que tienen frente a otros mecanismos de comunicación entre procesos (IPC – Inter-Process Communication) es que:
Posibilitan la comunicación aun cuando ambos procesos estén corriendo en distintos sistemas unidos mediante una red.
- De hecho, el API de sockets es la base de cualquier aplicación que funcione en red
Ofrece una librería de funciones básicas que el programador puede usar para desarrollar aplicaciones en red



Sockets TCP/IP



- Permiten la comunicación de dos procesos que estén conectados a través de una red TCP/IP.
*Cada máquina está identificada por medio de su dirección IP
Cada máquina puede estar ejecutando múltiples procesos simultáneamente.*
 - Cada uno de estos procesos se asocia con un número de puerto
Un socket se identifica únicamente por la dupla dirección IP + número de puerto.
- Una comunicación entre dos procesos se identifica mediante:
La asociación de los sockets que estos emplean para enviar y recibir información hacia y desde la red
identificador de socket origen + identificador de socket destino.

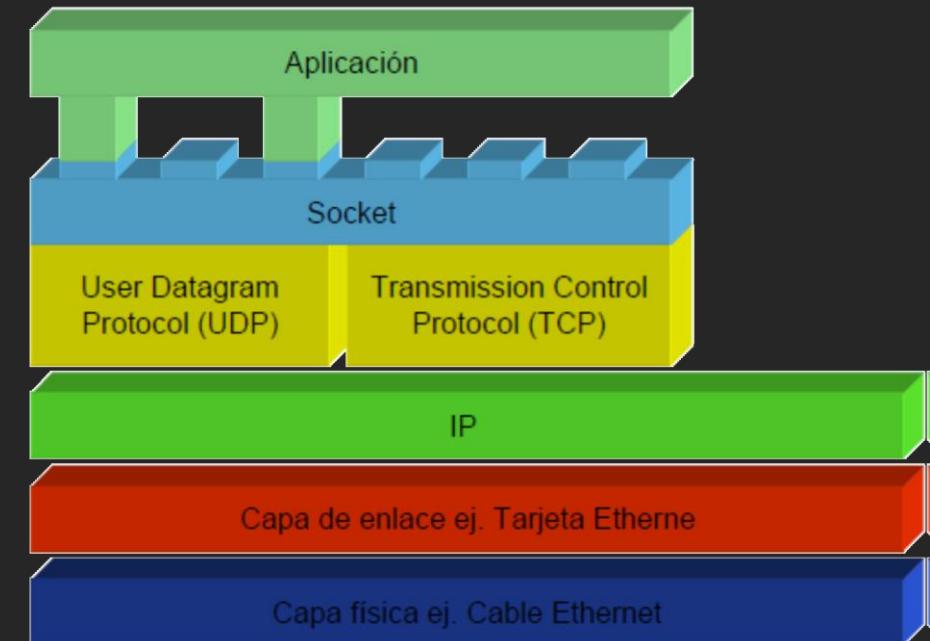
Protocolos y sockets

- Un socket es una abstracción a través de la cual una aplicación puede enviar y recibir información

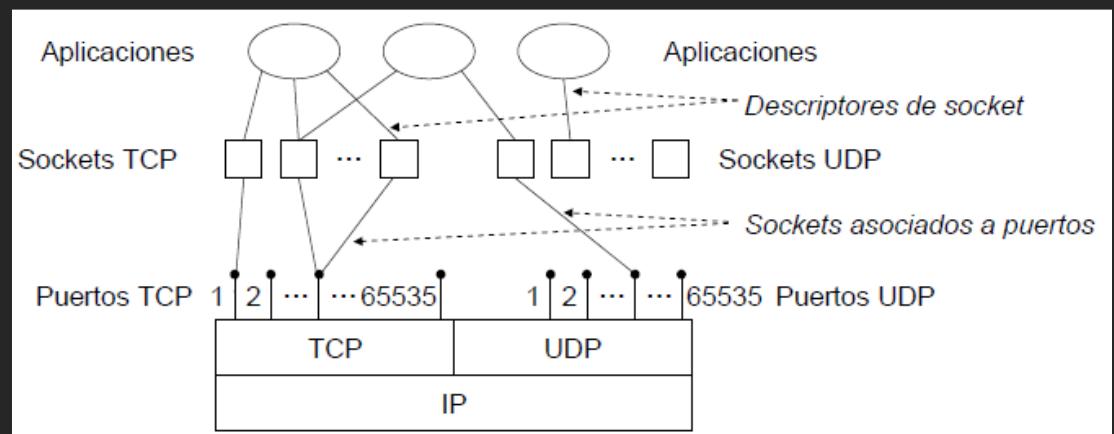
Muy similar a como se escribe y lee de un archivo.

La información que una aplicación envía por su socket origen puede ser recibida por otra aplicación en el socket destino y viceversa.

- Existen diferentes tipos de sockets dependiendo de la pila de protocolos sobre la que se cree dicho socket.



Sockets, Protocolos y Puertos



- Relación lógica entre aplicaciones, sockets, protocolos y puertos en un dispositivo.

Aspectos por destacar en esta relación

- Primero, un programa puede usar más de un socket al mismo tiempo
- Segundo, diferentes programas pueden usar el mismo socket al mismo tiempo,

Cada socket tiene asociado un puerto TCP o UDP, según sea el caso.

Cuando se recibe un paquete dirigido a dicho puerto, este paquete se pasa a la aplicación correspondiente.

Dominios de comunicación

- Los sockets se crean dentro de lo que se denomina un dominio de comunicación que define cual será la pila de protocolos que se usará en la comunicación

Dominio	Propósito
PF_UNIX, PF_LOCAL	Procesos que se comunican en un mismo sistema UNIX
PF_INET	Procesos que se comunican usando una red IPv4
PF_INET6	Procesos que se comunican usando una red IPv6

PF_IPX	Procesos que se comunican usando una red Novell
PF_NETLINK	Comunicación con procesos del kernel
PF_X25	ITU-T X.25 / ISO-8208 protocol
PF_AX25	Amateur radio AX.25
PF_ATMPVC	Acceso a PVCs ATM
PF_APPLETALK	Appletalk
PF_PACKET	Interfaz con paquetes de bajo nivel

Tipos de Sockets

- En el dominio PF_INET se definen los siguientes tipos de sockets:

Sockets Stream

- hace uso del protocolo TCP que provee un flujo de datos bidireccional, orientado a conexión, secuenciado, sin duplicación de paquetes y libre de errores

Sockets Datagram

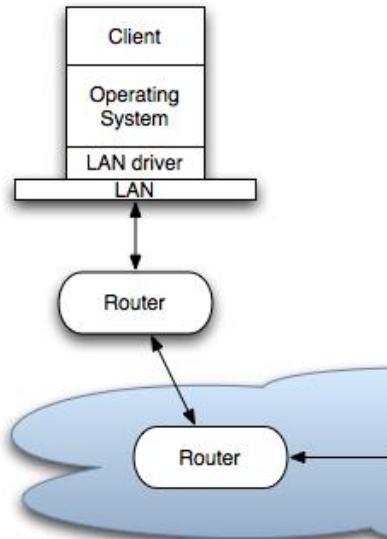
- hacen uso del protocolo UDP, el cual provee un flujo de datos bidireccional, no orientado a conexión, en el cual los paquetes pueden llegar fuera de secuencia, puede haber pérdidas de paquetes o pueden llegar con errores

Sockets Raw

- Permiten un acceso a más bajo nivel, pudiendo acceder directamente al protocolo IP del nivel de Red. Su uso está mucho más limitado ya que está pensado principalmente para desarrollar nuevos protocolos de comunicación, o para obviar los protocolos del nivel de transporte.

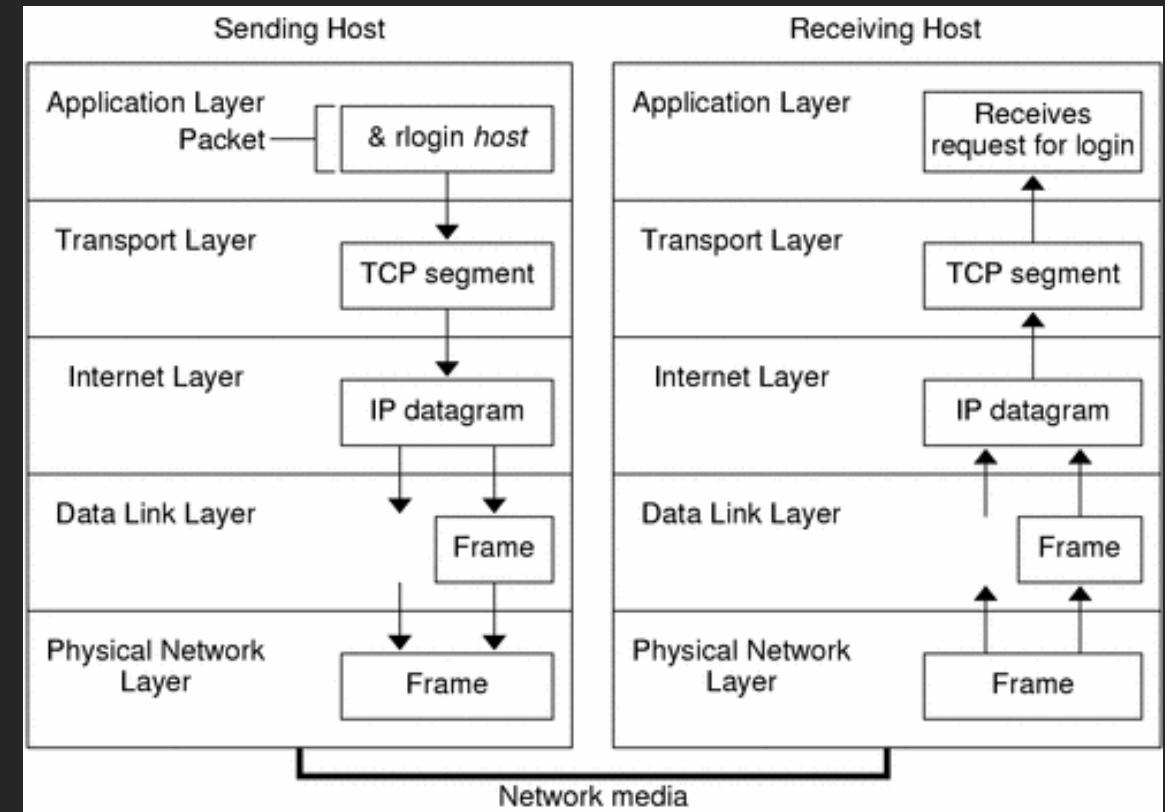
Modelo Cliente Servidor

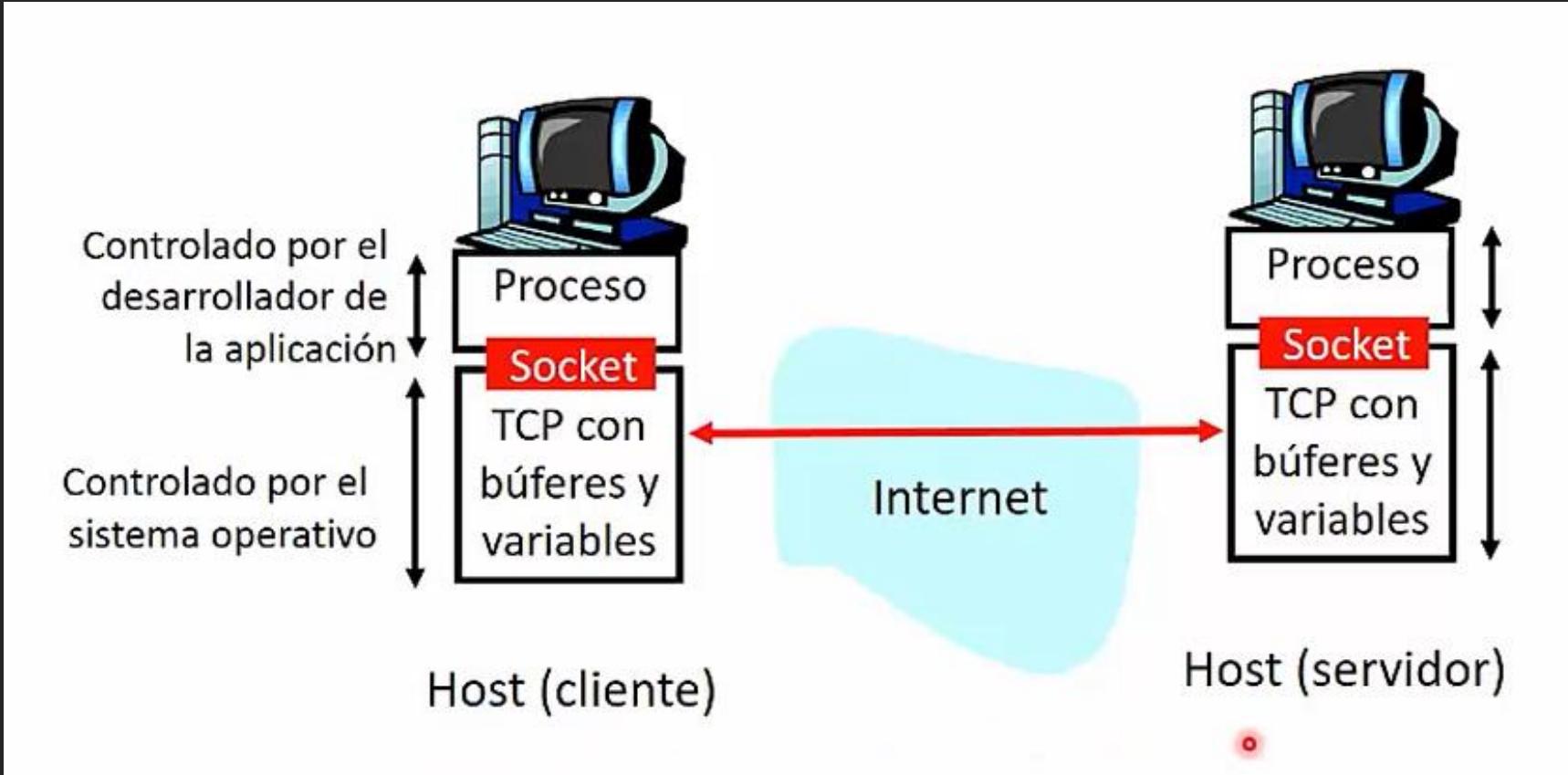
IPcte



IPser

Cliente inicia comunicación





Programación de Sockets TCP

Servicios de sockets

C/C++ - Linux sockets

```
socket uns = socket(AF_INET , SOCK_STREAM , 0);
```

C/C++ - Winsockets

```
SOCKET uns = socket(family, socktype, protocol);
```

Plataforma Java

```
import java.net.*;
var uns = new Socket(Dirección, puerto);
```

Plataforma .NET

```
using System.Net.Sockets;
Socket uns = new Socket(Family, SocketType, ProtocolType);
```

Sockets en lenguajes

Cliente

1. Crear un socket local TCP del cliente
2. Indicar la dirección IP y puerto del servidor
3. Activar la conexión con el servidor
4. Enviar y recibir información
5. Cerrar la conexión

Servidor

1. Crear un socket local TCP del servidor
2. Establecer un puerto de escucha
3. Quedar a la espera de peticiones
4. Atender peticiones entrantes
5. Volver al paso 3

Programación de Sockets TCP

Servicio orientado a la conexión (TCP)	
Cliente	Servidor
	socket()
	bind()
	listen()
	accept()
socket()	
[bind()]	
connect()	
send()	recv()
recv()	send()

Secuencia de ejecución

Manejo de socket

- Creación de Socket

Los sockets se crean llamando a la función socket(), que devuelve el identificador de socket, de tipo entero

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int socket(int domain, int type, int protocol);

#include <stdio.h> /* para perror() */
#include <sys/types.h>
#include <sys/socket.h>

...
int sockfd;
sockfd = socket ( PF_INET, SOCK_STREAM, 0 );
if(sockfd < 0)
perror("Error creating the socket");
```

Manejo de sockets

- Vinculación del socket a una dirección IP y un número de puerto específico

La función bind() se utiliza para asociar el socket a una dirección IP y número de puerto de la máquina local a través del que se enviarán y recibirán datos.

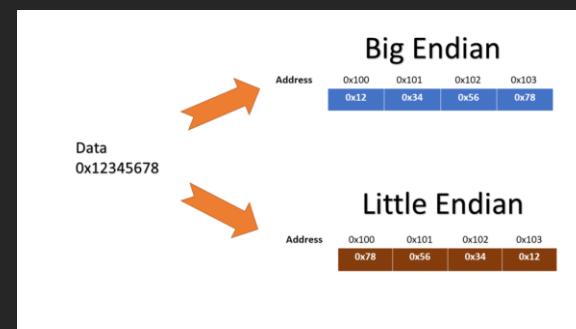
El formato de la función es el siguiente:

```
#include <sys/types.h>
#include <sys/socket.h>

...
int bind(int local_s, const struct sockaddr *addr, int addrlen);
```

```
struct sockaddr_in
{
    short int sin_family; // PF_INET
    unsigned short sin_port; // Numero de puerto.
    struct in_addr sin_addr; // Dirección IP.
    unsigned char sin_zero[8]; // Relleno.
};
```

```
...
struct sockaddr_in sin;
...
sin.sin_family = PF_INET;
sin.sin_port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
sin.sin_addr.s_addr = inet_addr ("132.241.5.10");
// IP por la que recibirá paquetes el programa
```



Manejo de sockets

- Vinculación

```
...
struct sockaddr_in sin;
...
sin.sin_family = PF_INET;
sin.sin_port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
sin.sin_addr.s_addr = inet_addr ("132.241.5.10");
// IP por la que recibirá paquetes el programa

...
struct sockaddr_in sin;
...
ret = bind (sockfd, (struct sockaddr *)&sin, sizeof (sin));
if(ret < 0)
    perror("Error binding the socket");
...
```

La llamada a la función `bind()` en el cliente es opcional, ya que en caso de no ser invocada el sistema la ejecutará automáticamente asignándole un puerto libre (al azar). En cambio, en el servidor es obligatorio ejecutar la llamada a la función `bind()` para reservar un puerto concreto y conocido.

Manejo de sockets

- Escuchar conexiones

El servidor escuche las conexiones entrantes mediante la función listen():

El servidor habilita su socket para poder recibir conexiones, llamando a la función listen().

- En el cliente este paso no es necesario, ya que no recibirá peticiones de conexión de otros procesos.

```
#include <sys/socket.h>
...
int listen(int sockfd, int backlog);
listen ( sockfd, 5);
```

Manejo de Sockets

- Aceptar conexiones

Cuando un cliente intenta conectarse, el servidor utiliza la función accept() para aceptar la conexión entrante

- Función listen() prepara el socket y lo habilita para recibir peticiones de establecimiento de conexión
- Función accept() la que realmente queda a la espera de estas peticiones.
- Cuando una petición realizada desde un proceso remoto (cliente) es recibida, la conexión se completa en el servidor siempre y cuando éste esté esperando en la función accept().
- La función accept() es utilizada en el servidor una vez que se ha invocado a la función listen().
- Esta función espera hasta que algún cliente establezca una conexión con el servidor.

Es una llamada bloqueante, esto es, la función no finalizará hasta que se haya producido una conexión o sea interrumpida por una señal.

Manejo de Sockets

- Es conveniente destacar que una vez que se ha producido la conexión, la función accept() devuelve un nuevo identificador de socket que será utilizado para la comunicación con el cliente que se ha conectado.

```
#include <sys/types.h>
#include <sys/socket.h>
...
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

```
new_sockfd = accept (sockfd, &remote_addr, &addrlen);
```

Manejo de sockets

```
...
int sockfd, new_sockfd;
struct sockaddr_in server_addr;
struct sockaddr_in remote_addr;
int addrlen;
// Creación del socket.
sockfd = socket (PF_INET, SOCK_STREAM, 0 );
// Definir valores en la estructura server_addr.
server_addr.sin_family = PF_INET;
server_addr.sin_port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
// Asociar valores definidos al socket
bind(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));
// Se habilita el socket para poder recibir conexiones.
listen ( sockfd, 5);
addrlen = sizeof (struct sockaddr );
// Se llama a accept() y el servidor queda en espera de conexiones.
new_sockfd = accept (sockfd, &remote_addr, &addrlen);
...
```

Lanzar peticiones de conexión

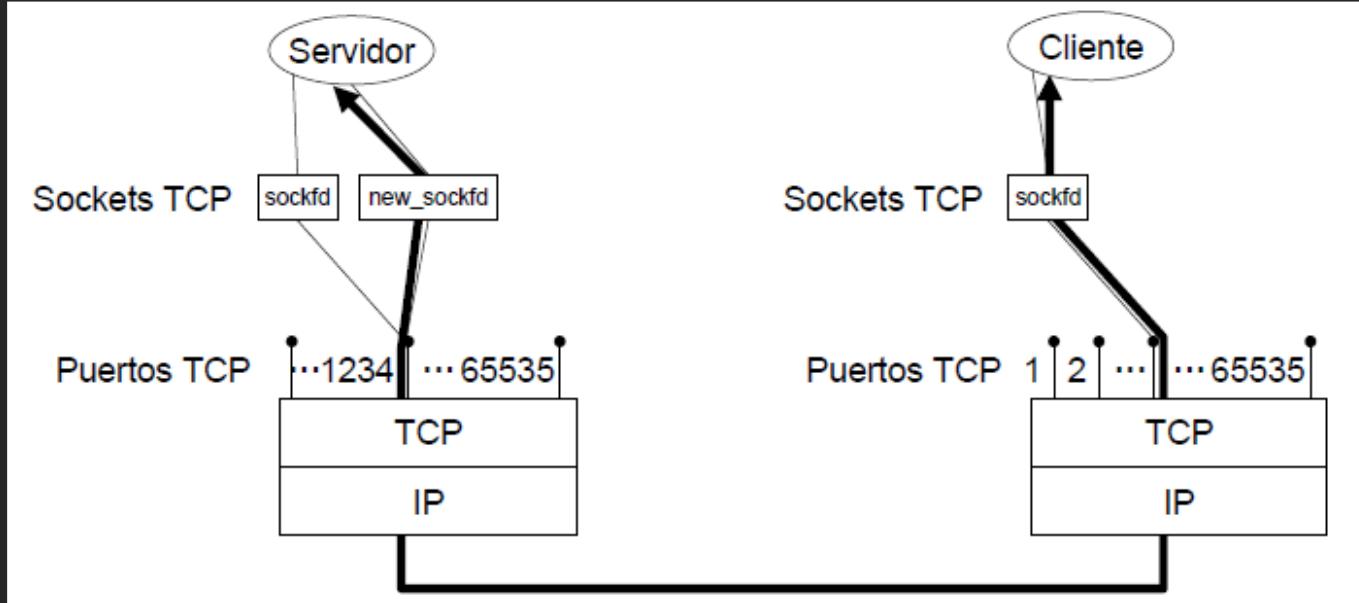
- Esta función es invocada desde el cliente para solicitar el establecimiento de una conexión TCP.

La función connect() inicia la conexión con el servidor remoto, por parte del cliente. El formato de la función es el siguiente:

```
#include <sys/types.h>
#include <sys/socket.h>
...
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

Lanzar peticiones de conexión

```
...
int sockfd;
struct sockaddr_in server_addr;
int addrlen;
// Creación del socket.
sockfd = socket (PF_INET, SOCK_STREAM, 0 );
// Definir valores en la estructura server_addr.
server_addr.sin_family = PF_INET;
server_addr.sin_port = htons ( 1234 );
// Número de puerto donde
// está esperando el servidor
server_addr.sin_addr.s_addr = inet_addr("1.2.3.4");
// Dirección IP
// del servidor
addrlen = sizeof (struct sockaddr );
// Se llama a connect () y se hace la petición de conexión al servidor
connect (sockfd, &server_addr, addrlen);
...
```



Conexión entre los sockets del cliente y servidor

- El cliente ha hecho la llamada a la función `connect()`
- Al concluir con éxito (lo cual significa que en el servidor se estaba esperando en la función `accept()` y se ha salido de ella y por tanto se ha creado el nuevo socket)

Enviar y recibir datos a través del socket

- Una vez que la conexión ha sido establecida, se inicia el intercambio de datos, utilizando para ello las funciones send() y recv().

La función send() devuelve el número de bytes enviados, que puede ser menor que la cantidad indicada en el parámetro len.

```
#include <sys/types.h>
#include <sys/socket.h>
...
ssize_t send(int s, const void *buf, size_t len, int flags);

ssize_t send (int sockfd, const void *buf, size_t len, int flags );
```

La función recv() se utiliza para recibir datos

```
#include <sys/types.h>
#include <sys/socket.h>
...
ssize_t recv(int s, void *buf, size_t len, int flags);

ssize_t recv (int sockfd, void *buf, size_t len, int flags);
```

Enviar y recibir datos a través del socket

- La función recv() es bloqueante

No finaliza hasta que se ha recibido algún tipo de información.

Resaltar que el campo len indica el número máximo de bytes a recibir

- No necesariamente se han de recibir exactamente ese número de bytes.

La función recv() devuelve el número de bytes que se han recibido.

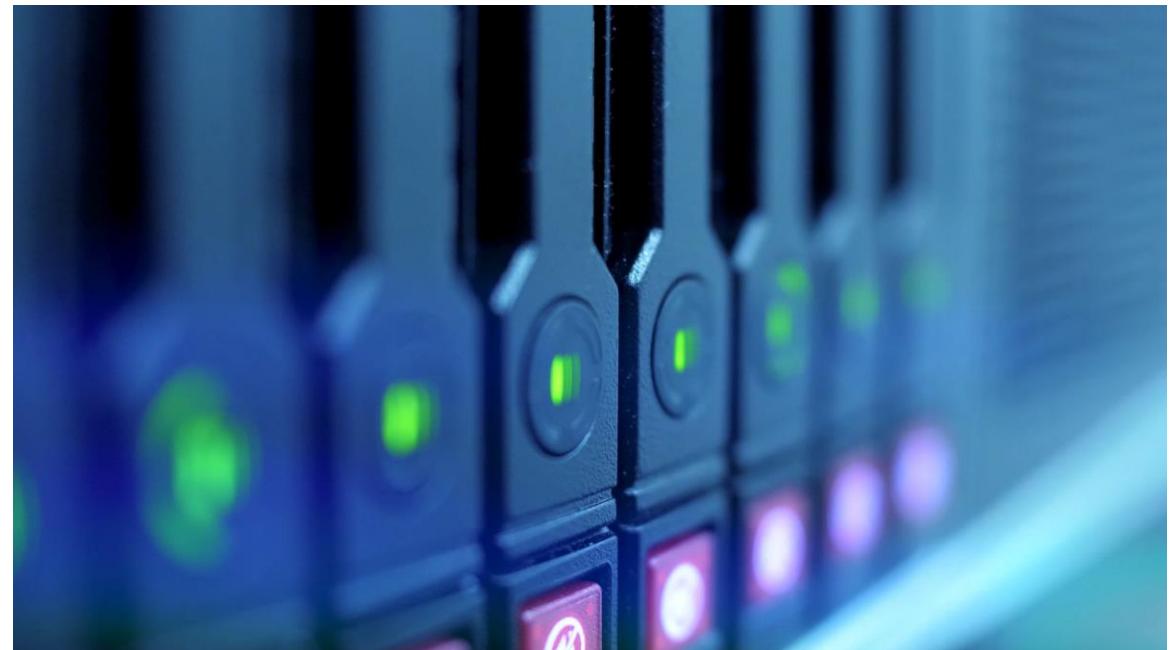
```
...
char mens_serv[100];
...
mens_clien = "Ejemplo";
send(sid, mens_clien, strlen(mens_clien)+1, 0);
...
recv(sid, mens_serv, 100, 0);
...
```

Cierre de un socket

- Simplemente hay que hacer la llamada a la función pasándole como argumento el descriptor del socket que se quiere cerrar.
- Es importante que cuando un socket no vaya a usarse más, se haga la llamada a close() para indicárselo al Sistema Operativo y que éste lo libere.

```
#include <unistd.h>
...
int close(int fd);
```

Definición y Características de los Sistemas de Transmisión



Unidad III

Uaaccbasdseici7

eiraagodic2024



Control de enlace de datos

Control de flujo

El control de flujo es un aspecto fundamental que dicta la eficacia de cualquier proceso, sistema u operación.

Es la mano invisible que guía el buen funcionamiento de los sistemas

- Garantiza que todas las piezas funcionen juntas en equilibrio
- Resulta crucial para que los sistemas informáticos sean más organizados y manejables

En el contexto de la TI

- Conjunto de procedimientos utilizados para gestionar la tasa de transferencia a la que se transmiten los datos entre dos nodos.
- Garantiza que un emisor, si opera a un ritmo más rápido, no inunde de datos a un receptor más lento.
- El mecanismo empleado permite al nodo receptor controlar la velocidad de transmisión.

Control de flujo

- Finalidad
 - Fundamental para mantener el equilibrio en la velocidad de transmisión de datos entre un emisor y un receptor
 - Al activar el control de flujo, se puede mejorar significativamente:
 - El rendimiento de la red
 - Reducir las retransmisiones
 - Aumentar la eficacia al evitar la pérdida de datos o la congestión.
- Ejemplos
 - Servidor rápido cliente lento
 - Tráfico elevado
- Estos métodos de control de flujo se emplean también para corregir errores



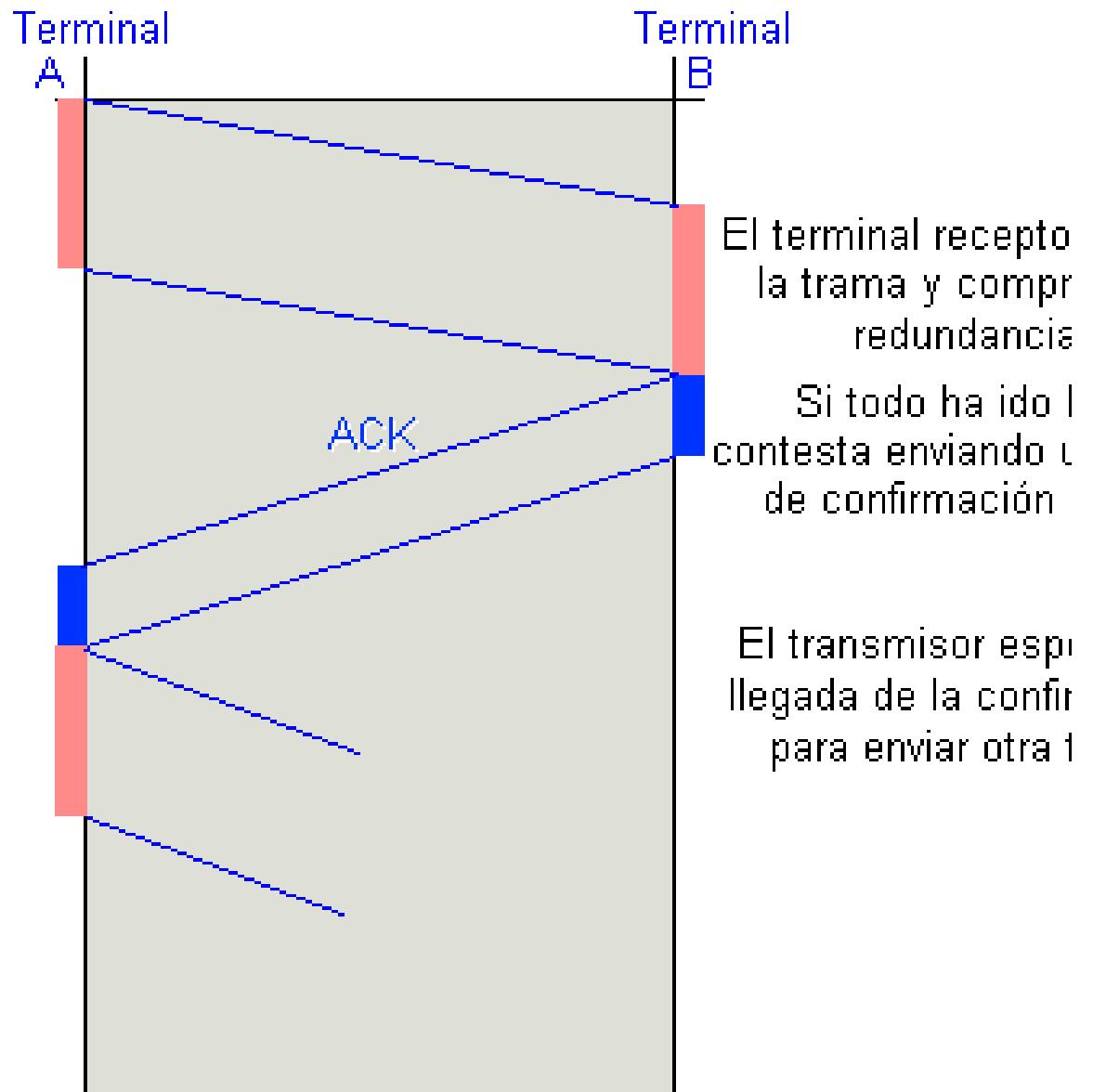
Control de flujo

- ARQ, Requerimiento automático de repetición
 - Se trata de sistemas de corrección hacia atrás
 - En estos sistemas la estación receptora que ha detectado la recepción de caracteres o bloques con errores procede a pedir a la estación emisora que repita lo recibido con error.
 - Debe observarse que esto requiere dar al sistema de comunicación algún medio para facilitar el diálogo entre la estación emisora y la estación receptora
 - El extremo receptor abandona el papel pasivo en la comunicación para participar en forma activa en el proceso.
- Existen dos estrategias principales en el diseño de los sistemas de corrección hacia atrás:
 - Pare y espere (stop and wait ARQ).
 - Ventana deslizante (Continuos ARQ).
 - Ambos trabajan en conjunto con métodos de detección de errores

Control de flujo

- Parada y espera
 - Las tramas se van intercambiando una a una.
 - Cuando el receptor recibe una trama procede a validarla
 - Si resulta que no contiene errores envía una señal de confirmación hacia el emisor
 - Esta señal se denomina ACK (acrónimo del término inglés acknowledge: confirmación).
 - Si hay errores se envía hacia el emisor una señal de recepción errónea
 - Denominada NACK (por negative acknowledge).
 - Mientras espera la recepción de ACK ó de NACK el emisor mantiene el mensaje enviado en un buffer
 - Cuando recibe NACK vuelve a enviar el contenido del buffer
 - Si recibe un ACK copia en el buffer la trama ó bloque siguiente y procede a enviarla

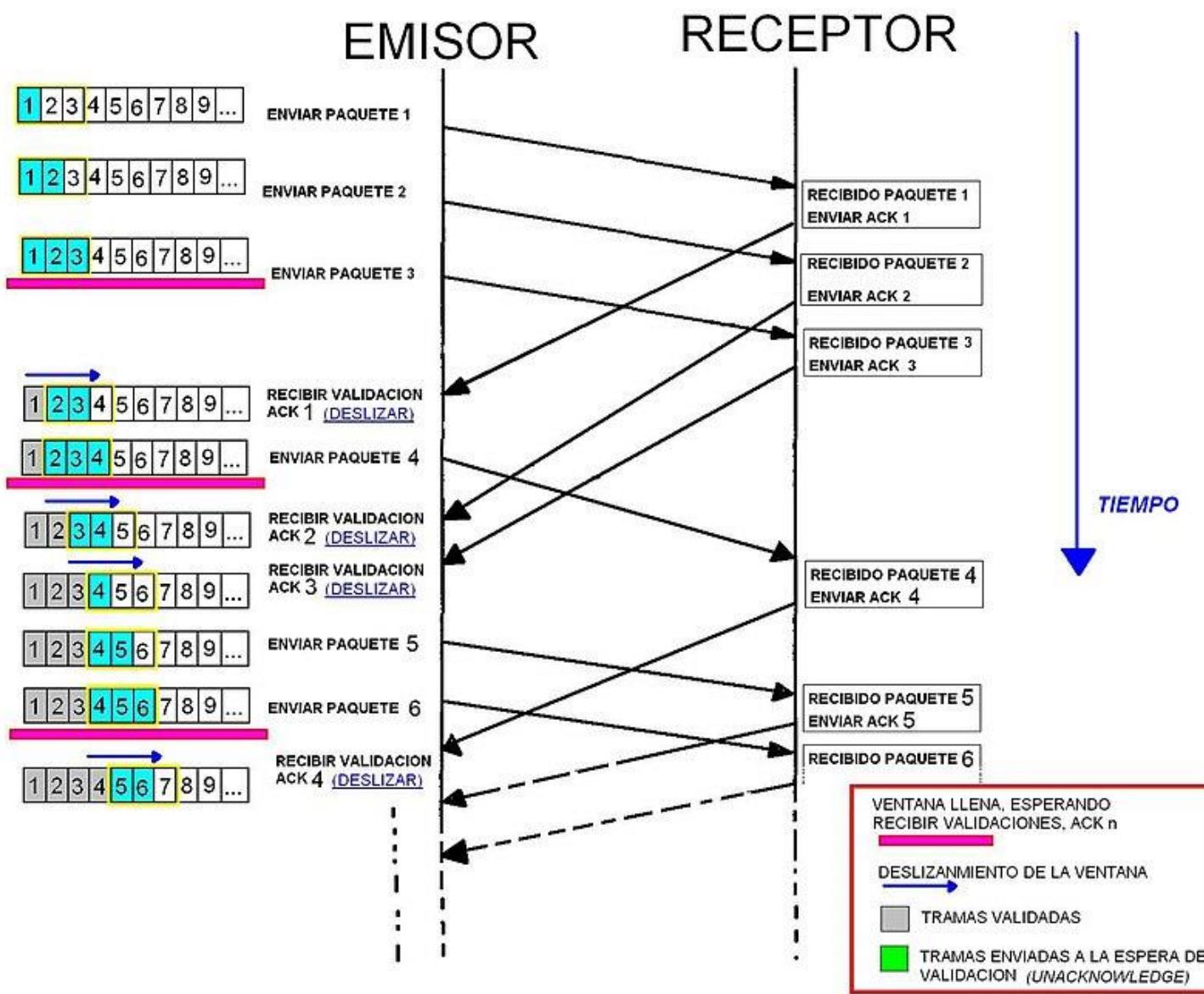
Control de flujo



Control de flujo

- El método de parada y espera tiene inconveniente de
 - Reducir el tiempo de utilización efectiva de los canales de comunicación
 - Cada mensaje debe ser confirmado individualmente
 - Todo se paraliza hasta que ello ocurre.
- Para corregir esto los métodos de ventana deslizante utilizan el mecanismo de:
 - Enviar continuamente la información sin esperar confirmación
 - Previamente se conviene en un número "m" que dará el número de mensajes al cabo de los cuales se va a enviar respuesta ACK ó NACK.
 - Cada bloque o trama contiene un número (o varios) de secuencia que la identifica.
 - Existen diversos métodos para enviar al ACK o el NACK, de los más conocidos:
 - Adelante y atrás N
 - En caso de error en el mensaje x, se pide que se retransmita la secuencia a partir de x retrocediendo $n = m - x$.
 - Rechazo selectivo
 - Se reenvía solamente la trama defectuosa

Control de flujo

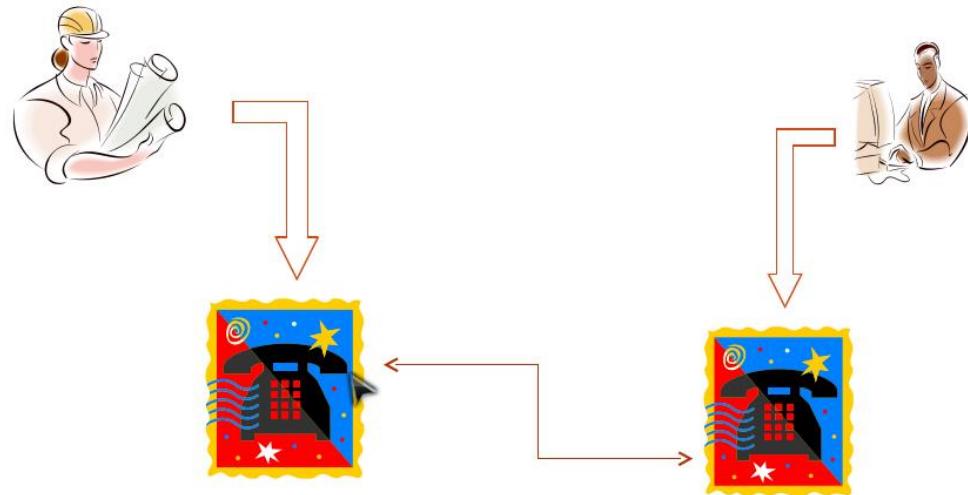


Control de flujo

- En muchos casos, y mientras espera la llegada de NACK ó ACK, el emisor arranca un temporizador.
 - El temporizador se detendrá al llegar cualquiera de las señales de confirmación
 - Si el temporizador llega a su término sin recibirlas pueden ocurrir dos cosas:
 - La primera consiste en abortar el proceso de comunicación dado que no hay respuesta del receptor
 - La segunda en enviar nuevamente la trama sin confirmar y arrancar nuevamente el temporizador.
 - Si esta situación se repite varias veces el sistema aborta la comunicación.
- En el receptor también se arranca un temporizador ya sea al recibir la trama o al enviar una señal ACK
 - Al vencer este tiempo el receptor procede a reenviar una señal de ACK u otra predeterminada
 - Si se repite esto un cierto número de veces se procede a abortar la comunicación pues no se recibe respuesta del emisor.
- Generalmente tanto el receptor como el emisor dispondrán de un contador
 - Para determinar el número de veces que se ha intentado retransmitir una trama sin éxito.
 - De alcanzar el contador el valor prefijado se procede a abortar la comunicación.

Control de errores

Introducción



- Las causas por las que la señal electromagnética se deteriora al viajar por el canal de comunicación son:
 - Distorsión
 - Atenuación
 - Limitación del ancho de banda
 - Ruido
 - Interferencia
 - Diafonía.
- Esta degradación de la señal puede hacer que se reciban en el receptor un carácter distinto al que fue emitido por el extremo transmisor, entonces se ha producido un error.

Definición

- Es imposible evitar que ocurran errores
 - Un buen diseño los minimizará.
 - En primer lugar, determinar la presencia de los errores
 - Aquí es donde aparecen las técnicas de detección de errores
 - Luego tratar de corregirlos, lo que da lugar a la corrección de errores
- La denominación genérica de estas técnicas es **Control de Errores**.



Detección de errores

- La detección de errores consiste en monitorear la información recibida y a través de técnicas implementadas en el Codificador de Canal ya descrito, determinar si un carácter, caso asincrónico, o un grupo de datos, caso sincrónico, presentan algún o algunos errores.
 - Las técnicas más comunes son:
 - Redundancia.
 - Codificación de cuenta exacta.
 - Chequeo de paridad vertical (VRC).
 - Chequeo de paridad horizontal (LRC).
 - Chequeo de paridad bidimensional (VRC/LRC).
 - Checksum
 - Chequeo de redundancia cíclica (CRC)

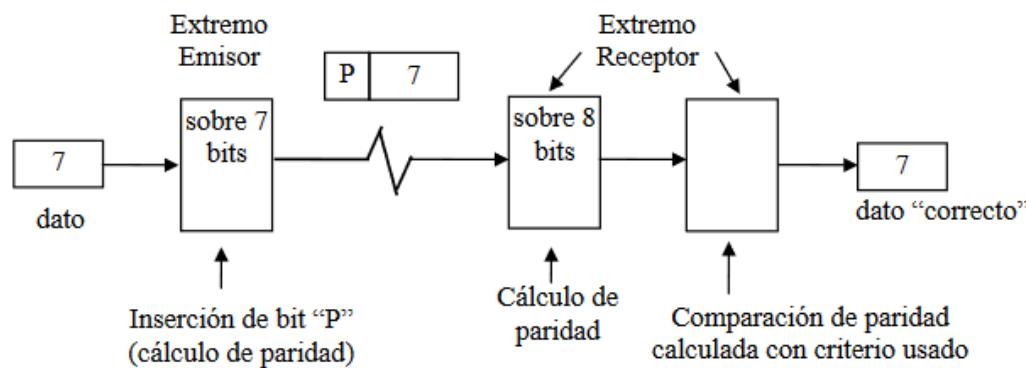
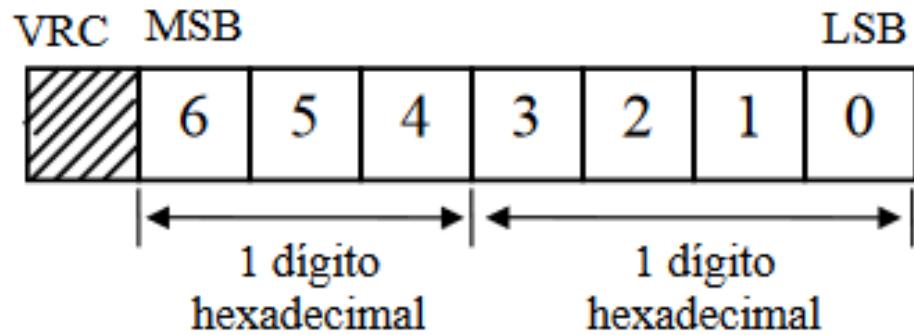


Detección de errores

<u># Carácter</u>	<u>Carácter</u>	<u>Valor decimal</u>
1	A	65
2	M	77
3	E	69
4	R	82
5	I	73
6	C	67
7	A	65
CHEKSUM		498

- Checksum
 - Método simple orientado al mensaje
 - Los valores (por ejemplo, decimales) que corresponden a cada carácter en el código ASCII son sumados y la suma es enviada al final del mensaje.
 - En el extremo receptor se repite el procedimiento de sumar los valores de los caracteres y se compara el resultado obtenido con el recibido al final del mensaje

Detección de errores



- Chequeo de paridad vertical o paridad de carácter (VRC).
 - Este método, como todos los que siguen, hace uso del agregado de bits de control.
 - Se trata de la técnica más simple usada en los sistemas de comunicación digitales (Redes Digitales, Comunicaciones de Datos)
 - Paridad par
 - Paridad impar

Detección de errores

- Chequeo de paridad bidimensional
 - Es la combinación de verificación de paridad vertical y paridad horizontal
 - Proporciona mayor protección
 - No supone gran consumo de recursos
 - Aunque tiene la misma sencillez conceptual de los métodos de paridad lineal, es más complicado y por ello menos popular.

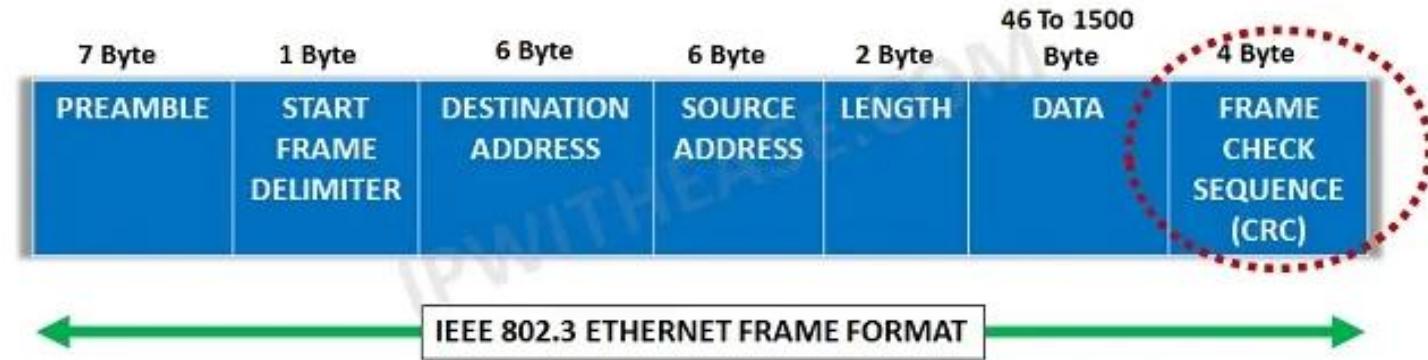
Paridad de carácter computada en el receptor	bit de paridad transmitido								Carácter #1 Carácter #2 Carácter #3 Carácter #4 Carácter #5 Carácter #6 Carácter #7
	8	7	6	5	4	3	2	1	
0	0	1	0	0	0	0	0	1	Carácter #1
0	0	1	0	0	1	1	0	1	Carácter #2
1	1	1	0	0	0	1	0	1	Carácter #3
1	1	1	0	1	0	0	1	0	Carácter #4
1	1	1	0	0	1	0	0	1	Carácter #5
1	1	1	0	0	0	0	1	1	Carácter #6
1*	0	1	0	1	0	0	0	1	Carácter #7
0	0	1	0	1	0	0	0	0	Paridad de columna recibida
1	0	1	0	0*	1	0	0	0	Paridad de columna computada en el extremo receptor

Diferencias de paridad

Detección de errores

- Métodos de paridad
 - Los métodos basados en el uso de paridad son sencillos de comprender y de implementar
 - Suministran cierto grado de protección contra los errores
 - Son limitados
 - Su efectividad es cuestionable en determinadas aplicaciones.
 - Por ello se utilizan solamente cuando resulta muy complicado ó muy costoso implementar otros métodos.
 - Paridad vertical requiere que cada carácter lleve su protección contra errores
 - Adecuado en entornos asíncronos
 - En entornos síncronos el uso de tantos bits de detección de errores consume un porcentaje importante de la capacidad del canal y resulta oneroso.
 - Por ello es necesario, en entornos síncronos, emplear métodos que tengan en cuenta dos factores importantes:
 - Detección más segura de errores
 - Eficiencia

Detección de errores



- Métodos de Código de Redundancia Ciclífica (CRC)
 - Cumplen con requisitos de
 - Detección más segura de errores
 - Eficiencia
 - Se basan en propiedades matemáticas de los códigos empleados para la transmisión de datos
 - Principio de funcionamiento
 - Deseamos transmitir al extremo receptor, mediante un canal de comunicación muy vulnerable a errores, un número.
 - Dadas las circunstancias es muy posible que, si enviamos, digamos el número 23, llegue al extremo receptor un número distinto
 - Una solución es elegir un número clave, por ejemplo el 5.
 - Ahora dividimos el número a transmitir entre la clave y calculamos el resto: $23/5 = 4$ resto 3
 - Enviamos conjuntamente con el 23 el resto, o sea, transmitimos 233.
 - En el extremo receptor se efectúa el proceso inverso, supongamos que hemos recibido 253 al dividir 25/5 el resto es 0 y 0 es distinto de 3 lo que indica error.

Detección de errores

- CRC
 - Emplean el principio de las propiedades de la operación módulo (se define módulo de dos números $a \text{ mod } b$ al resto de dividir a por b).
 - Para ello se considera la cadena de bits a transmitir como el conjunto de coeficientes de un polinomio
 - Por ejemplo, al enviar 1100100110, el polinomio equivalente $P(x)$ es: $P(x) = x^9 + x^8 + x^5 + x^2 + x$.
- Se debe ahora especificar la clave para efectuar la división.
- La selección de esta clave es esencial para la capacidad de respuesta del código frente a los diversos tipos de errores.
- El CCITT especifica algunas claves, que como se van a emplear para dividir un polinomio serán también polinomios, denominados polinomio generador.
 - En el CRC denominado CRC-16 correspondiente a la norma CCITT V.41, se utiliza el siguiente polinomio generador:

$$G(x) = x^{16} + x^{12} + x^5 + x^0$$

Detección de errores

- CRC

- El procedimiento es el siguiente:

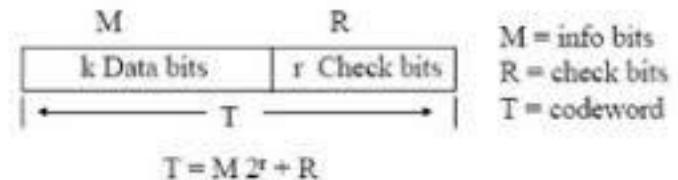
- Se toma el polinomio de datos $P(x)$
 - Se multiplica por x^k , (el número de ceros que se agrega a $P(x)$,
 - donde k es el exponente más alto de $G(x)$

- Este polinomio así construido se divide por $G(x)$, división en módulo 2
 - Se obtiene un polinomio resto $R(x)$, llamado BCS (Block Character Sequence)
 - Se procede a enviar el polinomio $T(x)$ construido así:

$$T(x) = x^k P(x) + R(x)$$

- En el extremo receptor se procederá a extraer lo que se supone es $x^k P(x)$
 - Se divide nuevamente por $G(x)$
 - Se calcula un polinomio resto que si coincide con el $R(x)$ recibido indicará que no hay errores.

Cyclic Redundancy Checks (CRC)



M = info bits

R = check bits

T = codeword

Detección de errores

- CRC (División módulo 2)
 - La operación de división empleada en CRC se denomina de módulo 2.
 - Es diferente de la división binaria:
 - Las restas durante la división(o sea la obtención del resto parcial o final) no son aritméticas sino módulo 2
 - Lo que significa una operación XOR entre los dígitos binarios que se están restando(1 y 0 da 1, 0 y 1 da 1, 0 y 0 dá 0, 1 y 1 dá 0).
 - Si el primer bit del resto parcial es 1 y queda uno o más bits del dividendo
 - Se baja el primer bit de la izquierda no usado y se hace el XOR.
 - En caso contrario se bajan bits de la izquierda del dividendo
 - Hasta que este resto con los bits bajados esté encabezado por un 1 y tenga la misma longitud del divisor.
 - De no lograrse el resto con todos los bits bajados será el resto final de la división

Detección de errores

- CRC Ejemplo
 - Determinar el BSC(Block Character Sequence), para los siguientes polinomios generadores de datos y CRC

$$\text{datos } P(x) = x^7 + x^5 + x^4 + x^2 + x^1 + x^0 \quad \text{ó} \quad 10110111$$

$$\text{CRC } G(x) = x^5 + x^4 + x^1 + x^0 \quad \text{ó} \quad 110011$$

- Solución
 - Primero $P(x)$ es multiplicado por el número de bits en el código CRC, 5

$$\begin{aligned}x^5(x^7 + x^5 + x^4 + x^2 + x^1 + x^0) &= x^{12} + x^{10} + x^9 + x^7 + x^6 + x^5 \\&= 1011011100000\end{aligned}$$

- Al dividir este polinomio por $G(x)$ obtenemos $R(x)$ o BCS que resulta

Detección de errores

$$\begin{array}{r} 110011 \quad 11010111 \\)1011011100000 \\ \underline{110011} \\ 111101 \\ \underline{110011} \\ 111010 \\ \underline{110011} \\ 100100 \\ \underline{110011} \\ 101110 \\ \underline{110011} \\ 111010 \\ \underline{110011} \\ 1001 = \text{Resto} \end{array}$$

- CRC (ejemplo)
 - Resto 01001 significa $R(x) = 0x^4 + 1x^3 + 0x^2 + 0x + 1$
 - Se transmite entonces
$$T(x) = x^k P(x) + R(x) \text{ o sea } 1011011101001$$
 - Suponiendo que se reciba $T_R(x)$
1011011101001 (lo mismo que se transmitió)
 - Al dividir $T_R(x)$ por $G(x)$ el resto será cero
 - Indicando que se recibió correctamente
 - O bien se separa lo que se supone es $x^k P(x)$ se divide por $G(x)$ y el resto se compara con $R(x)$

Corrección de errores

- Detectar los errores no es suficiente, hay que corregirlos.
- Dos formas principales de corrección de errores son:
 - Requerimiento automático de repetición: (ARQ) (Automatic Request for Repeat).
 - Pare y espere (stop and wait ARQ).
 - Envío continuo (Continuos ARQ).
 - Adelante y Atrás N
 - Rechazo selectivo
 - Estos métodos se analizaron en control de flujo
 - Corrección de errores hacia adelante: FEC (Forward Error Correction).
 - Bloque
 - Paridad Bidimensional
 - Hamming
 - Otros



Corrección de errores

- Se basan en la idea de reconstruir la información deteriorada por los errores
- La reconstrucción tiene lugar en el equipo receptor
 - Deben emplearse en los códigos un gran número de bits lo que disminuye la efectividad del código.
- En la detección por paridad bidimensional se menciona que este método puede corregir el error si solo se presenta uno
- Otro método empleando paridad es el Código Hamming que igualmente solo puede corregir cuando se presenta un solo error

Corrección de errores

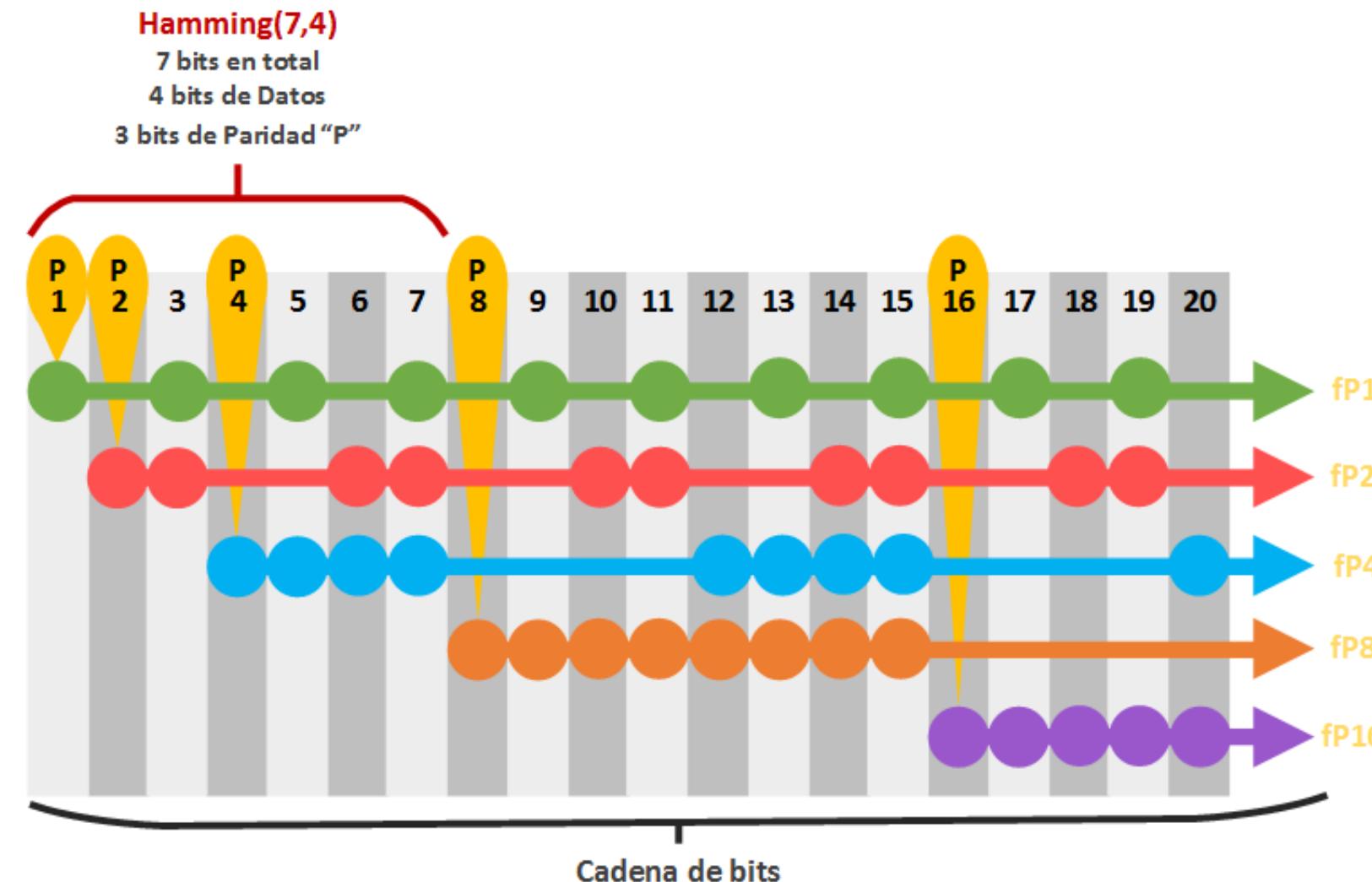
Paridad de carácter computada en el receptor	bit de paridad transmitido								-
	8	7	6	5	4	3	2	1	
0	0	1	0	0	0	0	0	1	Carácter #1
0	0	1	0	0	1	1	0	1	Carácter #2
1	1	1	0	0	0	1	0	1	Carácter #3
1	1	1	0	1	0	0	1	0	Carácter #4
1	1	1	0	0	1	0	0	1	Carácter #5
1	1	1	0	0	0	0	1	1	Carácter #6
1*	0	1	0	1	0	0	0	1	Carácter #7
0	0	1	0	1	0	0	0	0	Paridad de columna recibida
1	0	1	0	0*	0	0	0	0	Paridad de columna computada en el extremo receptor

Diferencias de paridad

- Paridad bidimensional
 - Requiere de una matriz de paridad

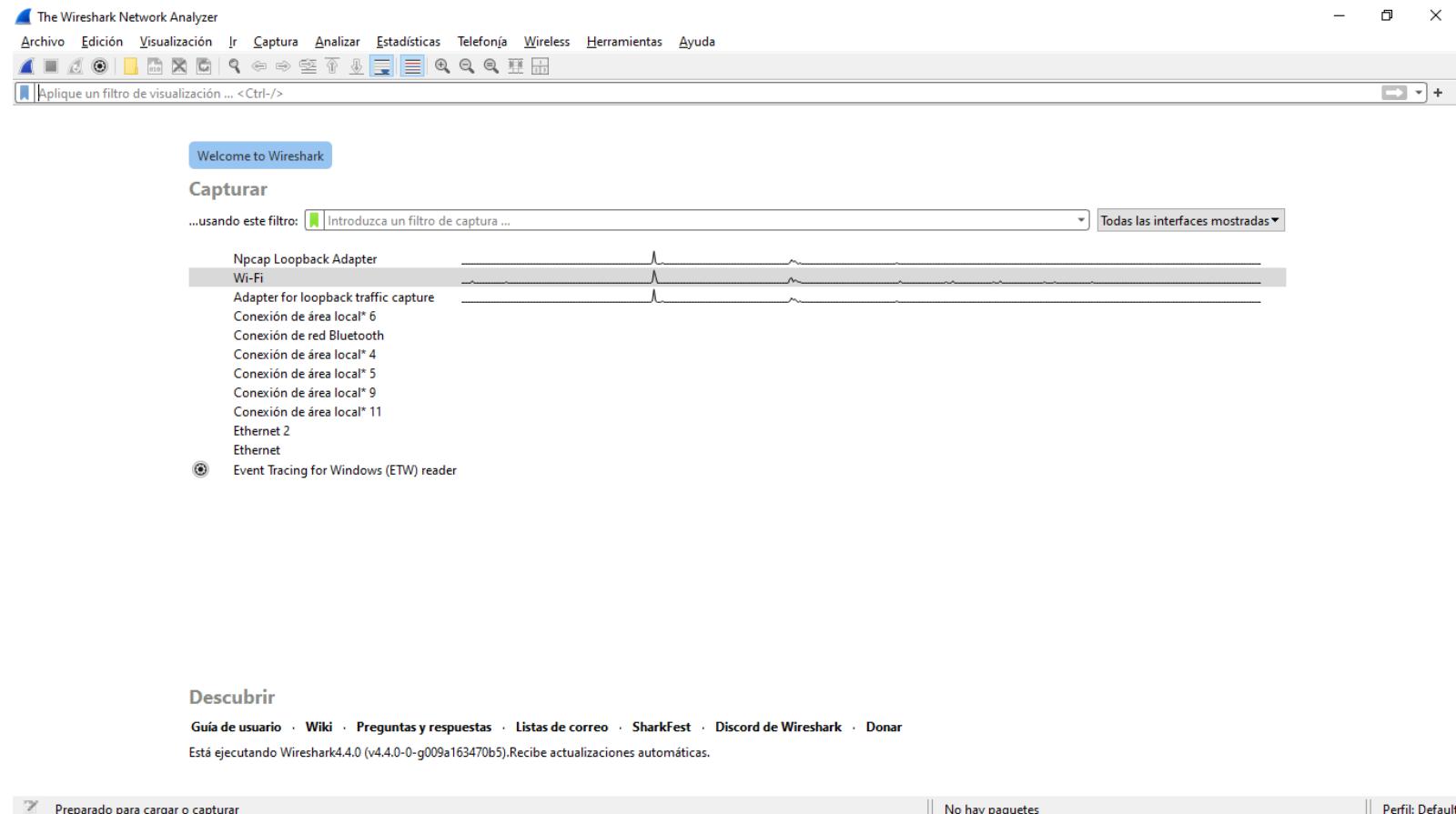
Corrección de errores

- Código de Hamming



Wireshark

Inicio de captura



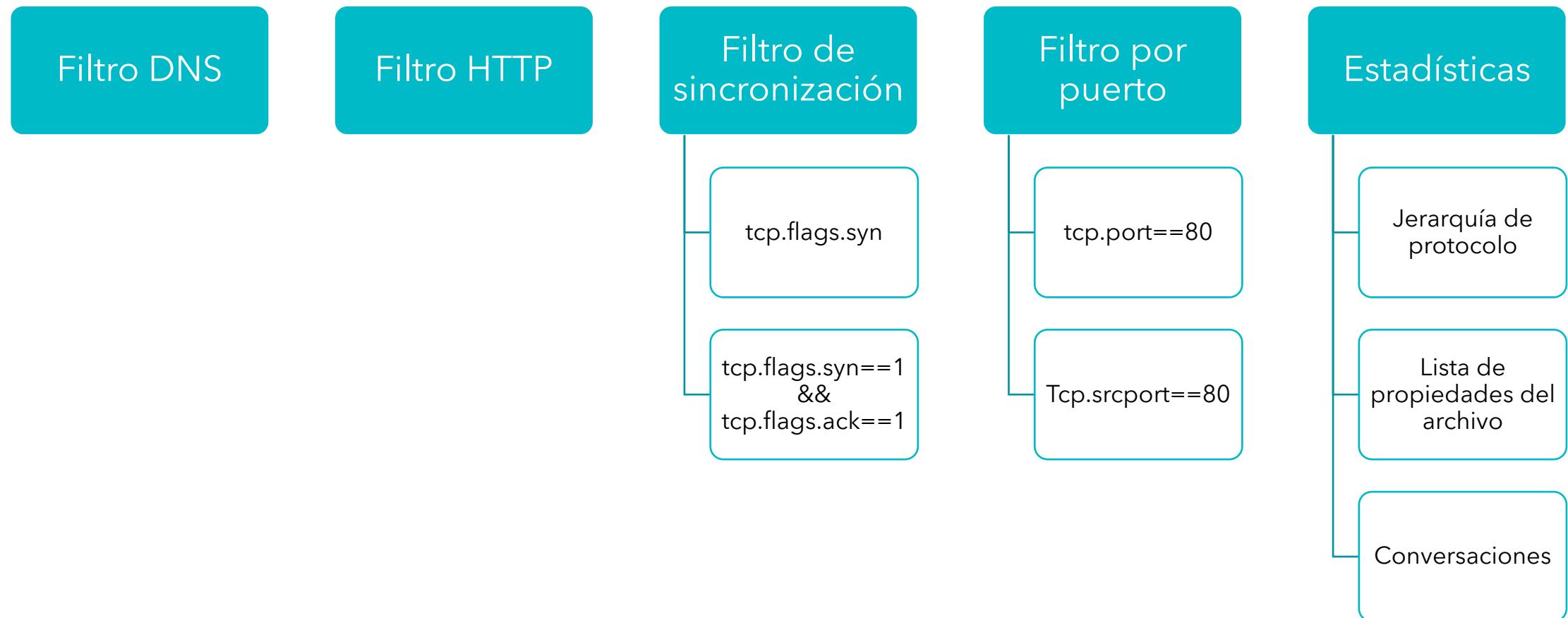
- Elegir la interfaz de captura
- Elegir filtro de captura si así se desea

Partes de Wireshark

The screenshot shows the Wireshark interface with the following components:

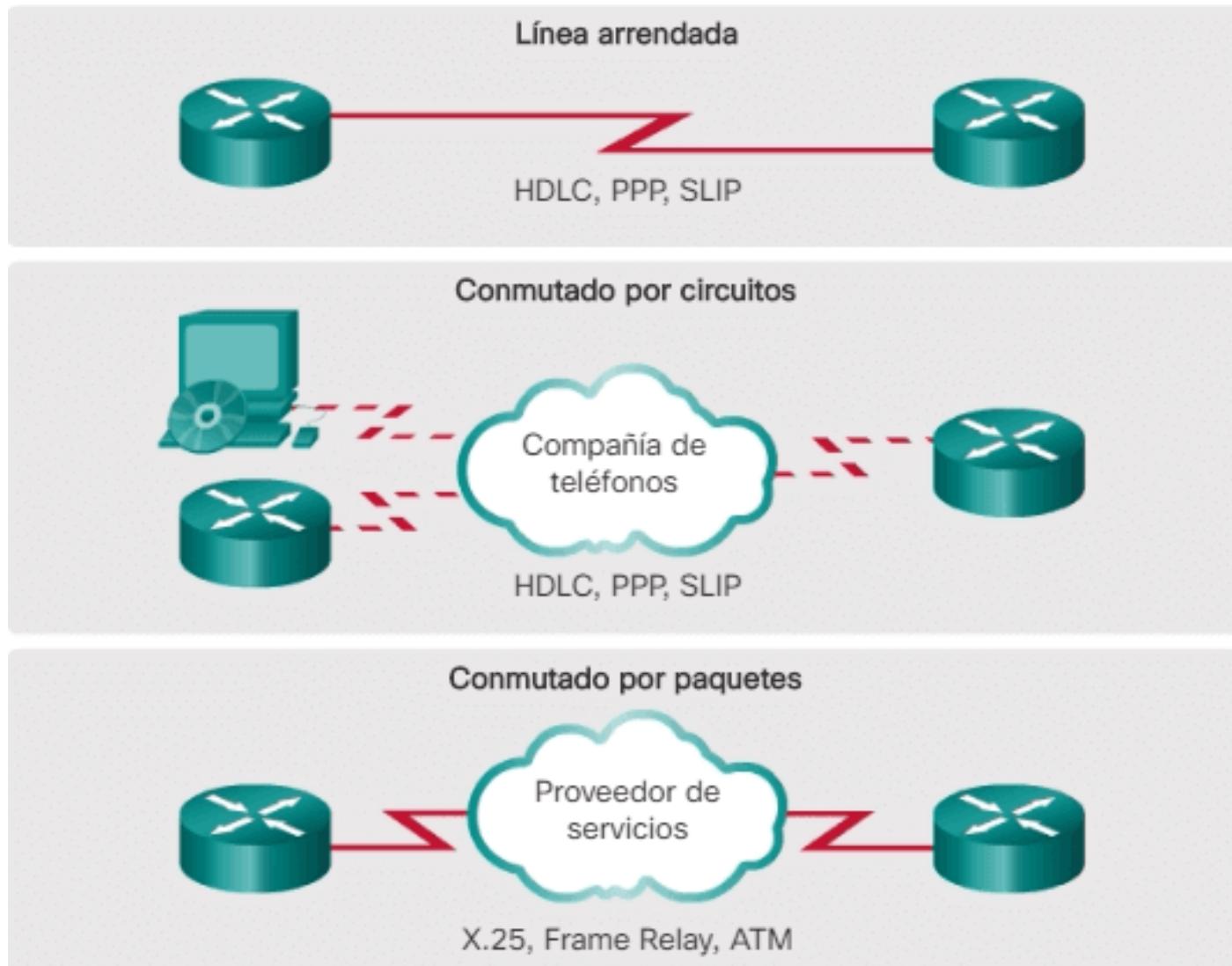
- Top Bar:** *Wi-Fi, Archivo, Edición, Visualización, Ir, Captura, Analizar, Estadísticas, Telefonía, Wireless, Herramientas, Ayuda.
- Toolbar:** Includes icons for file operations (New, Open, Save, Print, Export, Find, Copy, Paste, etc.) and search.
- Search Bar:** Aplique un filtro de visualización ... <Ctrl-/>
- Main Window:** Displays a list of network frames. Frame 1 is selected, showing details:
 - Frame 1: 212 bytes on wire (1696 bits), 212 bytes captured (1696 bits) on interface *Wi-Fi at 0.000000 seconds ago
 - Ethernet II, Src: 22:17:a0:8d:46:6a (22:17:a0:8d:46:6a), Dst: IPv6mcast_fb (ff02::fb)
 - Internet Protocol Version 6, Src: fe80::10d1:d1:62f4:563f, Dst: ff02::fb
 - User Datagram Protocol, Src Port: 5353, Dst Port: 5353
- Protocol Tree:** Multicast Domain Name System (query)
 - Transaction ID: 0x0000
 - Flags: 0x0000 Standard query
 - Questions: 4
 - Answer RRs: 2
 - Authority RRs: 0
 - Additional RRs: 0
 - Queries
 - _companion-link._tcp.local: type PTR, class IN, "QM" question
 - _homekit._tcp.local: type PTR, class IN, "QM" question
 - _hap._tcp.local: type PTR, class IN, "QM" question
 - _hap._udp.local: type PTR, class IN, "QM" question
 - Answers
- Hex Editor:** Shows the raw hex and ASCII representation of the selected frame's payload.
- Bottom Status Bar:** wireshark_Wi-FiK2I9V2.pcapng, Paquetes: 69, Perdido: 0 (0.0%), Perfil: Default.

Filtro de display



HDLC y otros protocolos

Protocolos de encapsulación WAN

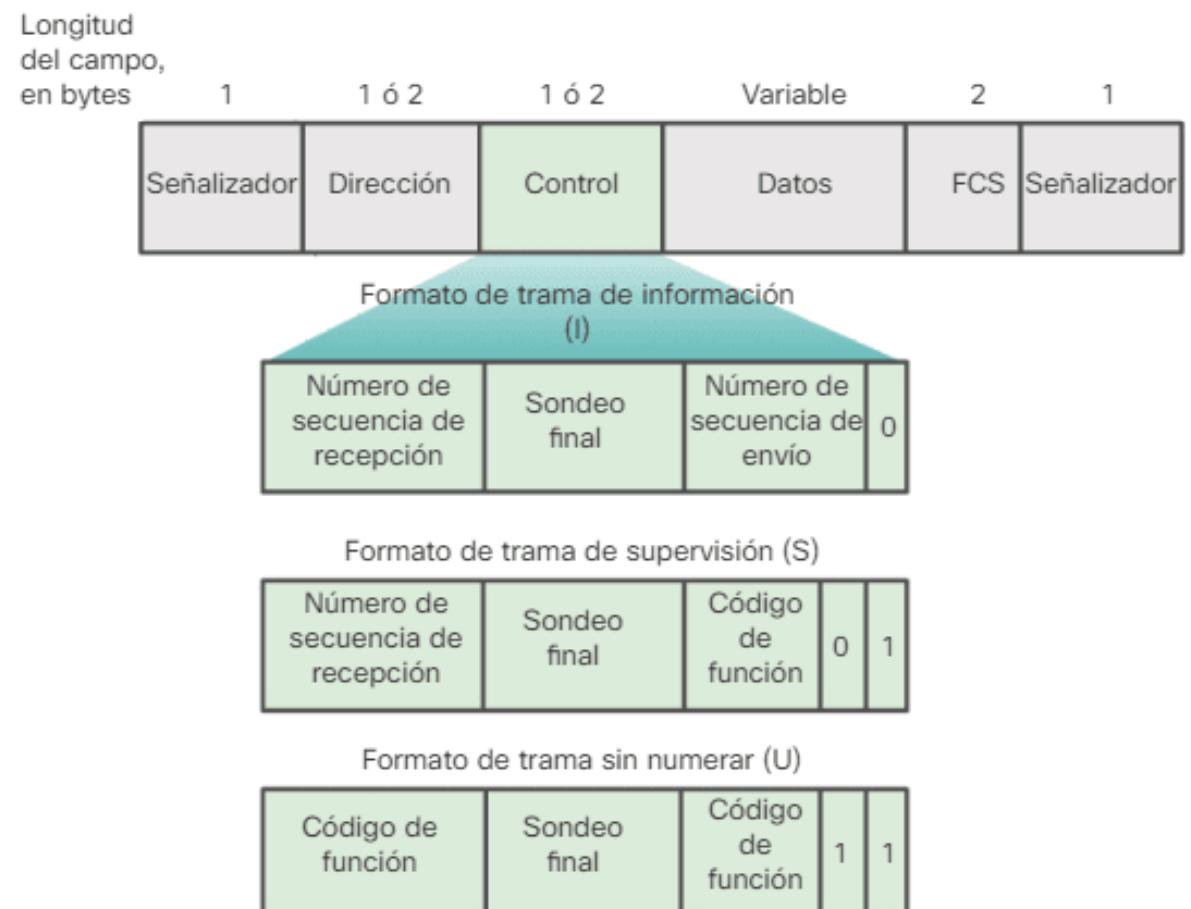


- En cada conexión WAN, se encapsulan los datos en las tramas antes de cruzar el enlace WAN.
- Para asegurar que se utilice el protocolo correcto, se debe configurar el tipo de encapsulación de capa 2 correspondiente.
- La opción de protocolo depende de la tecnología WAN y el equipo de comunicación.

Tipos de Protocolo WAN

- **HDLC:**

- Es el tipo de encapsulación predeterminado en las conexiones
 - Punto a punto
 - Enlaces dedicados
 - Conexiones comutadas por circuitos.
- HDLC es la base para PPP síncrono que usan muchos servidores para conectarse a una WAN, generalmente Internet.



Otros tipos de protocolos WAN

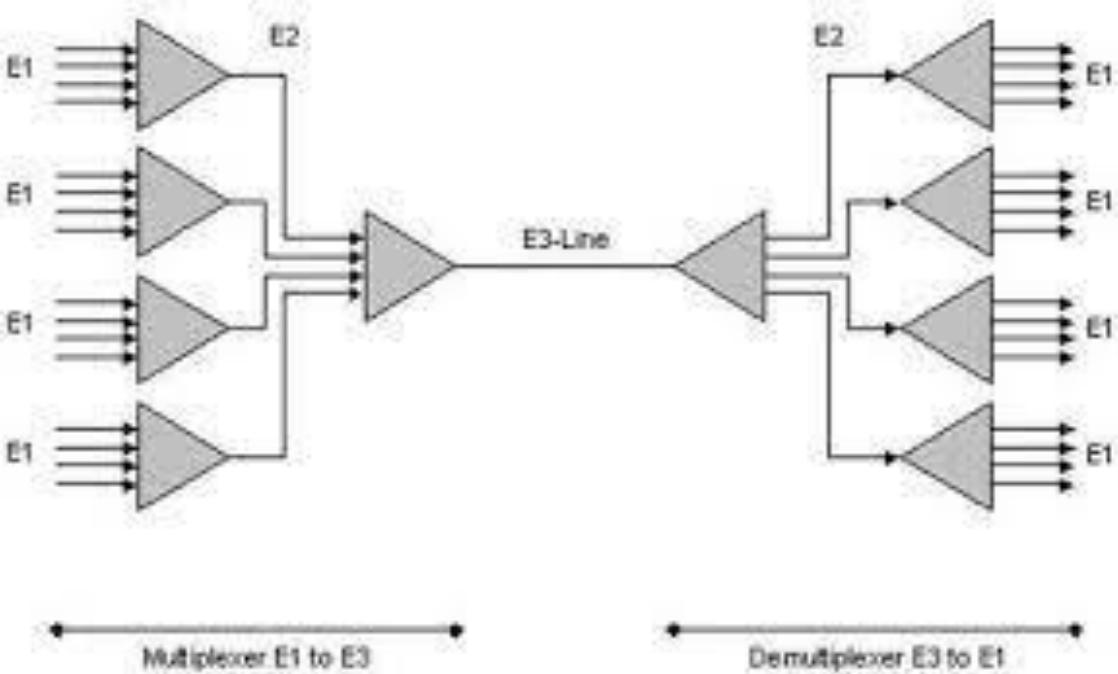
- **PPP:** proporciona conexiones de router a router y de host a red a través de circuitos síncronos y asíncronos. PPP funciona con varios protocolos de capa de red, como IPv4 e IPv6. Utiliza el protocolo de encapsulación HDLC, pero también tiene mecanismos de seguridad incorporados como PAP y CHAP.
- **Protocolo de Internet de línea serial (SLIP):** es un protocolo estándar para conexiones seriales punto a punto mediante TCP/IP. PPP reemplazó ampliamente al protocolo SLIP.
- **Procedimiento de acceso al enlace balanceado (LAPB) X.25:** es un estándar del UIT-T que define cómo se mantienen las conexiones entre un DTE y un DCE para el acceso remoto a terminales y las comunicaciones por computadora en las redes de datos públicas.
- **Frame Relay:** es un protocolo de capa de enlace de datos conmutado y un estándar del sector que maneja varios circuitos virtuales. Frame Relay elimina algunos de los procesos prolongados (como la corrección de errores y el control del flujo) empleados en X.25.
- **ATM:** es el estándar internacional de retransmisión de celdas en el que los dispositivos envían varios tipos de servicios (como voz, video o datos) en celdas de longitud fija (53 bytes). Las celdas de longitud fija permiten que el procesamiento se lleve a cabo en el hardware, lo que disminuye las demoras en el tránsito.

Multicanalización

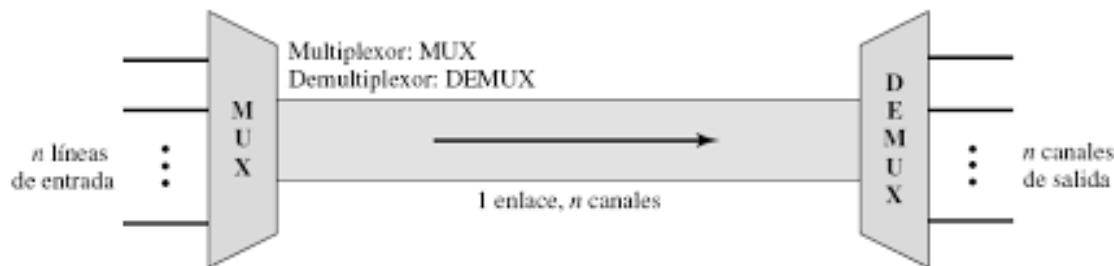
Multiplexión

Definición

- La Multiplexación es la combinación de dos o más canales de información en un solo medio de transmisión usando un dispositivo llamado multiplexor.
 - Procedimiento por el cual diferentes canales pueden compartir un mismo medio de transmisión de información.



Objetivo de la multicanalización



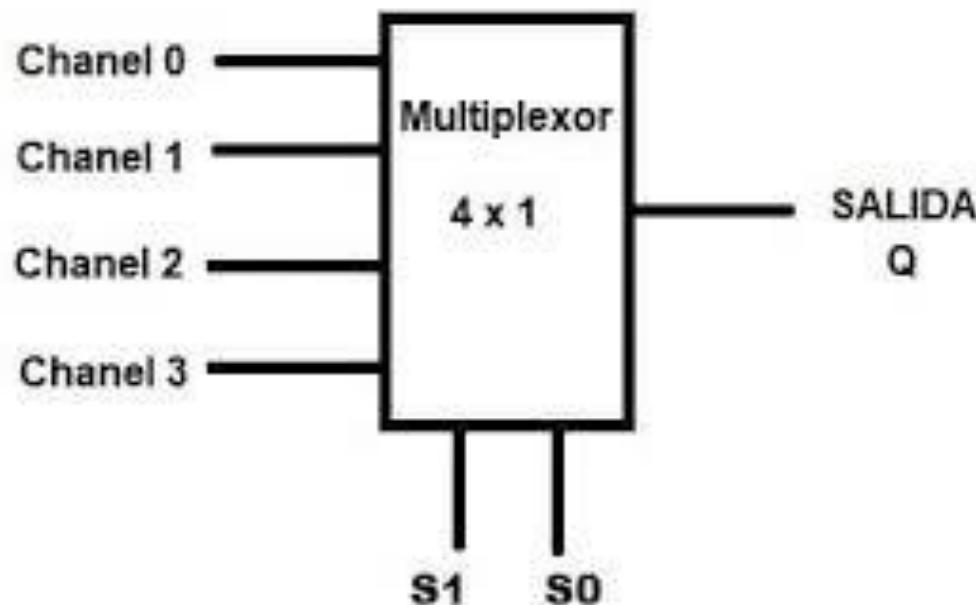
- Compartir la capacidad de transmisión de datos sobre un mismo enlace para aumentar la eficiencia (sobre todo en líneas de grandes distancias).
- Minimizar la cantidad de líneas físicas requeridas y maximizar el uso del ancho de banda de los medios

Multicanalización ¿qué es?

- Optimización de la utilización del medio de transmisión
 - Mediante un conjunto de técnicas que permite la transmisión simultanea de múltiples señales a través de un único enlace



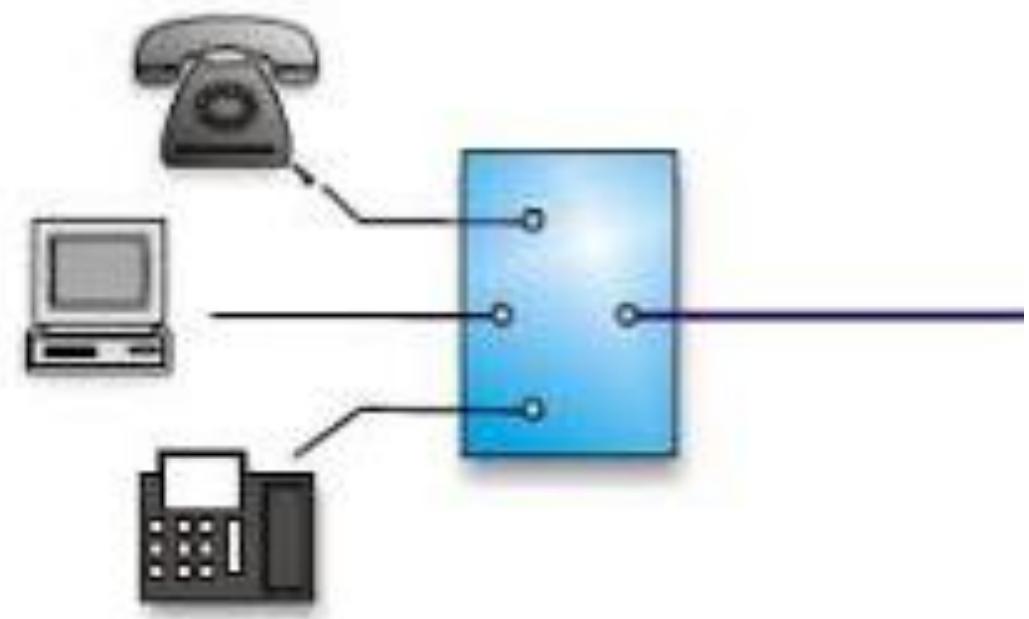
Definiendo un multiplexor



- Es un dispositivo que puede recibir varias entradas y transmitirlas por un medio de transmisión compartido
- Divide el medio de transmisión en múltiples canales, para que varios nodos puedan comunicarse al mismo tiempo.

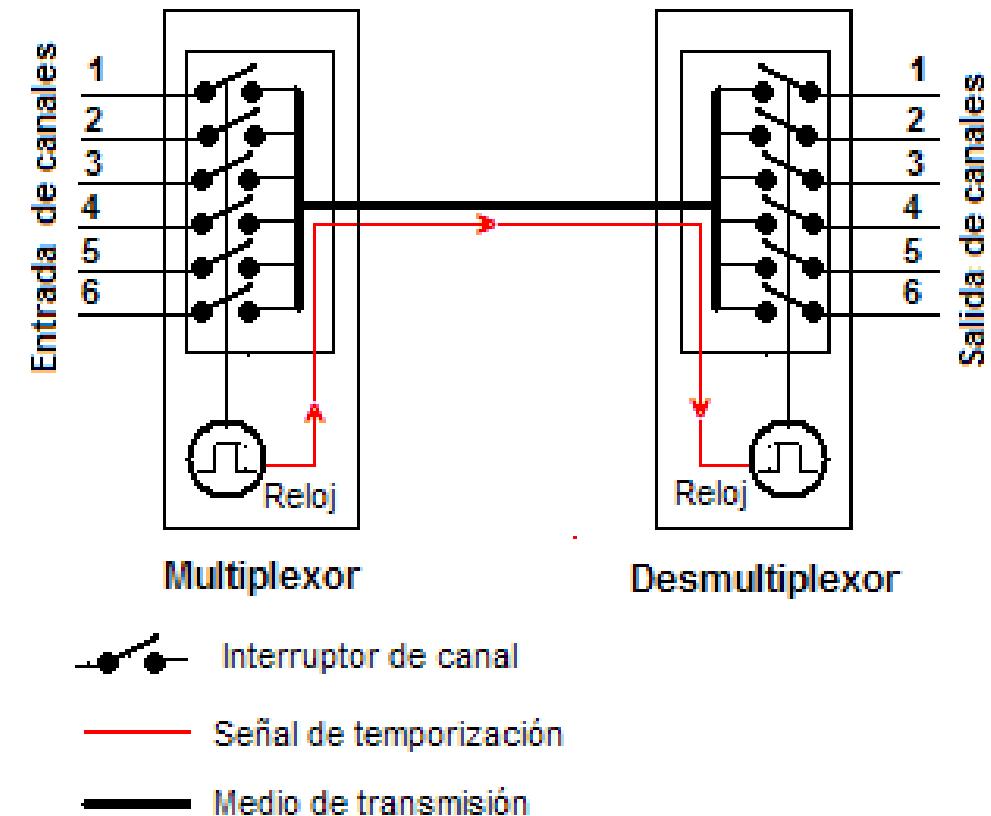
Función de un multiplexor

- La función de un multiplexor da lugar a diversas aplicaciones:
 - Selector de entradas.
 - Serializador: Convierte datos de paralelo a serial.
 - Transmisión multiplexada: En una misma línea de conexión, transmite diferentes datos de distinta procedencia.
 - Realización de funciones lógicas: Utilizando inversores y conectando a 0 ó 1 las entradas se puede diseñar funciones de un modo más compacto, que utilizando puertas lógicas.



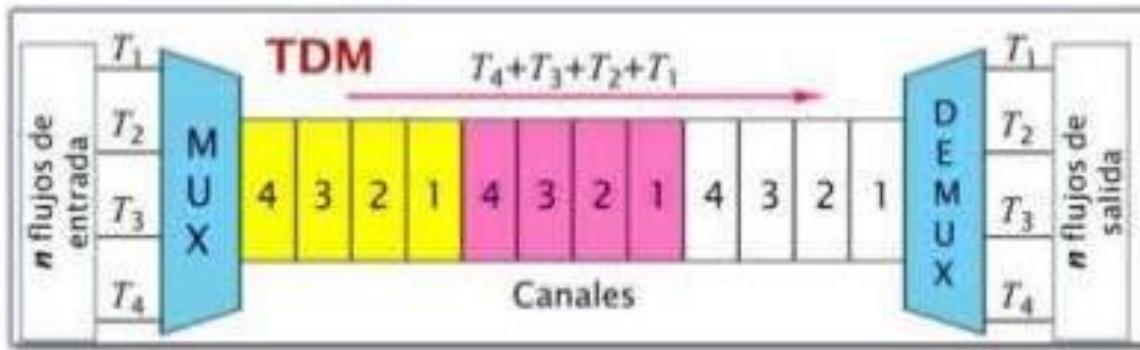
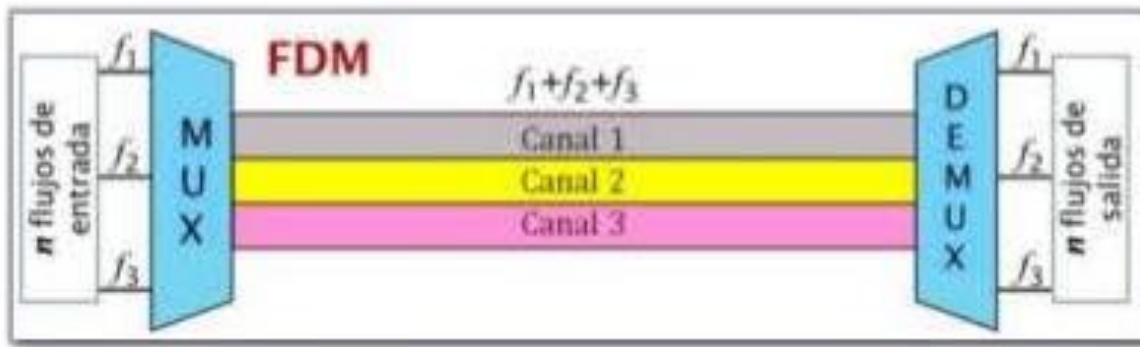
Implementación de la multicanalización

- Las entradas de 6 canales llegan a los interruptores de canal controlados por una señal de reloj
 - Cada canal es conectado al medio de Tx durante un tiempo determinado por la duración de los impulsos del reloj.
 - El Desmultiplexor realiza la función inversa: conecta al medio de Tx, secuencialmente con la salida de cada uno de los 5 canales mediante interruptores controlados por el reloj del D.
 - El reloj del extremo receptor funciona de forma sincronizada con el del Multiplexor mediante señales de temporización que son trasmitidas a través del propio medio de Tx



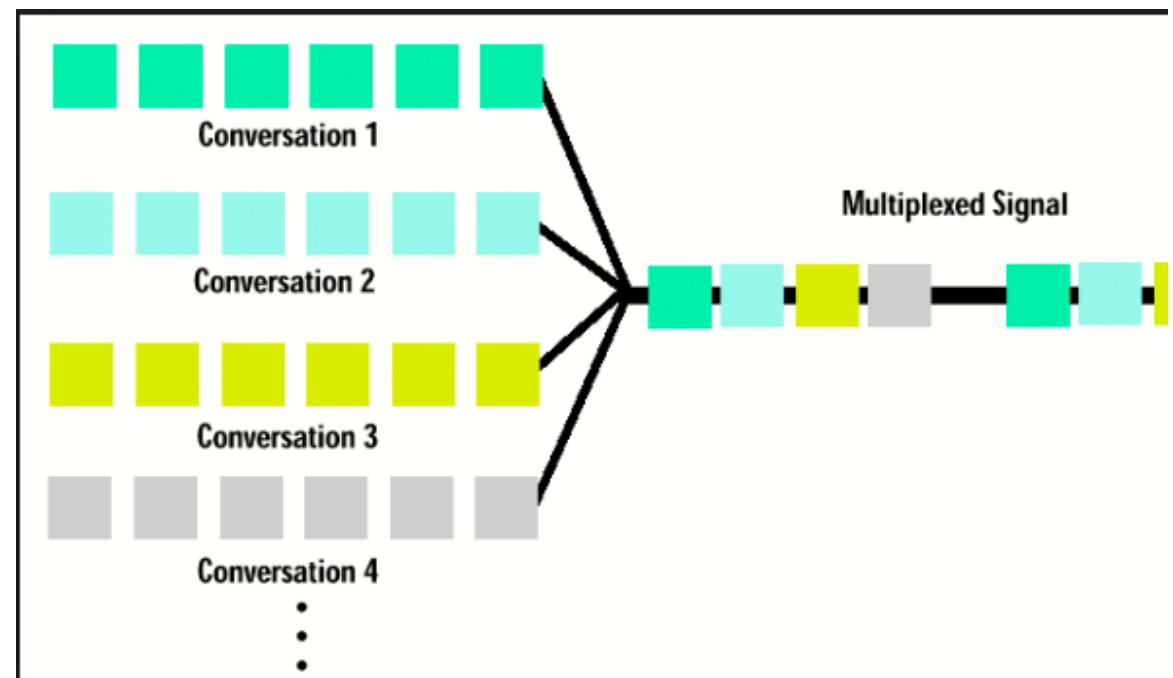
Métodos de multicanalización

- La Multiplexación o multicanalización es la transmisión de información, de más de una fuente a más de un destino, por el mismo medio de transmisión.
- Los principales métodos de realizar este proceso son:
 - La multiplexación de división de frecuencia (FDM: Frequency Division Multiplexing),
 - La multiplexación por división de código (CDM: Coded Division Multiplexing),
 - La multiplexación por división de longitud de onda (WDM: Wavelength Division Multiplexing)
 - La multiplexación por división de tiempo (TDM: Time Division Multiplexing)



Multicanalización por División de Tiempo (TDM)

- La multiplexación por división de tiempo es una técnica para compartir un canal de transmisión entre varios usuarios.
- Consiste en asignar a cada usuario, durante unas determinadas "ranuras de tiempo", la totalidad del ancho de banda disponible.

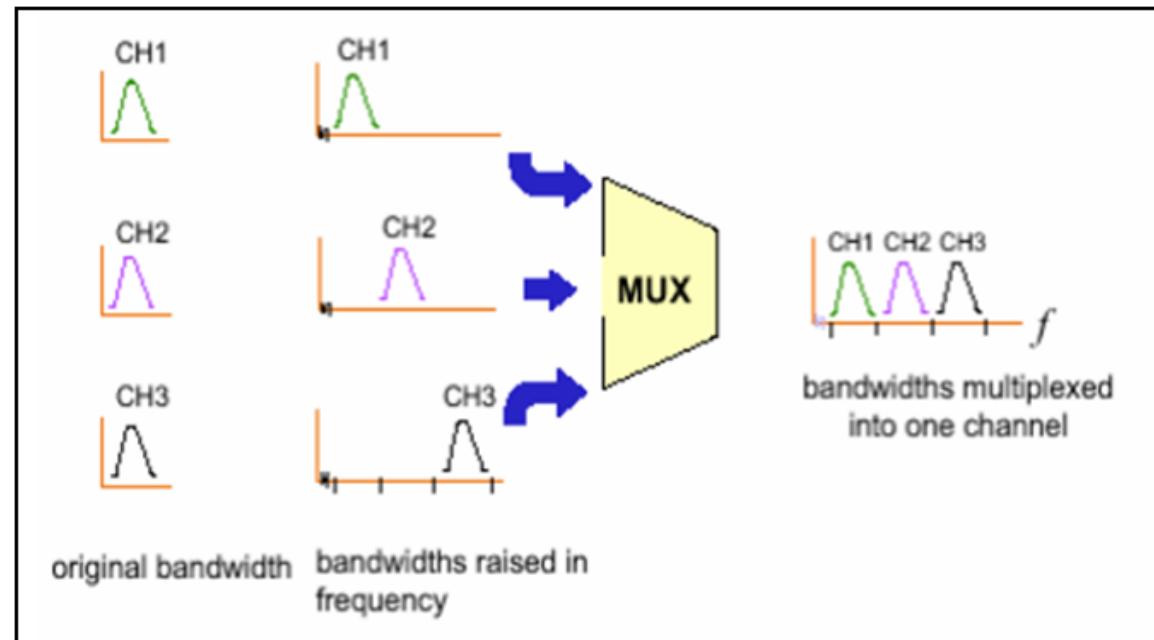


Multicanalización por división de tiempo (TDM)

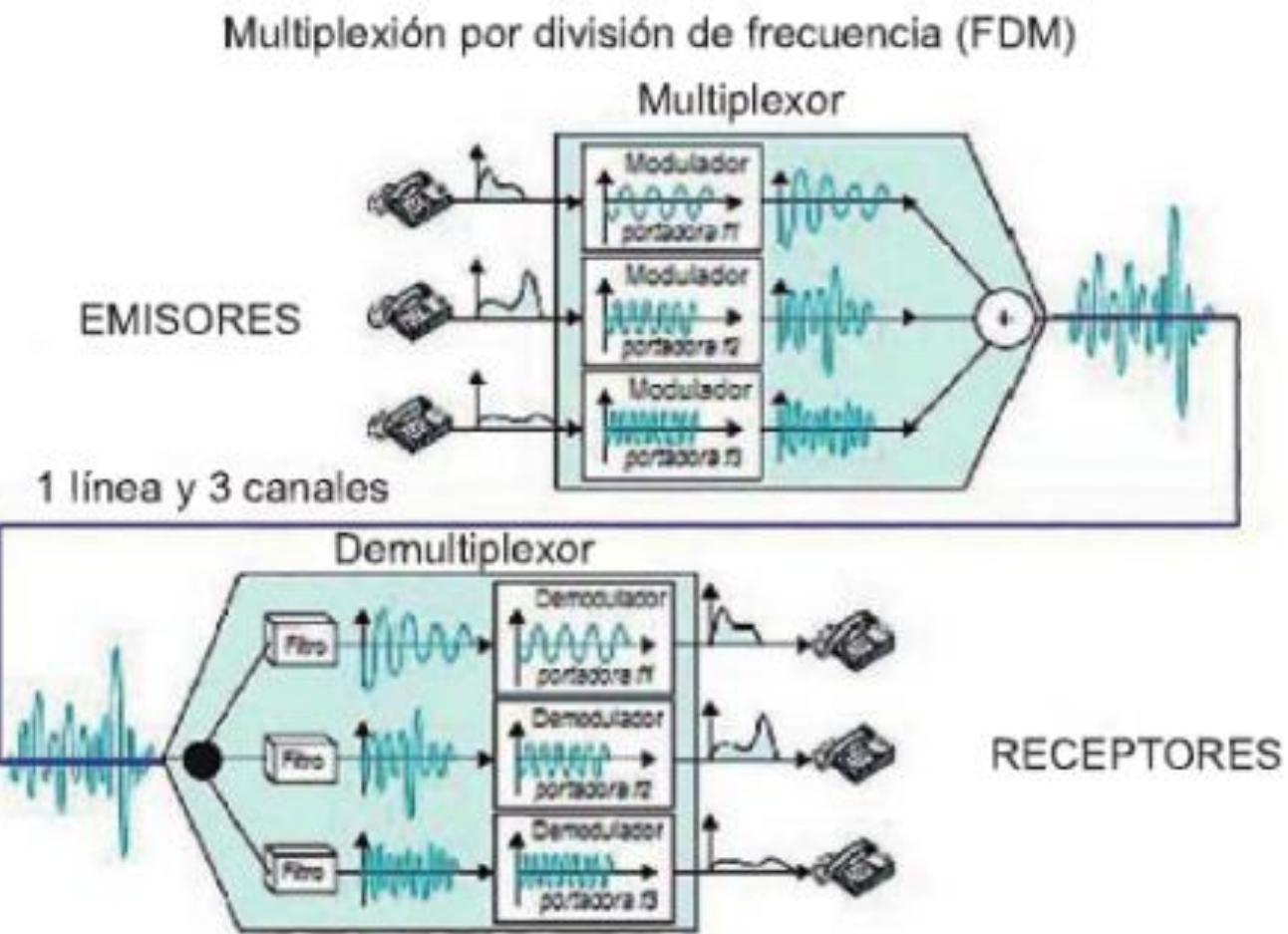
- Características
 - Consiste en ocupar un canal de transmisión a partir de distintas fuentes, mejor aprovechamiento del medio de transmisión.
 - El ancho de banda total del medio de transmisión es asignado a cada canal durante una fracción del tiempo total (intervalo de tiempo).
- Ventajas de TDM
 - El uso de la capacidad es alto.
 - Cada uno para ampliar el número de usuarios en un sistema en un coste bajo.
- Desventajas de TDM
 - La sensibilidad frente a otro problema de usuario es alta.
 - El coste inicial es alto.
 - La complejidad técnica.

Multicanalización por división de frecuencia (FDM)

- Esta técnica que consiste en:
 - Dividir mediante filtros el espectro de frecuencias del canal de transmisión
 - Desplazar la señal a transmitir dentro del margen del espectro correspondiente mediante modulaciones
 - Cada usuario tiene posesión exclusiva de su banda de frecuencias.



Multicanalización por división de frecuencia



- Características del FDM
 - El ancho de banda del medio debe ser mayor que el ancho de banda de la señal transmitida.
 - Capacidad de transmisión de varias señales a la vez.
 - La señal lógica transmitida a través del medio es analógica.
 - La señal recibida puede ser analógica o digital.
 - Para la comunicación análoga el ruido tiene menos efecto.

Multicanalización por división de frecuencia

VENTAJAS DE FDM

- El sistema de FDM apoya el flujo de dúplex total de información que es requerido por la mayor parte de la aplicación.
- El problema del ruido para la comunicación análoga tiene menos el efecto.
- Aquí el usuario puede ser añadido al sistema por simplemente añadiendo otro par de modulador de transmisor y modulador receptor.

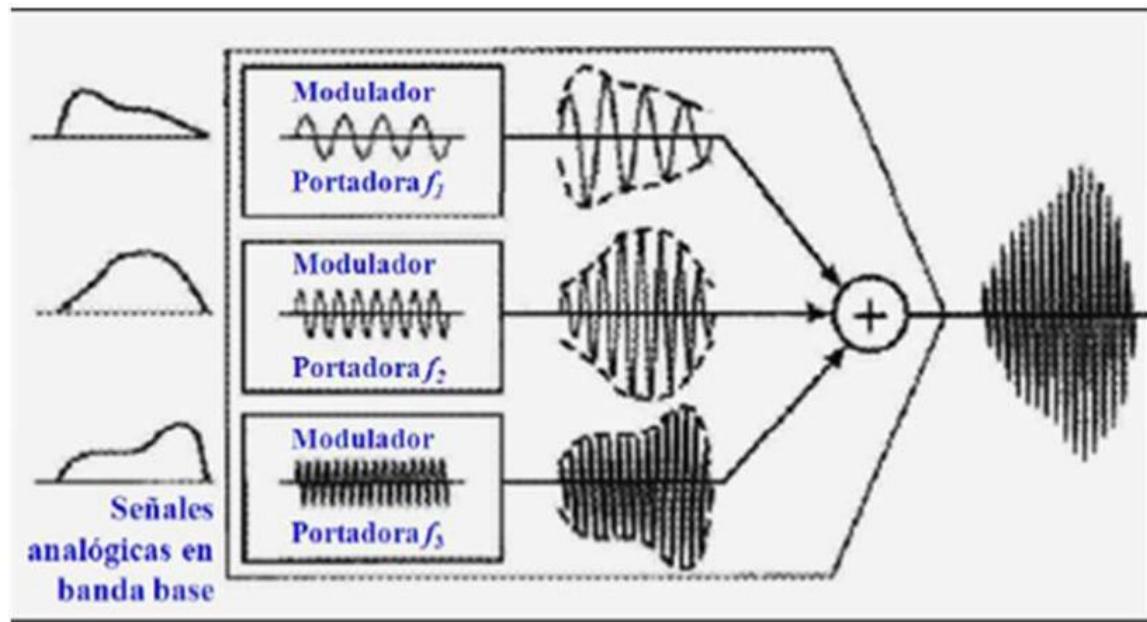
DESVENTAJAS DE FDM

- En el sistema FDM, el coste inicial es alto. Este puede incluir el cable entre los dos finales y los conectores asociados para el cable.
- En el sistema FDM, un problema para un usuario puede afectar a veces a otros.
- En el sistema FDM, cada usuario requiere una frecuencia de portador precisa.

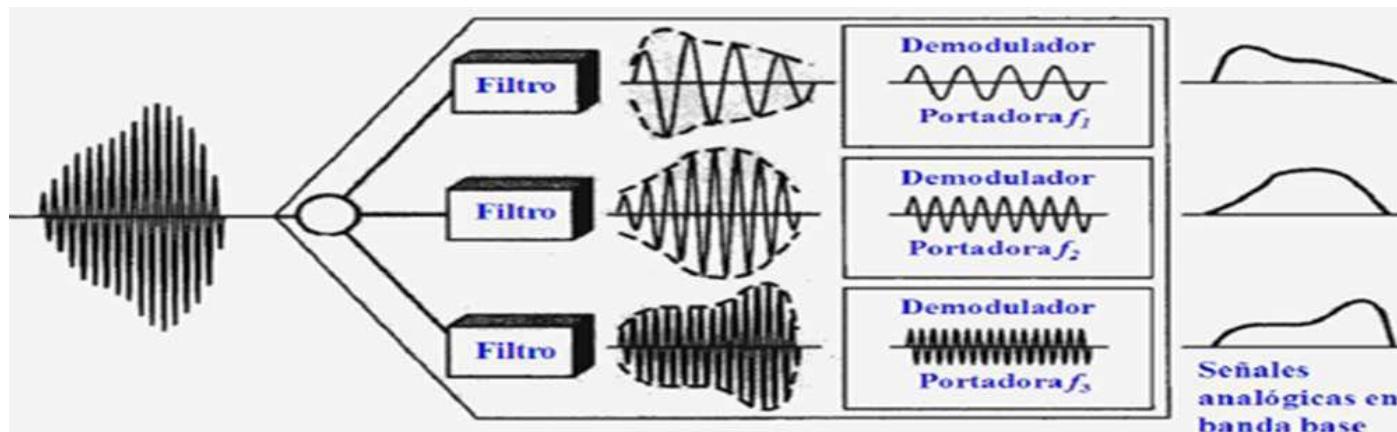
Multicanalización por división de frecuencia

Proceso de Multicanalización

- Cada fuente genera una señal con un rango de frecuencia similar.
 - Dentro del MUX, estas señales similares se modulan sobre distintas frecuencias portadoras (f_1, f_2, f_3 , etc.)
- Las señales moduladas se combinan en una única señal compuesta que se envía sobre un enlace.



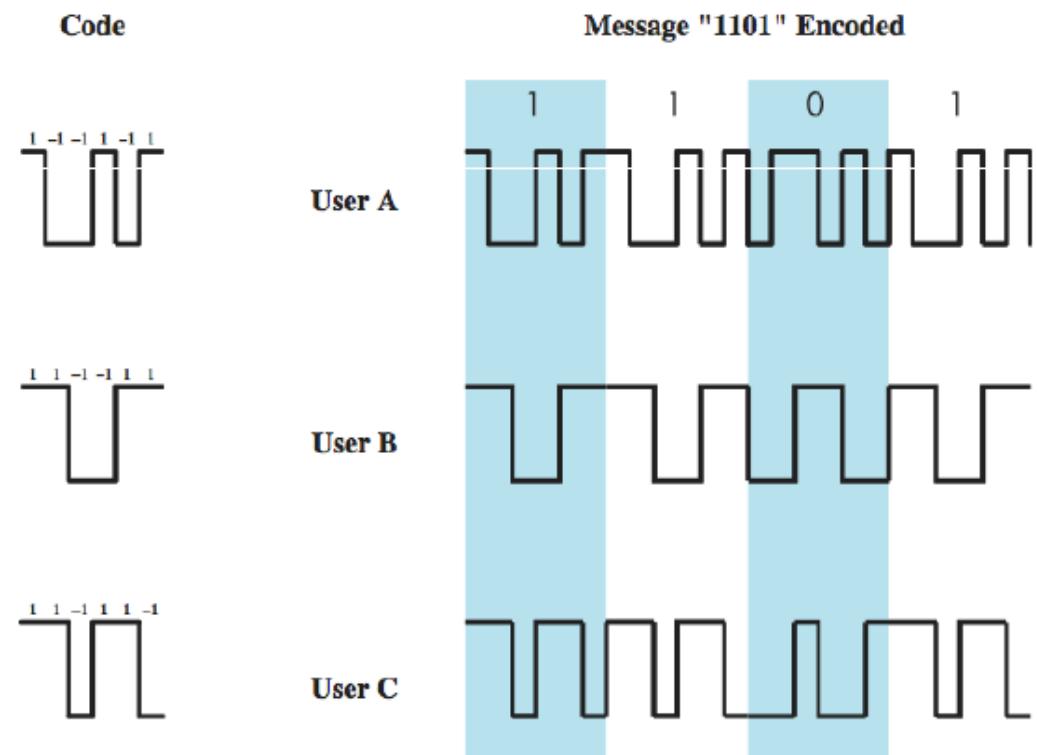
Multicanalización por división de frecuencia



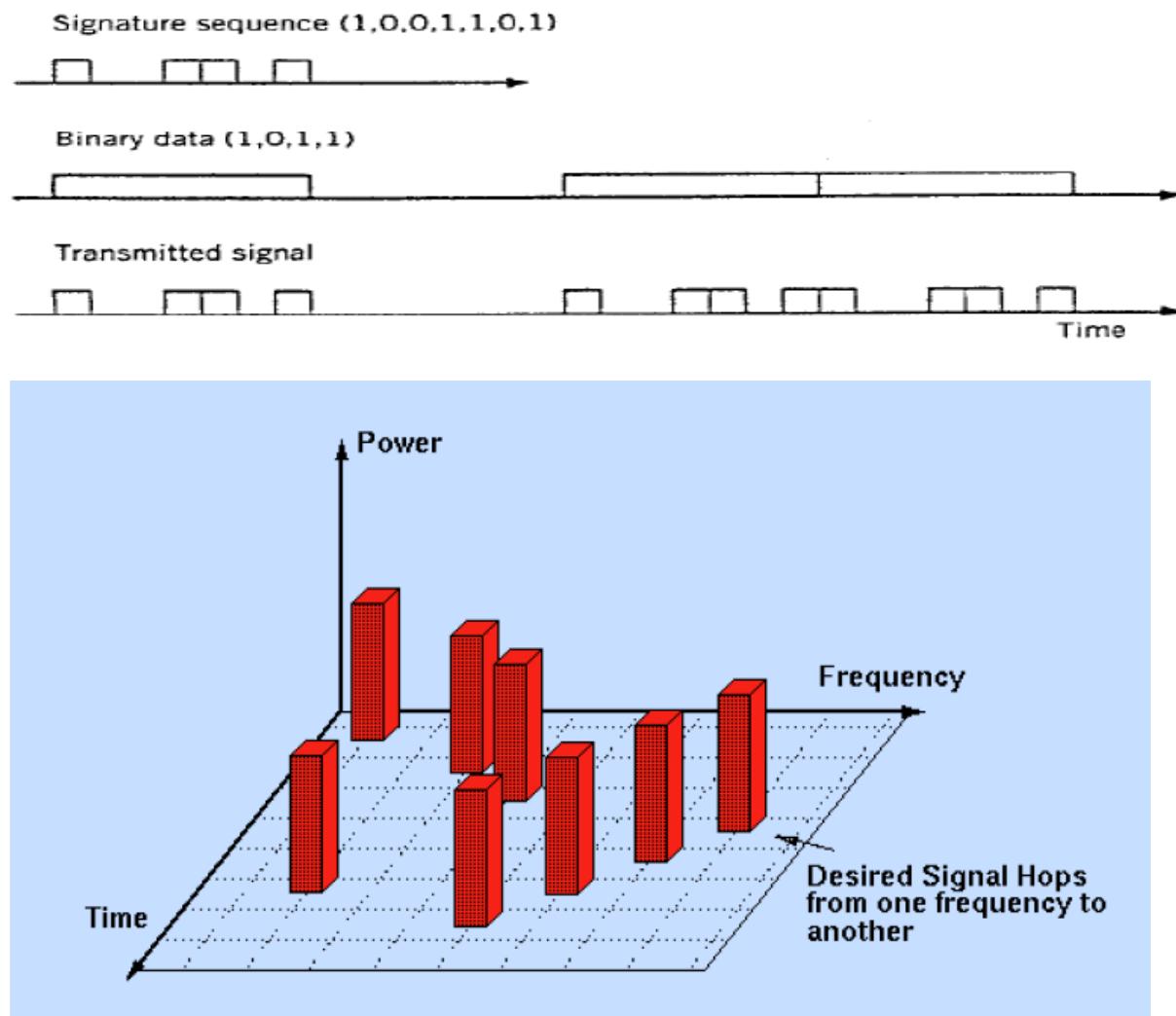
- El DEMUX usa filtros para descomponer la señal multiplexada en las señales que la constituyen.
- Las señales individuales se pasan después a un demodulador que las separa de sus portadoras y las pasa a la línea de salida

Multicanalización por división de código

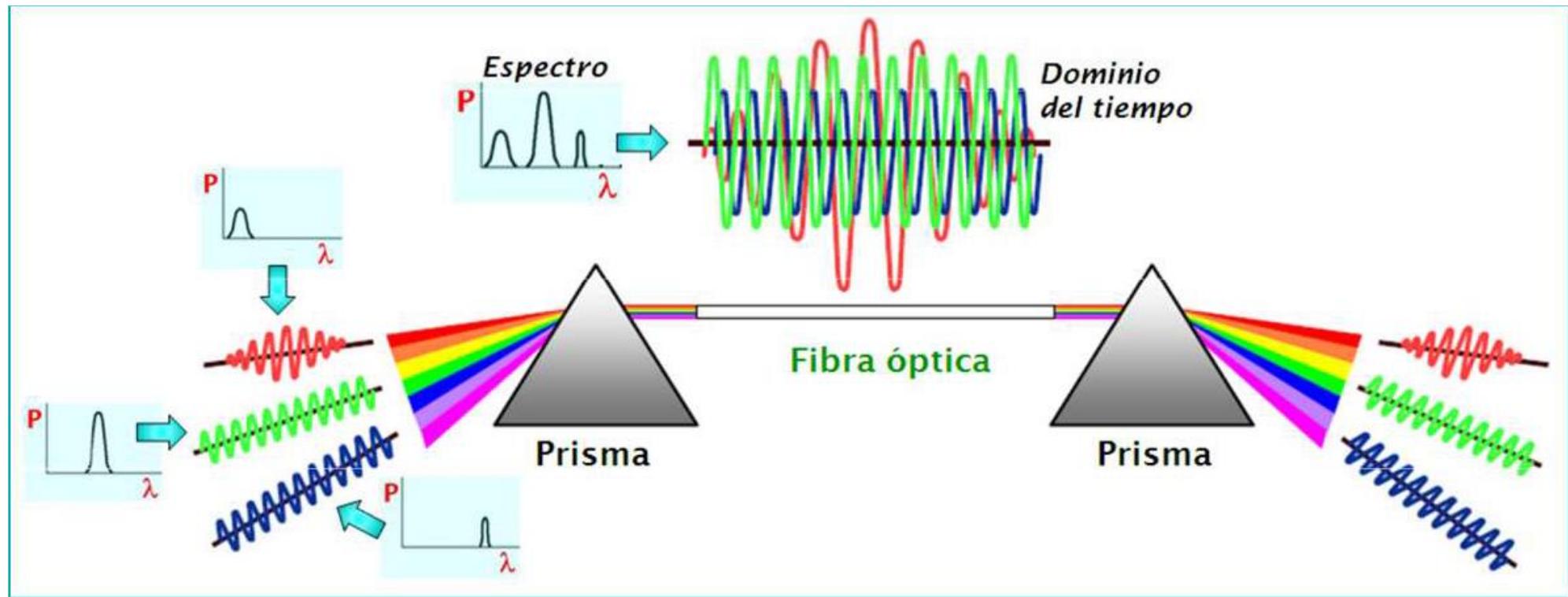
- La **multiplexación por división de código, acceso múltiple por división de código o CDMA**
- Es un término genérico para varios métodos de multiplexación o control de acceso al medio basado en la tecnología de espectro expandido.
- CDMA emplea una tecnología de espectro expandido y un esquema especial de codificación, por el que a cada transmisor se le asigna un código único, escogido de forma que sea ortogonal respecto al del resto
- En CDMA, la señal se emite con un ancho de banda mucho mayor que el precisado por los datos a transmitir
 - La división por código es una técnica de acceso múltiple de espectro expandido.



Multicanalización por división de código



- Tipos
 - Ensanchamiento espectral dado por la clave
 - Las claves las conocen el Rx y el Tx
 - Espectral
 - Esparcimiento espectral se hace con saltos en frecuencia (frequency hopping).
 - Los saltos de frecuencia están dados por el código.

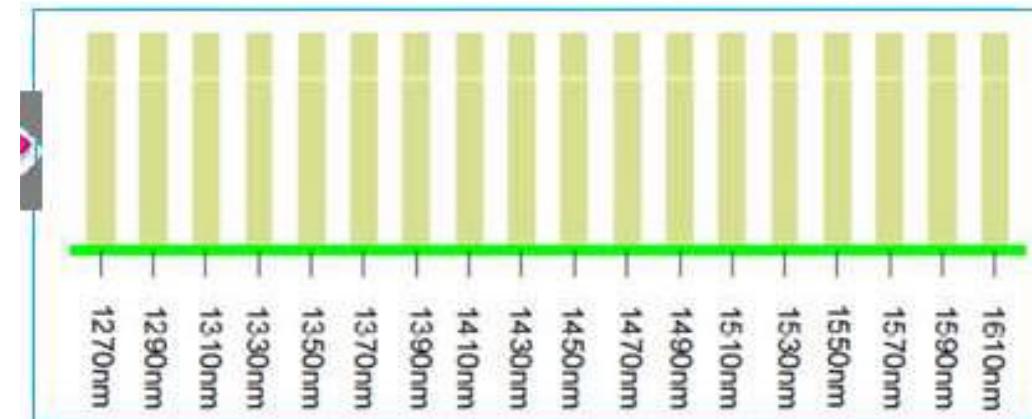


Multicanalización por división de onda

- Es una tecnología que multiplexa un número de señales portadoras ópticas en una sola fibra óptica mediante el uso de diferentes longitudes de onda de la luz láser.
- Esta técnica permite comunicaciones bidireccionales sobre una hebra de fibra, así como la multiplicación de la capacidad.
- Se diseñó para utilizar la capacidad de alta tasa de datos de la fibra.
- Conceptualmente es la misma que FDM, excepto que involucra señales luminosas de frecuencias muy altas.

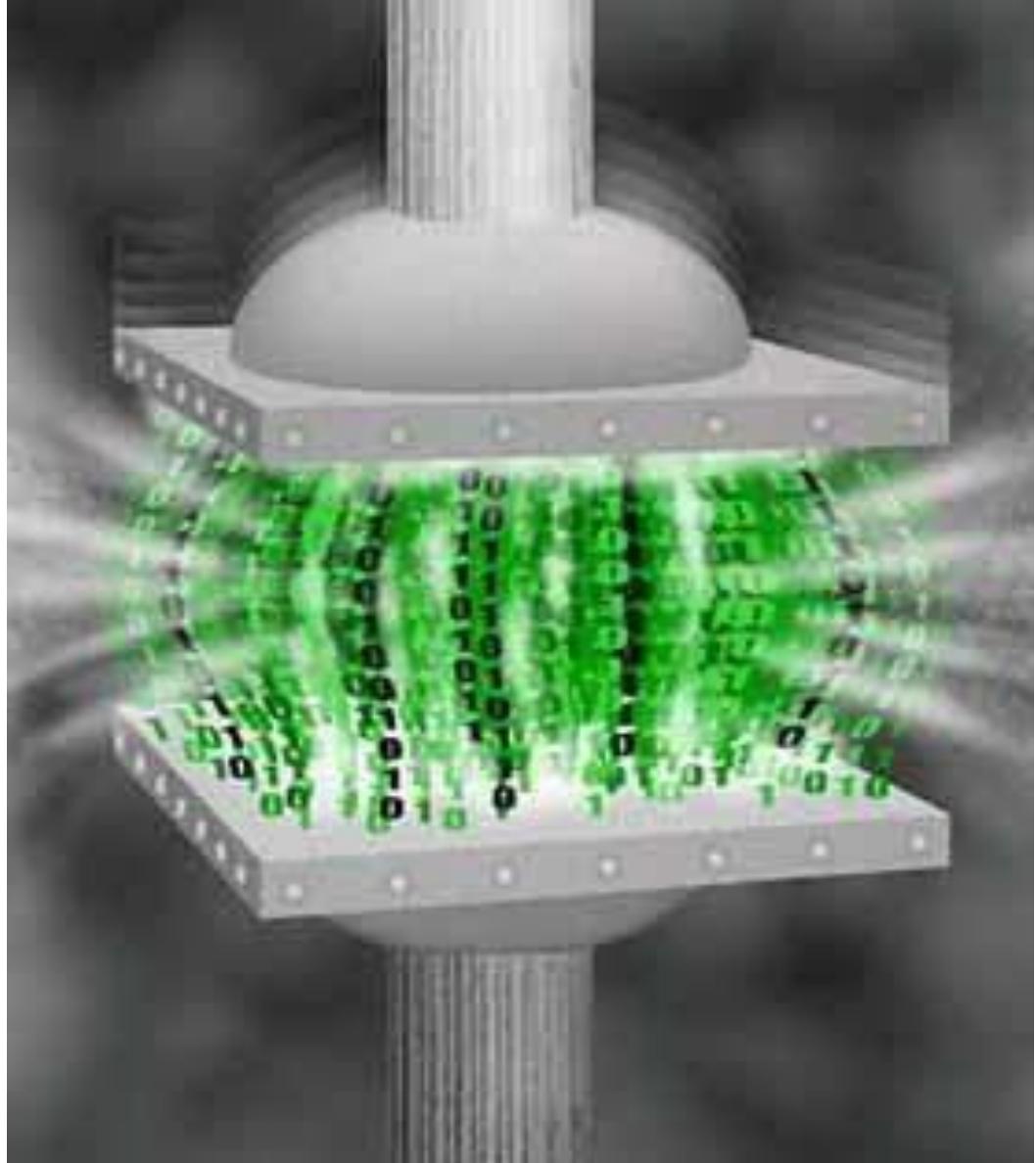
Multicanalización por división de onda

- Multiplexación por división de longitud de onda se aplica comúnmente a una portadora óptica
- Multiplexación por división de frecuencia típicamente se aplica a una portadora de radio
- Dado que la longitud de onda y la frecuencia están unidas entre sí a través de una simple relación directamente inversa, los dos términos en realidad describen el mismo concepto.
- Tipos de sistemas WDM
 - Los primeros sistemas WDM usaron 2 longitudes de onda centradas en las ventanas de 1310 nm y 1550 nm.
 - Despues fue CWDM (*Coarse WDM*) . La ITU (G.694.2) define una banda óptica de 18 l's, entre 1270 y 1610 nm, espaciadas entre ellas 20 nm.





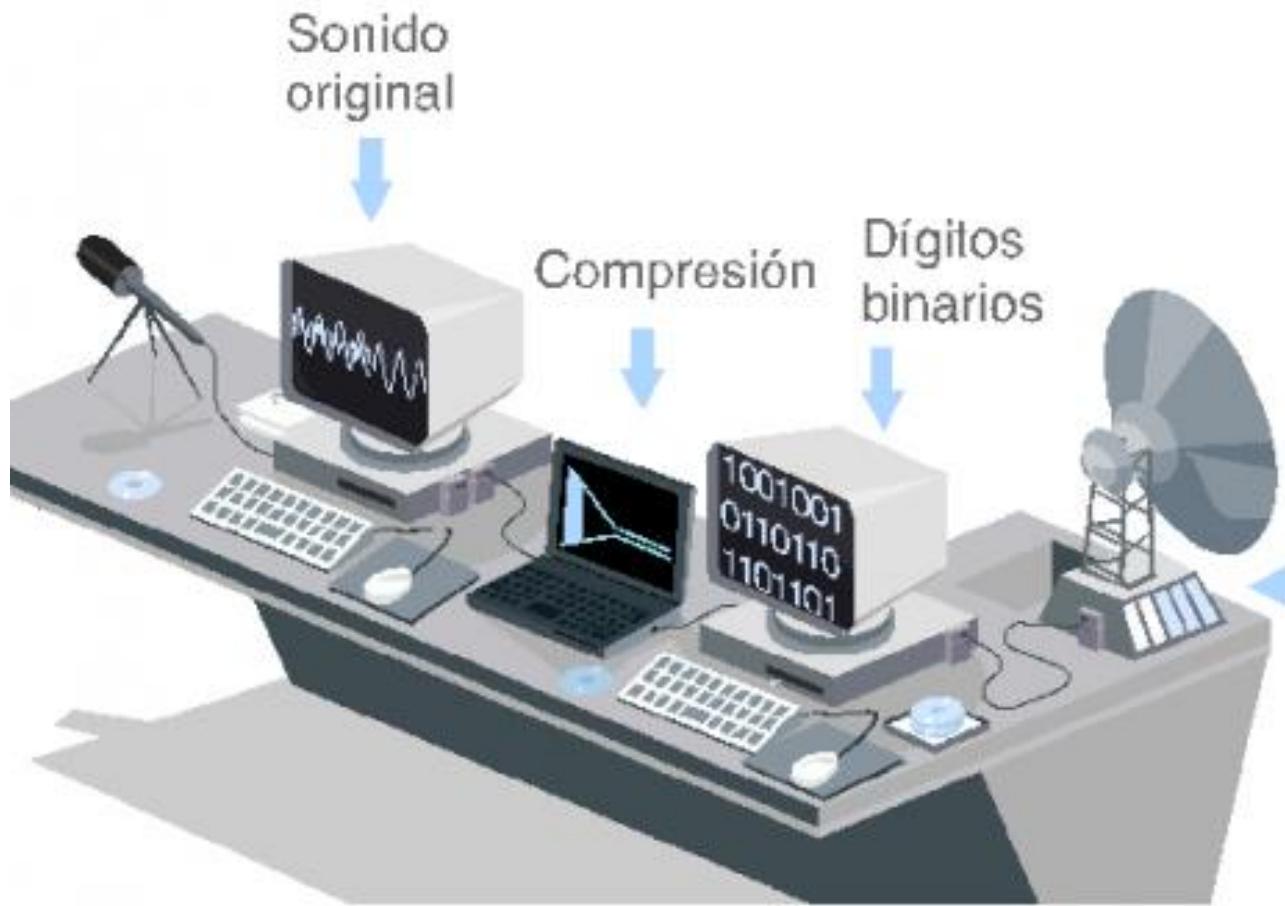
Compresión de datos



¿Qué es la compresión de datos?

- Se refiere al proceso de reducir la cantidad de datos necesarios para almacenar o transmitir información.
 - Consiste en codificar la información utilizando menos bits que la representación original.
 - Este proceso es similar a reducir un documento físico a un tamaño más pequeño y manejable sin perder su legibilidad.

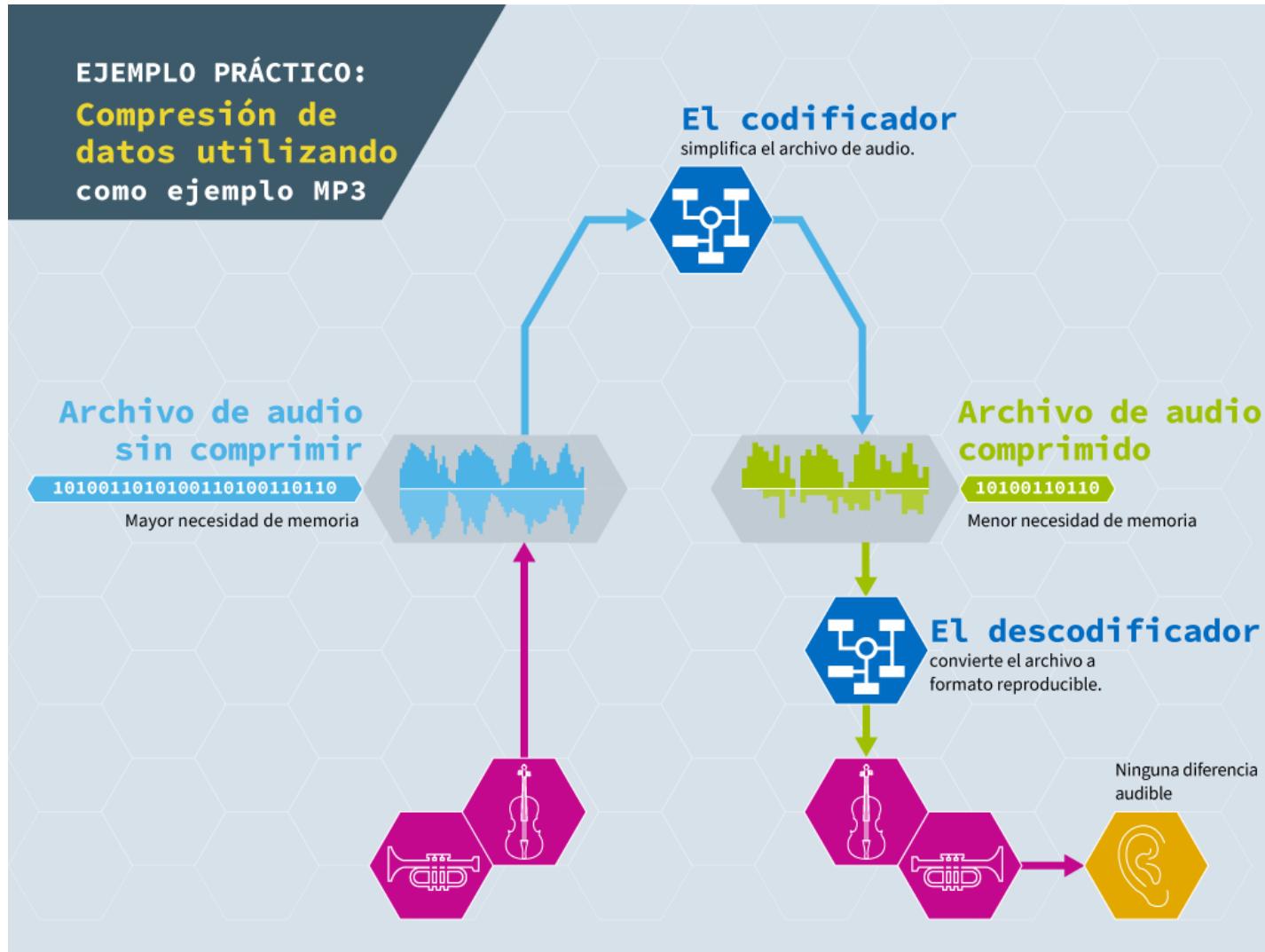
Importancia



- Los datos se han vuelto tan valiosos como cualquier otra mercancía
- La compresión de datos es clave en la era digital actual
 - Permite almacenar y transmitir datos de forma eficaz
 - Reduce
 - El espacio necesario
 - El tiempo de transferencia de datos.
 - Desempeña un papel fundamental en la gestión y utilización eficaz de este recurso.

Funcionamiento

- La compresión de datos funciona identificando y eliminando la redundancia estadística.
 - Siempre hay grado de redundancia, independientemente del tipo de datos que se procesen:
 - Texto
 - Imágenes
 - Vídeo
 - Sonido
- Eliminar redundancia da como resultado datos comprimidos que ocupan menos espacio.

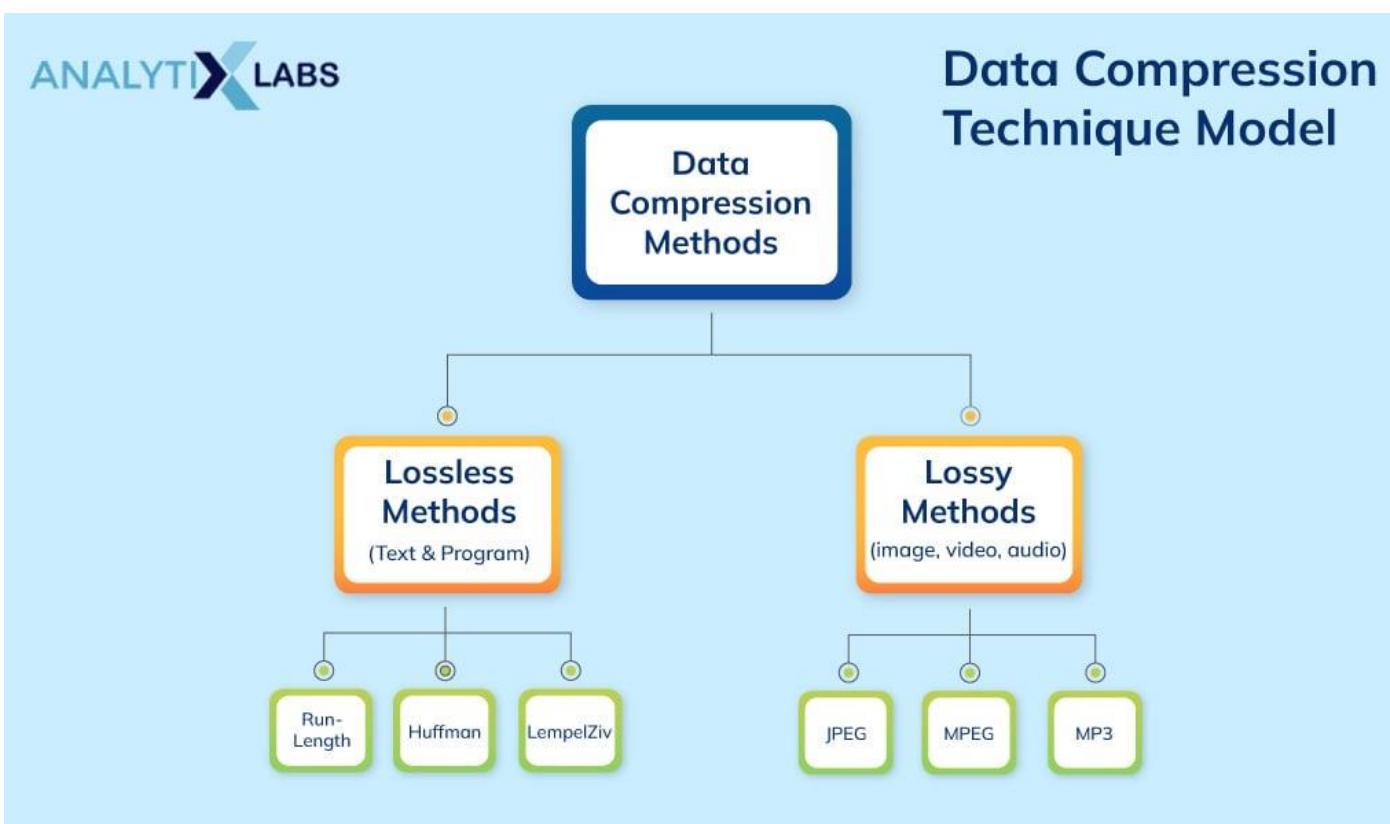


Utilización



- La compresión de datos tiene aplicación en multitud de ámbitos:
 - **Redes informáticas:** reduce la cantidad de datos transmitidos a través de las redes, aumentando así la velocidad de transmisión e incluso el ancho de banda.
 - **Almacenamiento:** al comprimir los archivos, se pueden almacenar más datos en el mismo espacio, lo cual es especialmente útil en dispositivos con capacidad de almacenamiento limitada.
 - **Servicios de streaming:** las plataformas de streaming utilizan la compresión de datos para ofrecer contenidos de forma más rápida y fluida, mejorando la experiencia del usuario.
 - **Copia de seguridad y archivo:** los datos comprimidos ocupan menos espacio, por lo que son ideales para copias de seguridad y archivos.

Compresión con o sin pérdida de datos



- Existen dos métodos principales de compresión de datos:
 - Con pérdida.
 - Reduce el tamaño del archivo eliminando la información innecesaria o menos importante.
 - Se suele utilizar en ámbitos en los que es aceptable una pérdida de calidad, como los archivos de audio y vídeo.
 - Sin pérdida
 - Método que reduce el tamaño del archivo sin pérdida de calidad.
 - Este tipo de compresión es ideal para aplicaciones en las que los datos originales deben conservarse perfectamente, como los archivos de texto.

- Diferencia entre datos e información.
 - Los datos son una colección de hechos o valores en bruto, a menudo desorganizados, y pueden significar números, texto, símbolos, etc.
 - Por otro lado, la información aporta contexto al organizar cuidadosamente los hechos.
 - Para poner esto en contexto, una imagen en blanco y negro de 4×6 pulgadas en 100 dpi (puntos por pulgada) tendrá 240.000 píxeles.
 - Cada uno de estos píxeles contiene datos en forma de un número entre 0 y 255, que representa la densidad de píxeles (0 es negro y 255 es blanco).
 - Esta imagen en su conjunto puede tener cierta información, como si fuera una foto del decimosexto presidente de los EE. UU., Abraham Lincoln.
 - Si mostramos una imagen en 50 dpi, es decir, en 60.000 píxeles, los datos necesarios para guardar la imagen se reducirán, y quizás también la calidad, pero la información permanecerá intacta.
 - Solo después de una pérdida considerable de datos, podemos perder la información.

Algoritmos con pérdida

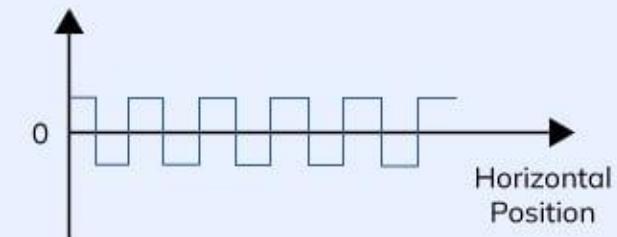
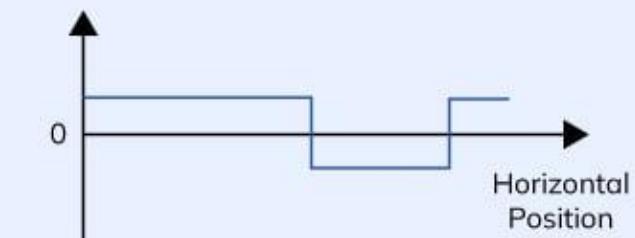
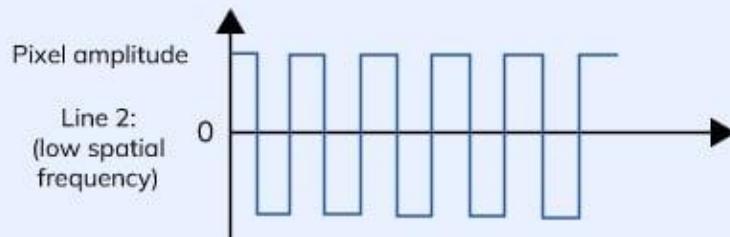
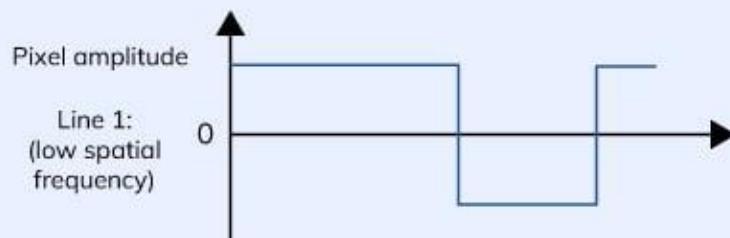
Algoritmo con pérdida



Example pixel patterns

	1	2	3	4	5	6	7	8	9	10	---
Line 1	1	2	3	4	5	6	7	8	9	10	---
2											
3											

	1	2	3	4	5	6	7	8	9	10	---
Line 1	1	2	3	4	5	6	7	8	9	10	---
2											
3											



Lossy Compression Algorithm

Algoritmo con pérdida



- Ventajas de la compresión con pérdida
 - Es relativamente rápida
 - Puede reducir drásticamente el tamaño del archivo
 - El usuario puede seleccionar el nivel de compresión.
 - Es beneficiosa para comprimir datos como:
 - Imágenes
 - Videos
 - Audio
 - Esto se debe a la limitación de nuestros ojos y oídos, ya que no pueden percibir una diferencia en la calidad de una imagen y un audio antes de cierto punto.

Algoritmos con pérdida

- Desventajas de la compresión con pérdida
 - No devolverá los mismos datos (en términos de calidad, tamaño, etc.).
 - Aun así, contendrá información similar (esto, de hecho, es útil en algunos casos, como la transmisión o la descarga de contenido de Internet).
 - La descarga y carga constante de un archivo puede comprimirlo y, en consecuencia, distorsionarlo más allá del punto de reconocimiento
 - Pérdida permanente de información.
 - Si el usuario utiliza un nivel de compresión severo, es posible que el archivo de salida no se parezca en nada al archivo de entrada original.

Example: Decrease Resolution

Represent the same image with much fewer pixels



637 x 637 pixels



90 x 90 pixels

Algoritmos con pérdida

- Modelos de técnicas de compresión con pérdida:
 - Los modelos más comunes basados en la técnica con pérdida son:
 - Codificación por transformada
 - Transformada discreta del coseno
 - Transformada discreta de wavelet
 - Compresión fractal

Algoritmo con pérdida

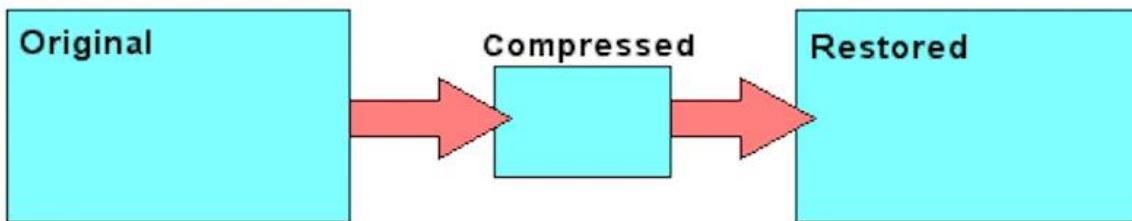
- La codificación por transformación es un tipo de compresión de datos para datos "naturales" como señales de audio o imágenes fotográficas.
- Normalmente, la transformación conlleva pérdida de información, resultando una copia de menor calidad que la entrada original.
- En la codificación por transformación, el conocimiento de la aplicación se utiliza para elegir la información a descartar para, de esa forma, disminuir su ancho de banda.
- La información restante se puede comprimir mediante varios métodos.
- Cuando se descodifica la salida, el resultado puede no ser idéntico a la entrada original, pero se espera que sea lo suficientemente parecido para los propósitos de la aplicación.

Algoritmos con pérdida

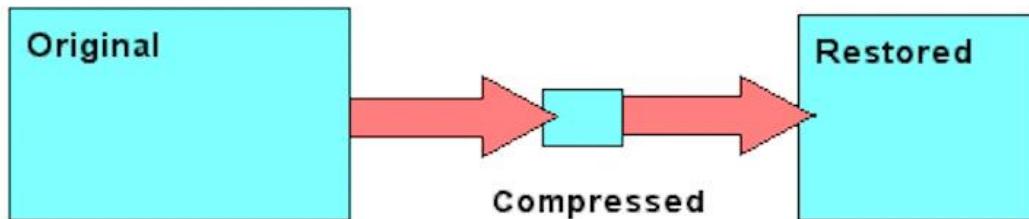
- Compresión de vídeo con pérdida
 - Flash (también admite sprites JPEG)
 - H.261
 - H.263
 - H.264/MPEG-4 AVC
 - MNG (admite sprites JPEG)
 - Motion JPEG
 - MPEG-1
 - MPEG-2
 - MPEG-4
 - OGG
 - Theora
 - códec para video Sorenson
 - VC-1
 - MP4
- Compresión de imagen con pérdida
 - Compresión fractal
 - JPEG
 - Compresión Wavelet
- Compresión de audio con pérdida
 - AAC
 - ADPCM
 - ATRAC
 - Dolby AC-3
 - DTS
 - MP2
 - MP3
 - Musepack
 - OGG
 - Vorbis
 - WMA
 - Opus (CELT)

Algoritmo sin pérdida

LOSSLESS

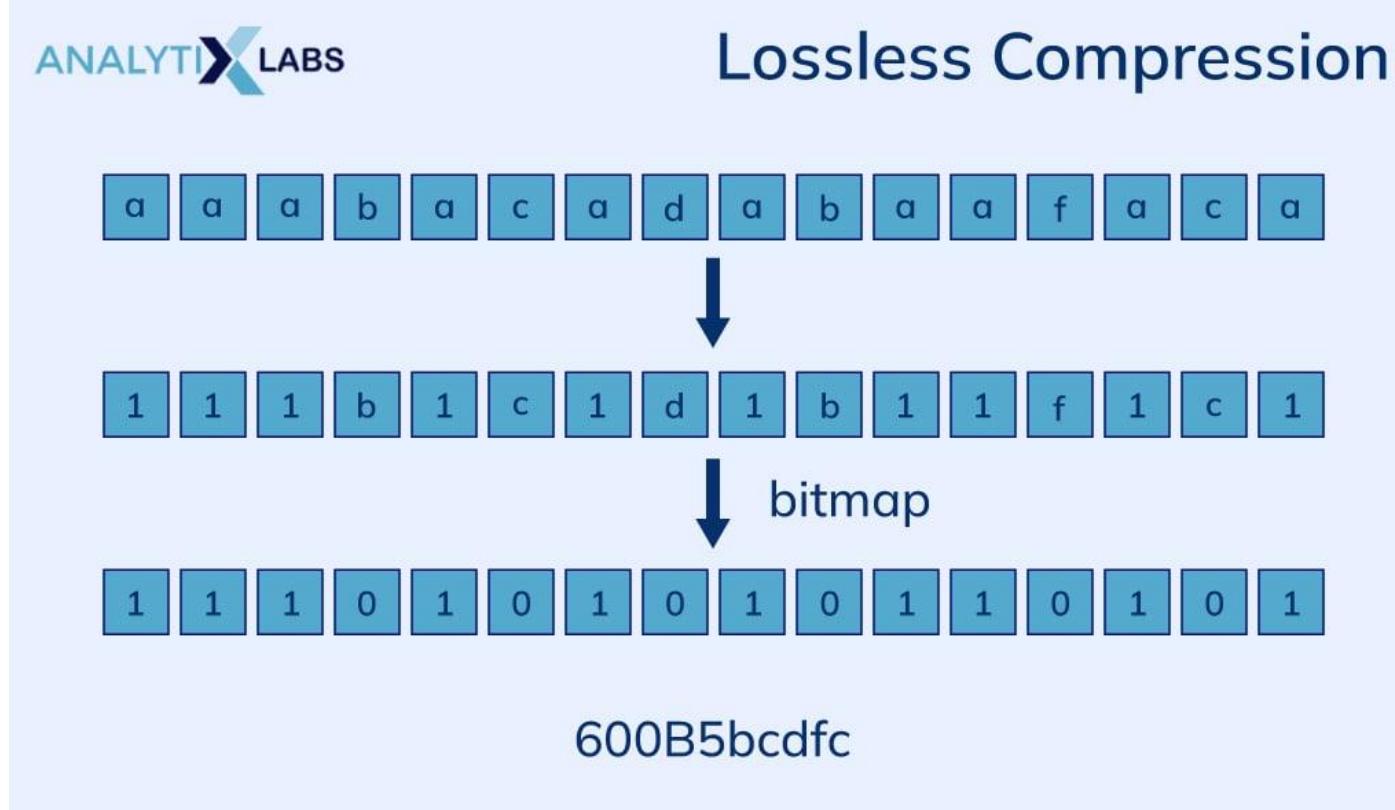


LOSSY



- La compresión sin pérdida no elimina ningún dato
 - Lo transforma para reducir su tamaño.
 - Entendiendo el concepto
 - Hay un fragmento de texto en el que la palabra "porque" se repite con bastante frecuencia.
 - El término está compuesto por siete letras
 - Utilizar una versión abreviada del mismo como "bcz", se puede transformar el texto.
 - Esta información de reemplazar "porque" por "bcz" se puede almacenar en un diccionario para su uso posterior (durante la descompresión).

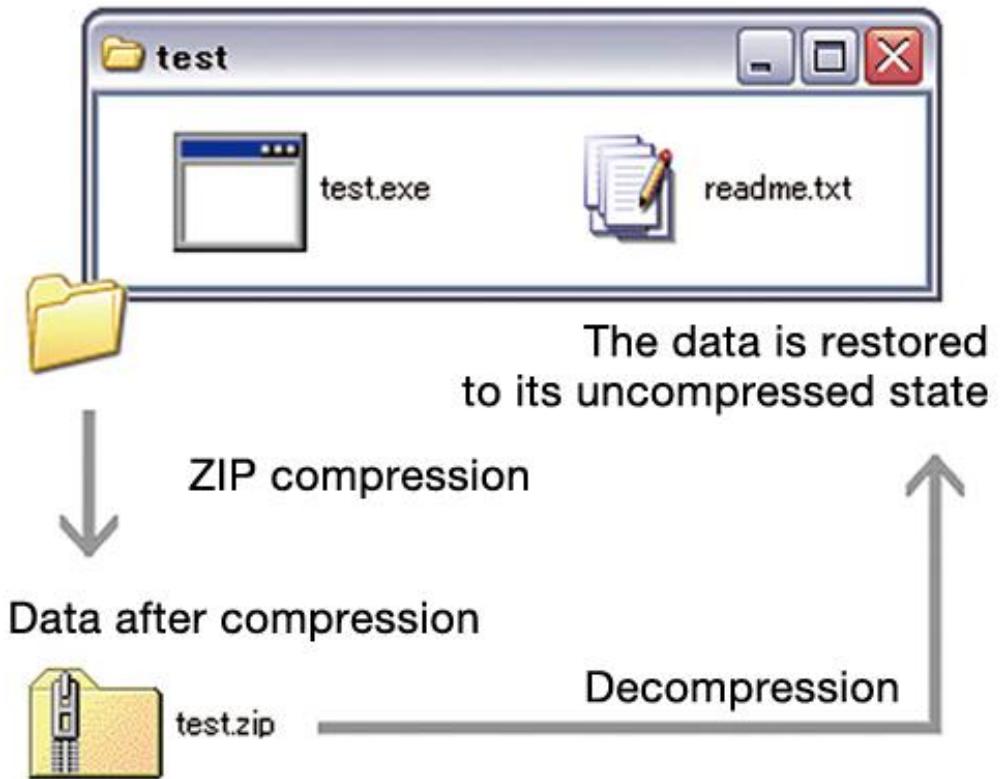
Algoritmo sin pérdida



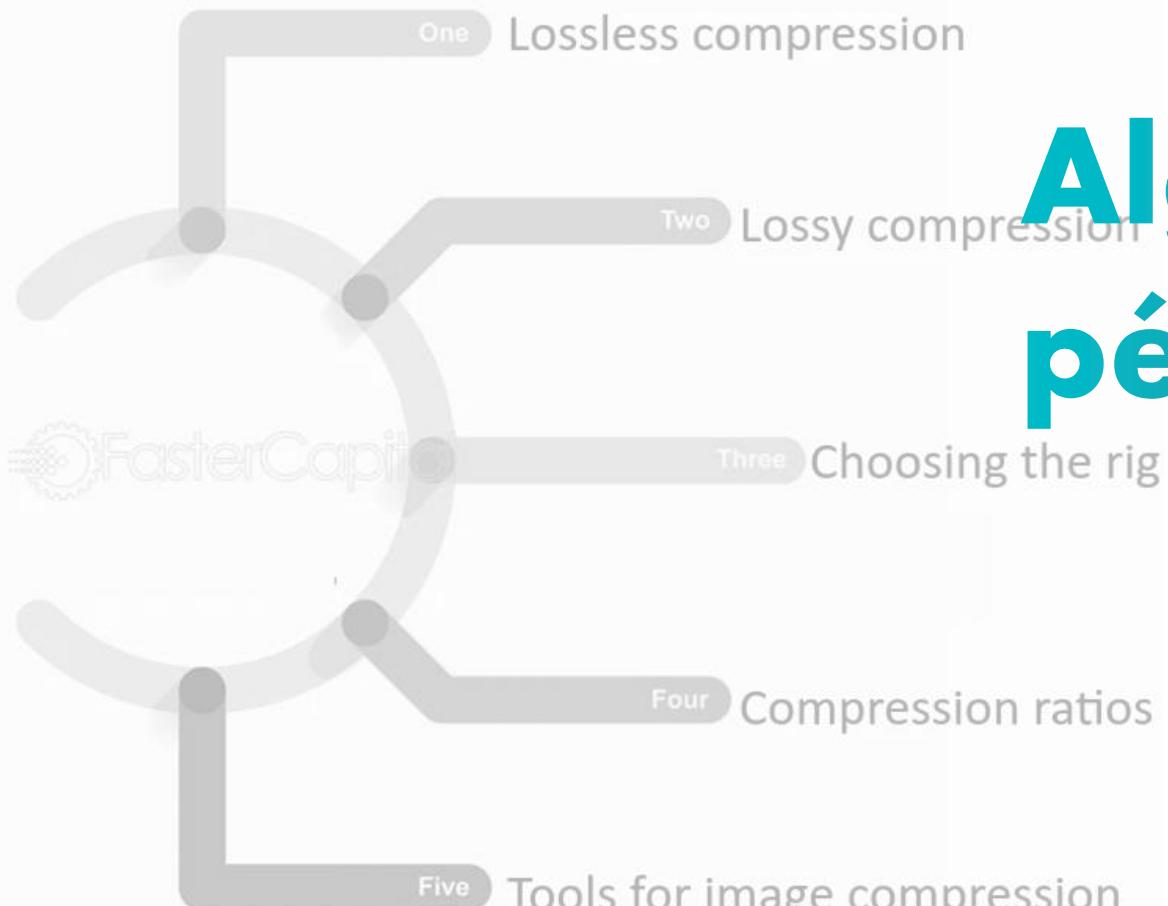
- Metodología:
 - La compresión con pérdida elimina fragmentos de datos redundantes o imperceptibles para reducir el tamaño
 - La compresión sin pérdida los transforma codificándolos mediante alguna fórmula o lógica. Así es como funciona la compresión sin pérdida.

Algoritmo sin pérdida

- Ventajas:
 - Existen tipos de datos en los que la compresión con pérdida no es viable.
 - Una hoja de cálculo
 - Un software
 - Un programa o cualquier dato compuesto por texto o números
 - La compresión con pérdida no puede funcionar
 - Todos los números pueden ser esenciales y no pueden considerarse redundantes
 - Cualquier reducción provocará inmediatamente la pérdida de información
 - La compresión sin pérdida se vuelve crucial,
 - Tras la descompresión, el archivo se puede restaurar a su estado original sin perder ningún dato.



Balancing File Size and Quality

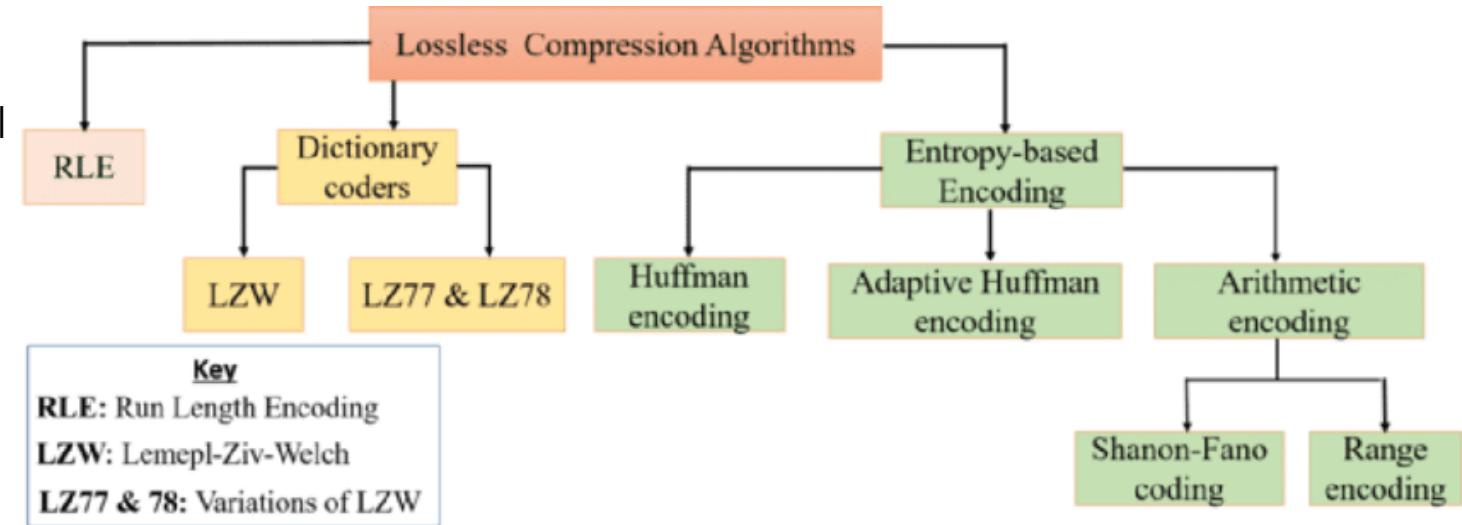


Algoritmos sin pérdida

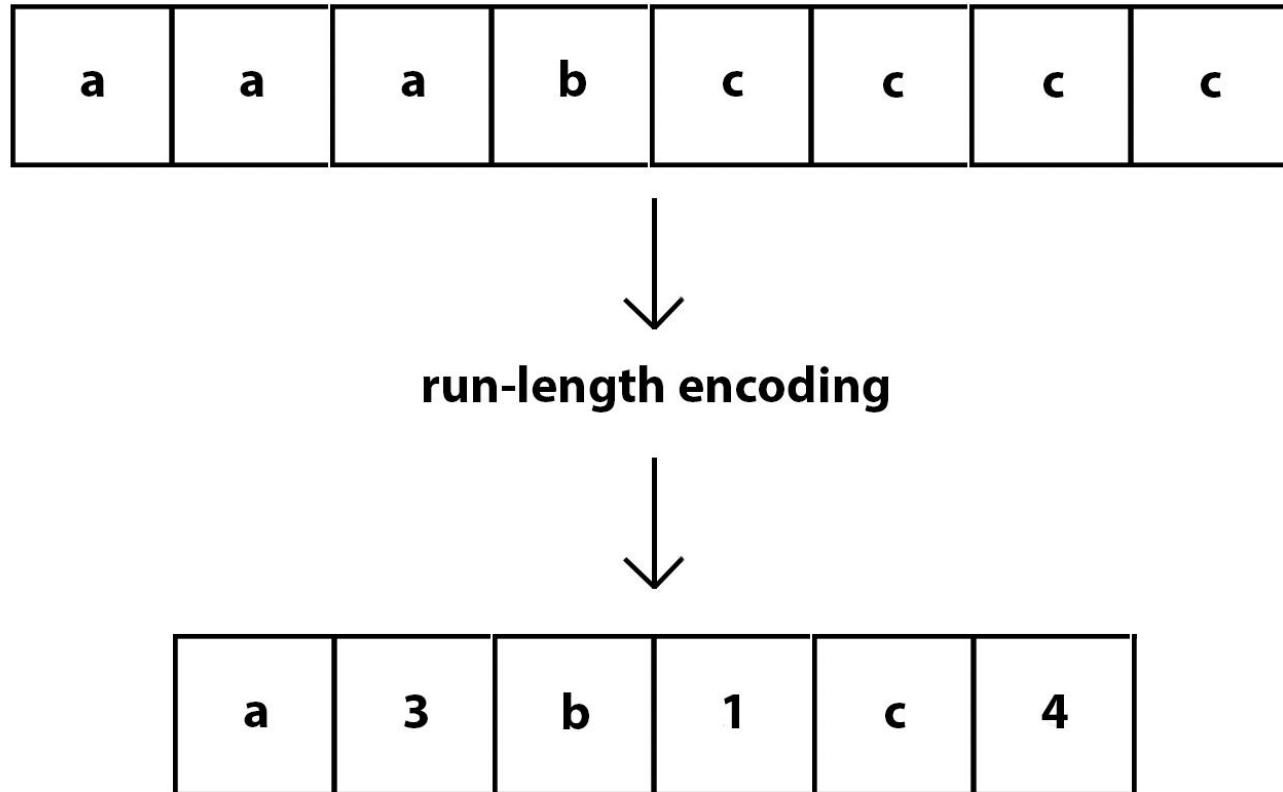
- Desventaja:
 - La compresión de datos tiene un límite.
 - Si los datos ya están comprimidos, volver a comprimirlos reducirá poco o nada su tamaño.
 - Es menos eficaz con archivos de mayor tamaño.

- Modelos de técnicas de compresión sin pérdida
 - Los modelos más comunes basados en la técnica sin pérdida son:
 - RLE (codificación de longitud de ejecución)
 - Codificador de diccionario (LZ77, LZ78, LZR, LZW, LZSS, LZMA, LZMA2)
 - Predicción por coincidencia parcial (PPM)
 - Deflate Mezcla de contenido
 - Codificación Huffman
 - Codificación Huffman adaptativa
 - Codificación Shannon Fano
 - Codificación aritmética
 - Codificación Lempel Ziv Welch
 - Zstandard
 - Bzip2 (Burrows y Wheeler)

Algoritmos sin pérdida



Algoritmo sin pérdida

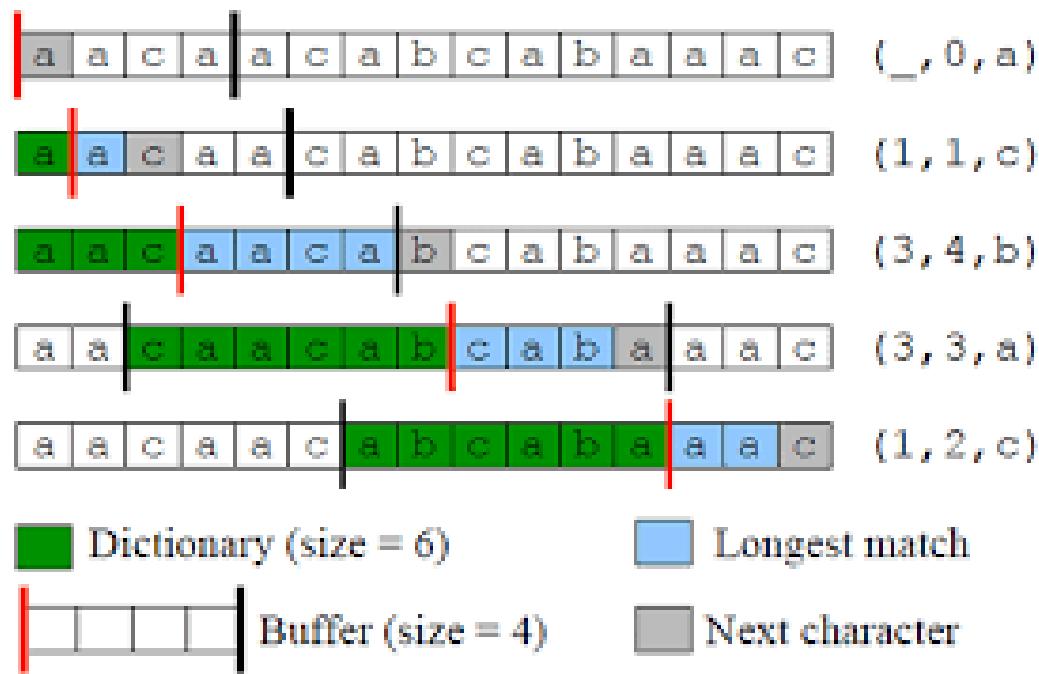


- RLE (codificación de longitud de ejecución)
 - El algoritmo Run-Length Encoding, o RLE, comprime datos que contienen una gran cantidad de repeticiones consecutivas de un mismo valor o símbolo.
 - En lugar de almacenar cada valor o símbolo por separado, la codificación RLE almacena el valor o símbolo repetido junto con la longitud de la repetición.



```
def rle_encode(data: str) -> str:  
    # Función auxiliar recursiva que toma la cadena de entrada y un índice  
    def rle_helper(data: str, i: int) -> str:  
        # Verificar si ya se ha llegado al final de la cadena de entrada  
        if i >= len(data):  
            return ""  
        # Contar la cantidad de repeticiones consecutivas del carácter en la posición actual i  
        def count_sequence(data, i):  
            if i + 1 >= len(data) or data[i] != data[i + 1]:  
                return 1  
            count = count_sequence(data, i + 1)  
            return count + 1  
        count = count_sequence(data, i)  
        # Si la cantidad de repeticiones es 1, simplemente agregar el carácter a la cadena comprimida  
        if count == 1:  
            return data[i] + rle_helper(data, i + 1)  
        # Si la cantidad de repeticiones es mayor que 1, agregar la cantidad y el carácter a la cadena comprimida  
        else:
```

Algoritmos sin pérdida



- LZ77
 - Es un compresor basado en algoritmo sin pérdida
 - Es un tipo de codificador diccionario en el cual existen los literales, banderas y palabras claves
 - Se recorre la cadena
 - Si se encuentra con un literal lo deja totalmente igual,
 - Si encuentra una bandera especifica si lo que sigue es
 - Un literal
 - Un comprimido (que es una especie de palabra clave)
 - Se lleva a una posición en un diccionario que arroja que bytes continúan.

Algoritmo sin pérdida

LZ77, ejemplo con abracadabarray

I	2	3	4	5	6	7	8	I	2	3	4	5	6	Output
								a	b	r	a	c	a	<0,0,a>
								a	b	r	a	c	a	d <0,0,b>
								a	b	r	a	c	a	d <0,0,r>
								a	b	r	a	c	a	b <3,1,c>
								a	b	r	a	c	a	b <2,1,d>
a	b	r	a	c	a	d	a	b	r	a	r	r	r <7,4,r>	
c	a	d	a	b	r	a	r	r	a	y				<3,2,y>

← →

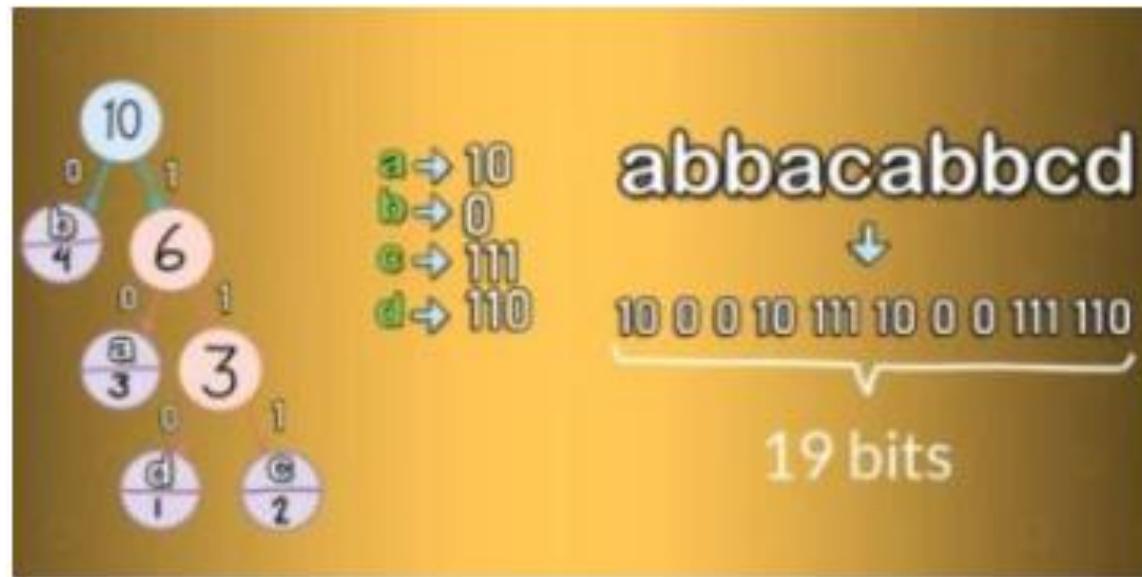
Search Buffer Look ahead buffer Buffer

Activate Windows
Go to Settings to activate Window

Algoritmo sin Pérdida

```
def compress(self, text: str) -> list[Token]:  
    """  
    Args:  
        text: string to be compressed  
  
    Returns:  
        output: the compressed text as a list of Tokens  
    """  
  
    output = []  
    search_buffer = ""  
  
    # while there are still characters in text to compress  
    while text:  
        # find the next encoding phrase  
        # - triplet with offset, length, indicator (the next encoding character)  
        token = self._find_encoding_token(text, search_buffer)  
  
        # update the search buffer:  
        # - add new characters from text into it  
        # - check if size exceed the max search buffer size, if so, drop the  
        #   oldest elements  
        search_buffer += text[: token.length + 1]  
        if len(search_buffer) > self.search_buffer_size:  
            search_buffer = search_buffer[-self.search_buffer_size :]  
        # update the text  
        text = text[token.length + 1 :]  
  
        # append the token to output  
        output.append(token)  
  
    return output
```

Algoritmo sin pérdida



Ejemplo:
ilaaaseeaalllssallsas

- Codificación de Huffman
 - Es un procedimiento que permite asignar a los diferentes símbolos a comprimir, un código binario.
 - Este algoritmo crea un árbol de nodos
 - Primero se debe establecer un orden prioritario
 - El más importante es el símbolo que aparece con menor frecuencia en la cadena
 - Luego, se eliminan los dos símbolos más prioritarios
 - Construyendo así un nuevo "padre" que es el resultado de la suma de las frecuencias eliminadas
 - Se ubica nuevamente en la cola de prioridad
 - Iterativamente se realiza este proceso hasta que solo quede un elemento
 - Así queda construido el árbol.
 - Para asignar el código binario se contarán los pasos efectuados para llegar a cada símbolo del árbol
 - Movimiento a la izquierda=0, a la derecha=1
 - Obteniendo el valor de cada uno
 - Por último, reemplazándolos en la cadena.

```
def Huffman_Encoding(data):
    symbol_with_probs = Calculate_Probability(data)
    symbols = symbol_with_probs.keys()
    probabilities = symbol_with_probs.values()
    print("symbols: ", symbols)
    print("probabilities: ", probabilities)

    nodes = []

    # converting symbols and probabilities into huffman tree nodes
    for symbol in symbols:
        nodes.append(Node(symbol_with_probs.get(symbol), symbol))

    while len(nodes) > 1:
        # sort all the nodes in ascending order based on their probability
        nodes = sorted(nodes, key=lambda x: x.prob)
        # for node in nodes:
        #     print(node.symbol, node.prob)

        # pick 2 smallest nodes
        right = nodes[0]
        left = nodes[1]

        left.code = 0
        right.code = 1

        # combine the 2 smallest nodes to create new node
        newNode = Node(left.prob+right.prob, left.symbol+right.symbol, left, right)

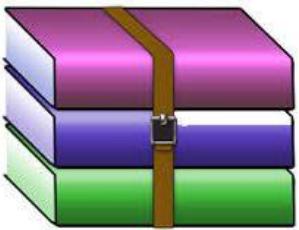
        nodes.remove(left)
        nodes.remove(right)
        nodes.append(newNode)

    huffman_encoding = Calculate_Codes(nodes[0])
    print(huffman_encoding)
    Total_Gain(data, huffman_encoding)
    encoded_output = Output_Encoded(data,huffman_encoding)
    print("Encoded output:", encoded_output)
    return encoded_output, nodes[0]
```

ARCHIVOS ZIP O RAR



archivo tipo zip



archivo tipo rar

compresión sin perdida

Algoritmos sin pérdida

- Wavelets
- Zip
- rar
- CAB
- LHA
- DGCA
- GCA

Modelos basados en redes neuronales

- Algunos modelos basados en redes neuronales también se utilizan para la compresión, como:
 - Compresión basada en perceptrón multicapa (MLP) (utilizada para la compresión de imágenes)
 - Compresión basada en red neuronal convolucional (CNN) como Deep Coder (utilizada para la compresión de video)
 - Compresión basada en red generativa (GAN) (utilizada para la compresión en tiempo real)

Programación con Sockets

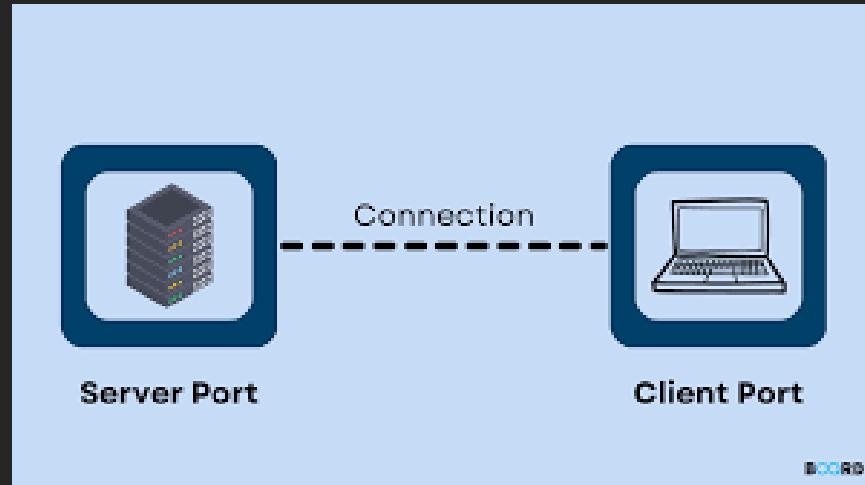
```
    _for object to mirror
    mirror_mod.mirror_object

    operation = "MIRROR_X":
    mirror_mod.use_x = True
    mirror_mod.use_y = False
    mirror_mod.use_z = False
    mirror_mod.operation = "MIRROR_Y":
    mirror_mod.use_x = False
    mirror_mod.use_y = True
    mirror_mod.use_z = False
    mirror_mod.operation = "MIRROR_Z":
    mirror_mod.use_x = False
    mirror_mod.use_y = False
    mirror_mod.use_z = True

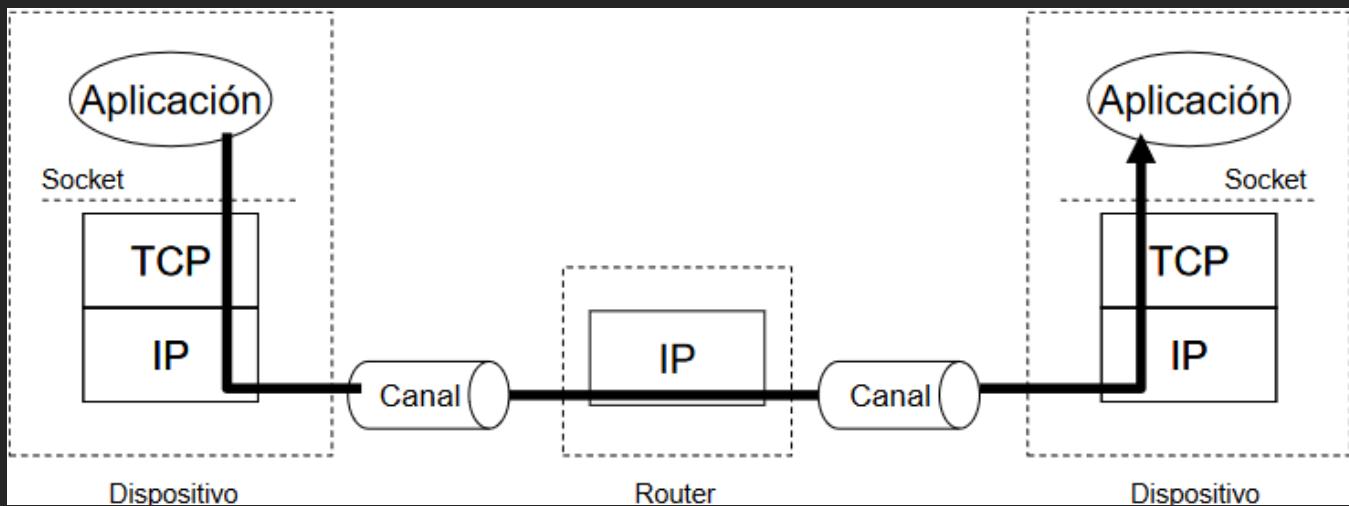
    selection at the end -add
    mirror_ob.select= 1
    mirror_ob.select=1
    context.scene.objects.active = one
    ("Selected" + str(modifier))
    mirror_ob.select = 0
    bpy.context.selected_objects = []
    data.objects[one.name].select = 1
    print("please select exactly one object")
    - OPERATOR CLASSES -
types.Operator:
    X mirror to the selected
    object.mirror_mirror_x"
    for X"
```

Definición

- Los sockets son una de las herramientas que ofrecen los Sistemas Operativos para la comunicación entre diferentes procesos.
- La particularidad que tienen frente a otros mecanismos de comunicación entre procesos (IPC – Inter-Process Communication) es que:
Posibilitan la comunicación aun cuando ambos procesos estén corriendo en distintos sistemas unidos mediante una red.
- De hecho, el API de sockets es la base de cualquier aplicación que funcione en red
Ofrece una librería de funciones básicas que el programador puede usar para desarrollar aplicaciones en red



Sockets TCP/IP



- Permiten la comunicación de dos procesos que estén conectados a través de una red TCP/IP.
*Cada máquina está identificada por medio de su dirección IP
Cada máquina puede estar ejecutando múltiples procesos simultáneamente.*
 - Cada uno de estos procesos se asocia con un número de puerto
Un socket se identifica únicamente por la dupla dirección IP + número de puerto.
- Una comunicación entre dos procesos se identifica mediante:
La asociación de los sockets que estos emplean para enviar y recibir información hacia y desde la red
identificador de socket origen + identificador de socket destino.

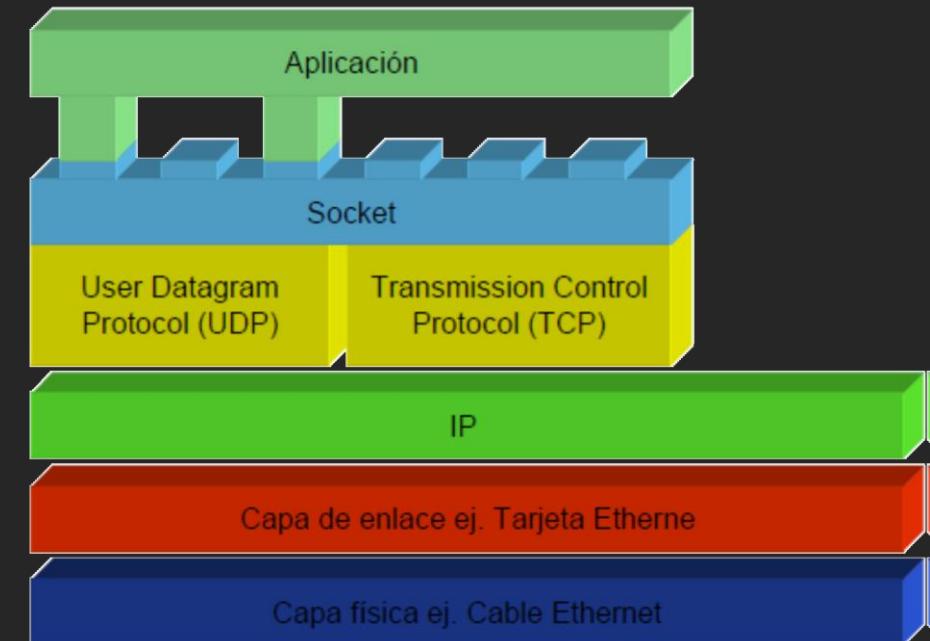
Protocolos y sockets

- Un socket es una abstracción a través de la cual una aplicación puede enviar y recibir información

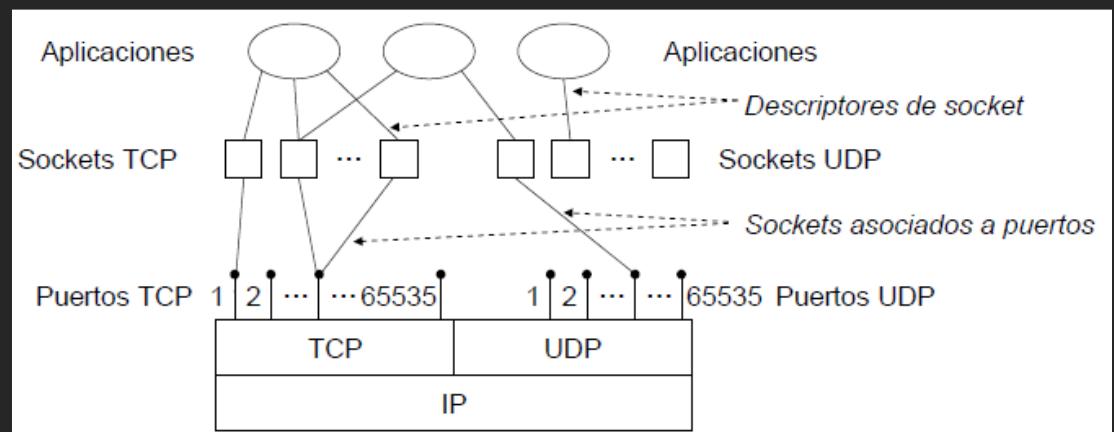
Muy similar a como se escribe y lee de un archivo.

La información que una aplicación envía por su socket origen puede ser recibida por otra aplicación en el socket destino y viceversa.

- Existen diferentes tipos de sockets dependiendo de la pila de protocolos sobre la que se cree dicho socket.



Sockets, Protocolos y Puertos



- Relación lógica entre aplicaciones, sockets, protocolos y puertos en un dispositivo.

Aspectos por destacar en esta relación

- Primero, un programa puede usar más de un socket al mismo tiempo
- Segundo, diferentes programas pueden usar el mismo socket al mismo tiempo,

Cada socket tiene asociado un puerto TCP o UDP, según sea el caso.

Cuando se recibe un paquete dirigido a dicho puerto, este paquete se pasa a la aplicación correspondiente.

Dominios de comunicación

- Los sockets se crean dentro de lo que se denomina un dominio de comunicación que define cual será la pila de protocolos que se usará en la comunicación

Dominio	Propósito
PF_UNIX, PF_LOCAL	Procesos que se comunican en un mismo sistema UNIX
PF_INET	Procesos que se comunican usando una red IPv4
PF_INET6	Procesos que se comunican usando una red IPv6

PF_IPX	Procesos que se comunican usando una red Novell
PF_NETLINK	Comunicación con procesos del kernel
PF_X25	ITU-T X.25 / ISO-8208 protocol
PF_AX25	Amateur radio AX.25
PF_ATMPVC	Acceso a PVCs ATM
PF_APPLETALK	Appletalk
PF_PACKET	Interfaz con paquetes de bajo nivel

Tipos de Sockets

- En el dominio PF_INET se definen los siguientes tipos de sockets:

Sockets Stream

- hace uso del protocolo TCP que provee un flujo de datos bidireccional, orientado a conexión, secuenciado, sin duplicación de paquetes y libre de errores

Sockets Datagram

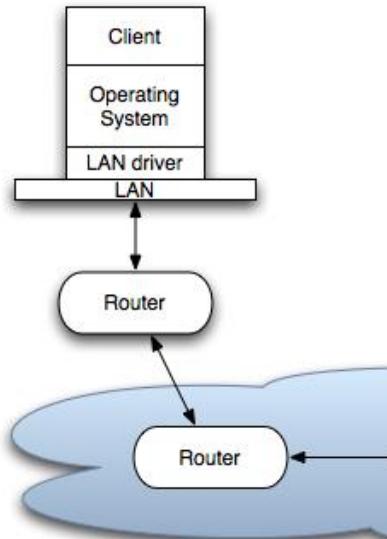
- hacen uso del protocolo UDP, el cual provee un flujo de datos bidireccional, no orientado a conexión, en el cual los paquetes pueden llegar fuera de secuencia, puede haber pérdidas de paquetes o pueden llegar con errores

Sockets Raw

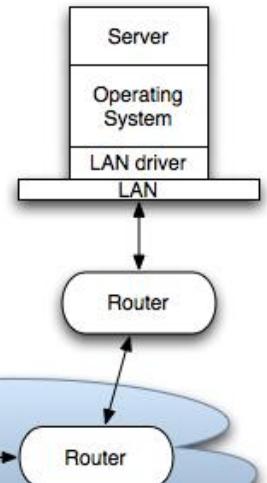
- Permiten un acceso a más bajo nivel, pudiendo acceder directamente al protocolo IP del nivel de Red. Su uso está mucho más limitado ya que está pensado principalmente para desarrollar nuevos protocolos de comunicación, o para obviar los protocolos del nivel de transporte.

Modelo Cliente Servidor

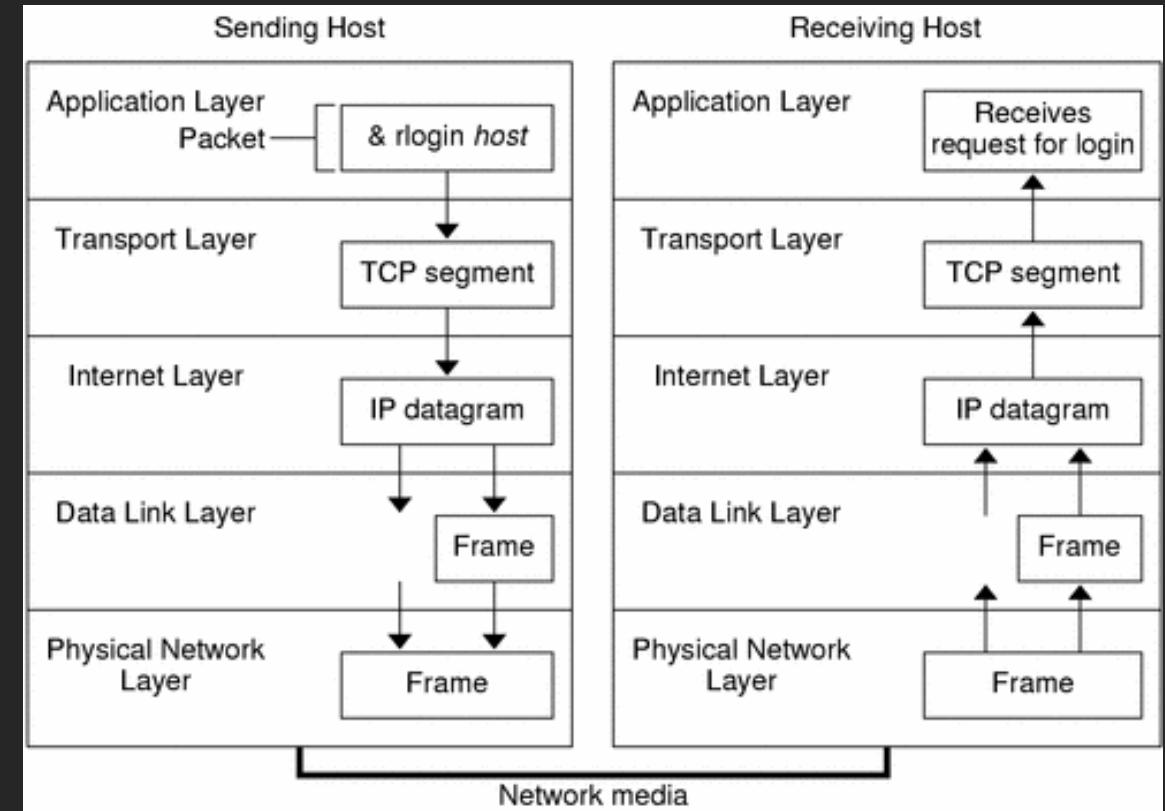
IPcte

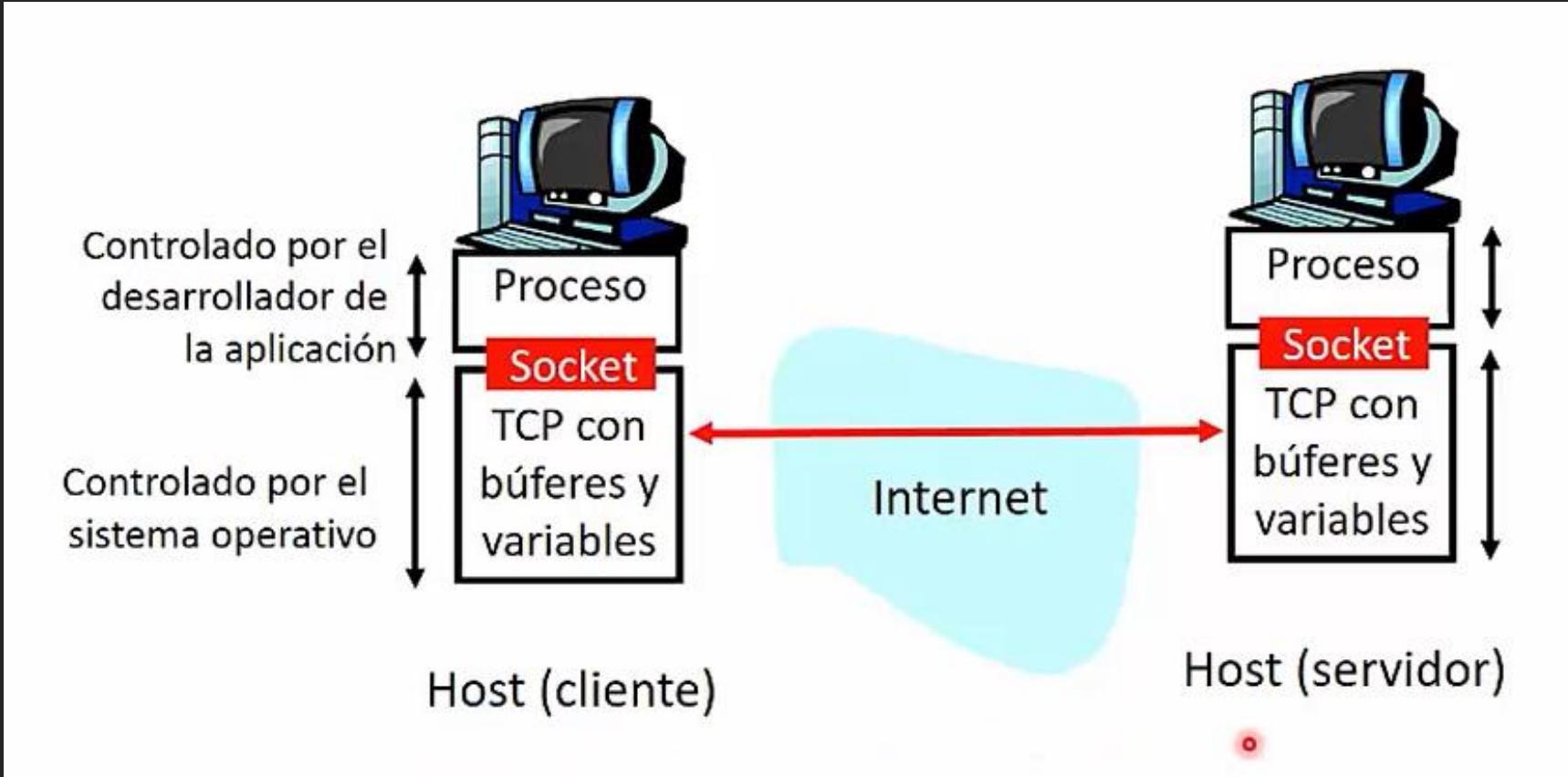


IPser



Cliente inicia comunicación





Programación de Sockets TCP

Servicios de sockets

C/C++ - Linux sockets

```
socket uns = socket(AF_INET , SOCK_STREAM , 0);
```

C/C++ - Winsockets

```
SOCKET uns = socket(family, socktype, protocol);
```

Plataforma Java

```
import java.net.*;
var uns = new Socket(Dirección, puerto);
```

Plataforma .NET

```
using System.Net.Sockets;
Socket uns = new Socket(Family, SocketType, ProtocolType);
```

Sockets en lenguajes

Cliente

1. Crear un socket local TCP del cliente
2. Indicar la dirección IP y puerto del servidor
3. Activar la conexión con el servidor
4. Enviar y recibir información
5. Cerrar la conexión

Servidor

1. Crear un socket local TCP del servidor
2. Establecer un puerto de escucha
3. Quedar a la espera de peticiones
4. Atender peticiones entrantes
5. Volver al paso 3

Programación de Sockets TCP

Servicio orientado a la conexión (TCP)	
Cliente	Servidor
	socket()
	bind()
	listen()
	accept()
socket()	
[bind()]	
connect()	
send()	recv()
recv()	send()

Secuencia de ejecución

Manejo de socket

- Creación de Socket

Los sockets se crean llamando a la función socket(), que devuelve el identificador de socket, de tipo entero

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
int socket(int domain, int type, int protocol);

#include <stdio.h> /* para perror() */
#include <sys/types.h>
#include <sys/socket.h>

...
int sockfd;
sockfd = socket ( PF_INET, SOCK_STREAM, 0 );
if(sockfd < 0)
perror("Error creating the socket");
```

Manejo de sockets

- Vinculación del socket a una dirección IP y un número de puerto específico

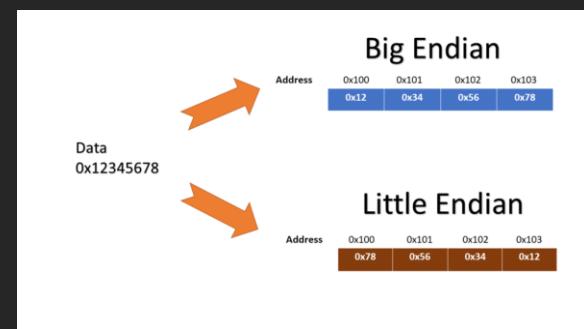
La función bind() se utiliza para asociar el socket a una dirección IP y número de puerto de la máquina local a través del que se enviarán y recibirán datos.

El formato de la función es el siguiente:

```
#include <sys/types.h>
#include <sys/socket.h>
...
int bind(int local_s, const struct sockaddr *addr, int addrlen);
```

```
struct sockaddr_in
{
    short int sin_family; // PF_INET
    unsigned short sin_port; // Numero de puerto.
    struct in_addr sin_addr; // Dirección IP.
    unsigned char sin_zero[8]; // Relleno.
};
```

```
...
struct sockaddr_in sin;
...
sin.sin_family = PF_INET;
sin.sin_port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
sin.sin_addr.s_addr = inet_addr ("132.241.5.10");
// IP por la que recibirá paquetes el programa
```



Manejo de sockets

- Vinculación

```
...
struct sockaddr_in sin;
...
sin.sin_family = PF_INET;
sin.sin_port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
sin.sin_addr.s_addr = inet_addr ("132.241.5.10");
// IP por la que recibirá paquetes el programa

...
struct sockaddr_in sin;
...
ret = bind (sockfd, (struct sockaddr *)&sin, sizeof (sin));
if(ret < 0)
    perror("Error binding the socket");
...
```

La llamada a la función `bind()` en el cliente es opcional, ya que en caso de no ser invocada el sistema la ejecutará automáticamente asignándole un puerto libre (al azar). En cambio, en el servidor es obligatorio ejecutar la llamada a la función `bind()` para reservar un puerto concreto y conocido.

Manejo de sockets

- Escuchar conexiones

El servidor escuche las conexiones entrantes mediante la función listen():

El servidor habilita su socket para poder recibir conexiones, llamando a la función listen().

- En el cliente este paso no es necesario, ya que no recibirá peticiones de conexión de otros procesos.

```
#include <sys/socket.h>
...
int listen(int sockfd, int backlog);
listen ( sockfd, 5);
```

Manejo de Sockets

- Aceptar conexiones

Cuando un cliente intenta conectarse, el servidor utiliza la función accept() para aceptar la conexión entrante

- Función listen() prepara el socket y lo habilita para recibir peticiones de establecimiento de conexión
- Función accept() la que realmente queda a la espera de estas peticiones.
- Cuando una petición realizada desde un proceso remoto (cliente) es recibida, la conexión se completa en el servidor siempre y cuando éste esté esperando en la función accept().
- La función accept() es utilizada en el servidor una vez que se ha invocado a la función listen().
- Esta función espera hasta que algún cliente establezca una conexión con el servidor.

Es una llamada bloqueante, esto es, la función no finalizará hasta que se haya producido una conexión o sea interrumpida por una señal.

Manejo de Sockets

- Es conveniente destacar que una vez que se ha producido la conexión, la función accept() devuelve un nuevo identificador de socket que será utilizado para la comunicación con el cliente que se ha conectado.

```
#include <sys/types.h>
#include <sys/socket.h>
...
int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);
```

```
new_sockfd = accept (sockfd, &remote_addr, &addrlen);
```

Manejo de sockets

```
...
int sockfd, new_sockfd;
struct sockaddr_in server_addr;
struct sockaddr_in remote_addr;
int addrlen;
// Creación del socket.
sockfd = socket (PF_INET, SOCK_STREAM, 0 );
// Definir valores en la estructura server_addr.
server_addr.sin_family = PF_INET;
server_addr.sin_port = htons ( 1234 ); // Número de puerto donde
// recibirá paquetes el programa
server_addr.sin_addr.s_addr = htonl(INADDR_ANY);
// Asociar valores definidos al socket
bind(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));
// Se habilita el socket para poder recibir conexiones.
listen ( sockfd, 5);
addrlen = sizeof (struct sockaddr );
// Se llama a accept() y el servidor queda en espera de conexiones.
new_sockfd = accept (sockfd, &remote_addr, &addrlen);
...
```

Lanzar peticiones de conexión

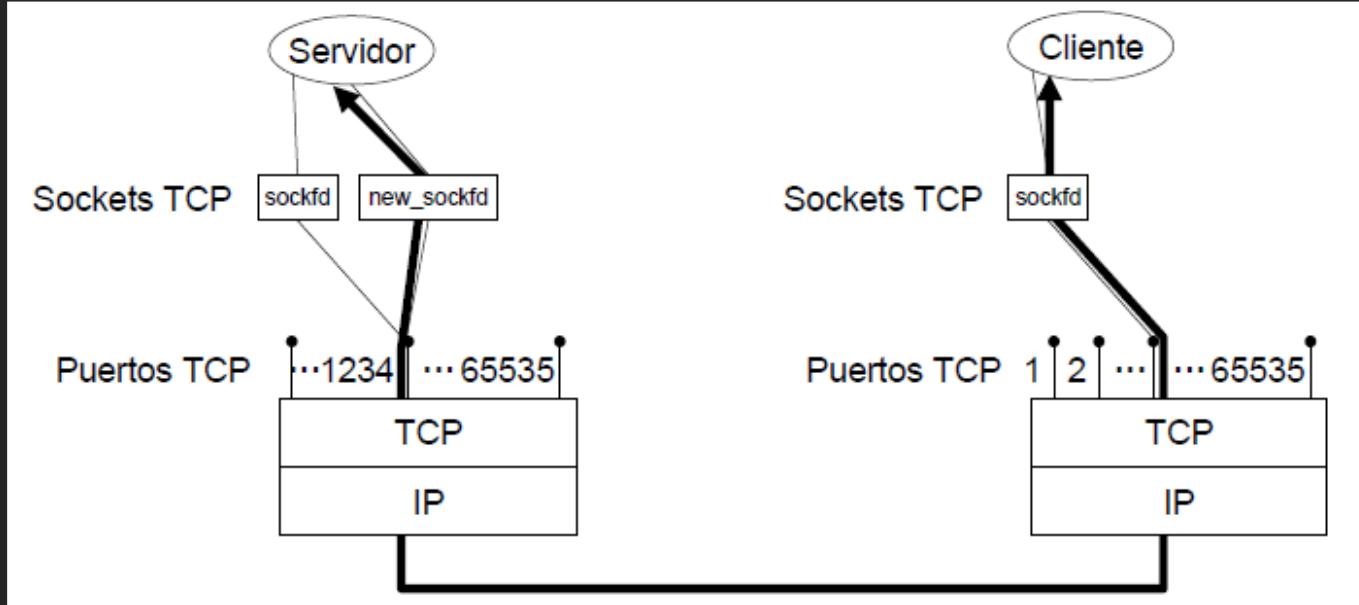
- Esta función es invocada desde el cliente para solicitar el establecimiento de una conexión TCP.

La función connect() inicia la conexión con el servidor remoto, por parte del cliente. El formato de la función es el siguiente:

```
#include <sys/types.h>
#include <sys/socket.h>
...
int connect(int sockfd, const struct sockaddr *serv_addr, socklen_t addrlen);
```

Lanzar peticiones de conexión

```
...
int sockfd;
struct sockaddr_in server_addr;
int addrlen;
// Creación del socket.
sockfd = socket (PF_INET, SOCK_STREAM, 0 );
// Definir valores en la estructura server_addr.
server_addr.sin_family = PF_INET;
server_addr.sin_port = htons ( 1234 );
// Número de puerto donde
// está esperando el servidor
server_addr.sin_addr.s_addr = inet_addr("1.2.3.4");
// Dirección IP
// del servidor
addrlen = sizeof (struct sockaddr );
// Se llama a connect () y se hace la petición de conexión al servidor
connect (sockfd, &server_addr, addrlen);
...
```



Conexión entre los sockets del cliente y servidor

- El cliente ha hecho la llamada a la función `connect()`
- Al concluir con éxito (lo cual significa que en el servidor se estaba esperando en la función `accept()` y se ha salido de ella y por tanto se ha creado el nuevo socket)

Enviar y recibir datos a través del socket

- Una vez que la conexión ha sido establecida, se inicia el intercambio de datos, utilizando para ello las funciones send() y recv().

La función send() devuelve el número de bytes enviados, que puede ser menor que la cantidad indicada en el parámetro len.

```
#include <sys/types.h>
#include <sys/socket.h>
...
ssize_t send(int s, const void *buf, size_t len, int flags);

ssize_t send (int sockfd, const void *buf, size_t len, int flags );
```

La función recv() se utiliza para recibir datos

```
#include <sys/types.h>
#include <sys/socket.h>
...
ssize_t recv(int s, void *buf, size_t len, int flags);

ssize_t recv (int sockfd, void *buf, size_t len, int flags);
```

Enviar y recibir datos a través del socket

- La función recv() es bloqueante

No finaliza hasta que se ha recibido algún tipo de información.

Resaltar que el campo len indica el número máximo de bytes a recibir

- No necesariamente se han de recibir exactamente ese número de bytes.

La función recv() devuelve el número de bytes que se han recibido.

```
...
char mens_serv[100];
...
mens_clien = "Ejemplo";
send(sid, mens_clien, strlen(mens_clien)+1, 0);
...
recv(sid, mens_serv, 100, 0);
...
```

Cierre de un socket

- Simplemente hay que hacer la llamada a la función pasándole como argumento el descriptor del socket que se quiere cerrar.
- Es importante que cuando un socket no vaya a usarse más, se haga la llamada a close() para indicárselo al Sistema Operativo y que éste lo libere.

```
#include <unistd.h>
...
int close(int fd);
```

TECNOLÓGICO NACIONAL DE MÉXICO

Ingeniería en Sistemas Computacionales

Fundamentos de Telecomunicaciones

Unidad V: Multiplexación

Material de clase desarrollado para la asignatura de **Fundamentos de Telecomunicaciones**
para Ingeniería en Sistemas Computacionales

FUNDAMENTOS DE TELECOMUNICACIONES

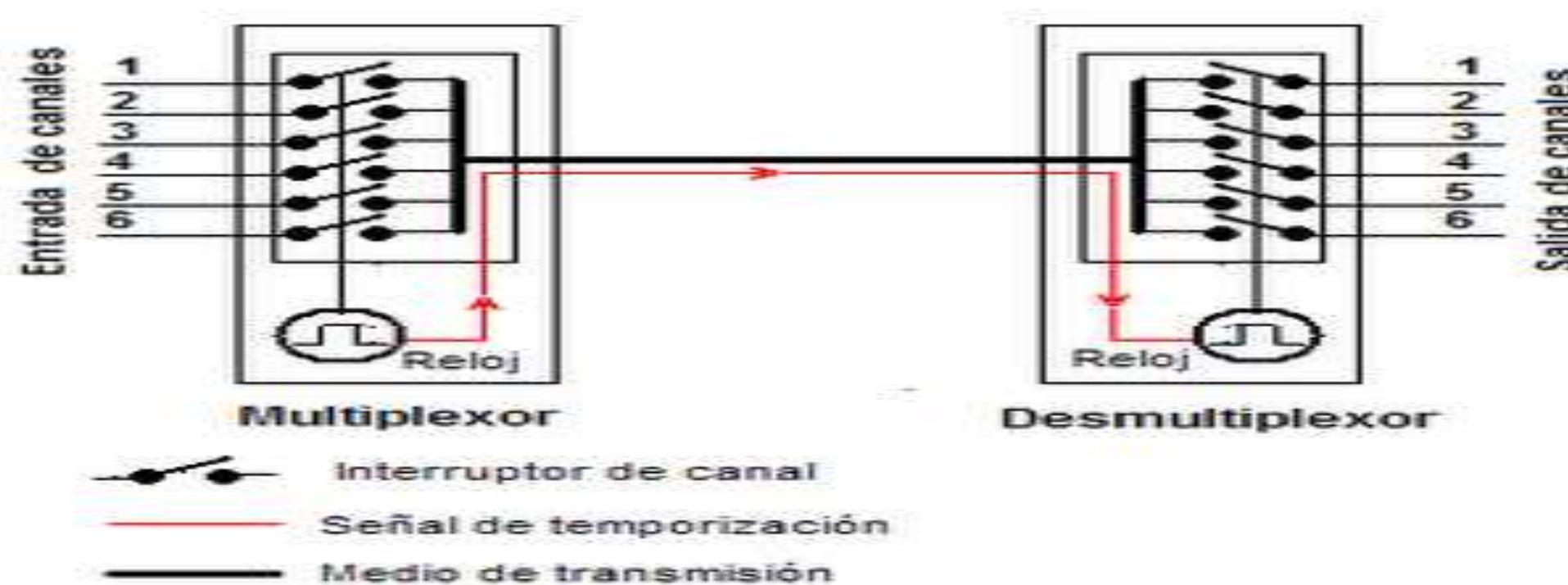
FUNDAMENTOS DE TELECOMUNICACIONES

Temario de la Unidad

Unidad	Temas	Subtemas
5	Multiplexación	5.1 TDM División de tiempo 5.2 FDM División de frecuencia 5.3 WDM División de longitud 5.4 CDM División de código

FUNDAMENTOS DE TELECOMUNICACIONES

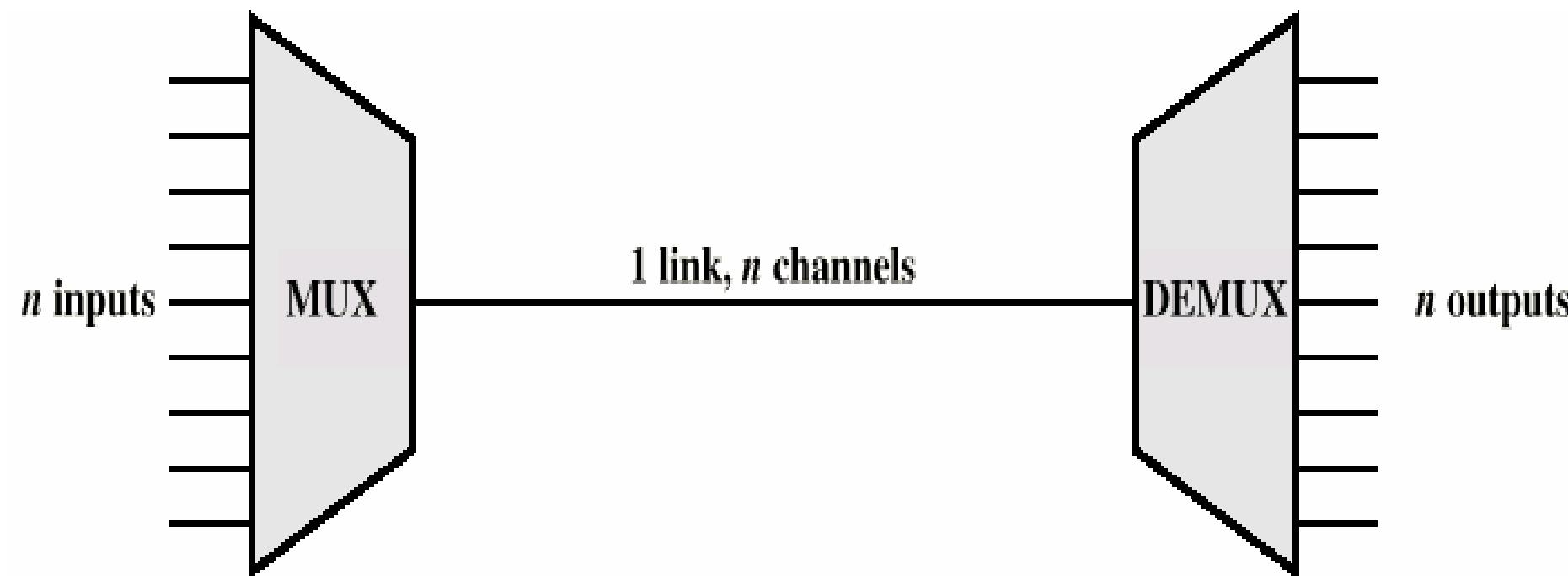
- La Multiplexación es la combinación de dos o más canales de información en un solo medio de transmisión usando un dispositivo llamado multiplexor.
- Es decir viene a ser un procedimiento por el cual diferentes canales pueden compartir un mismo medio de transmisión de información.



FUNDAMENTOS DE TELECOMUNICACIONES

Objetivo de la Multiplexación

- Es compartir la capacidad de transmisión de datos sobre un mismo enlace para aumentar la eficiencia (sobre todo en líneas de grandes distancias).
- Minimizar la cantidad de líneas físicas requeridas y maximizar el uso del ancho de banda de los medios



FUNDAMENTOS DE TELECOMUNICACIONES

La Multiplexación o multicanalización es la transmisión de información, de más de una fuente a más de un destino, por el mismo medio de transmisión.

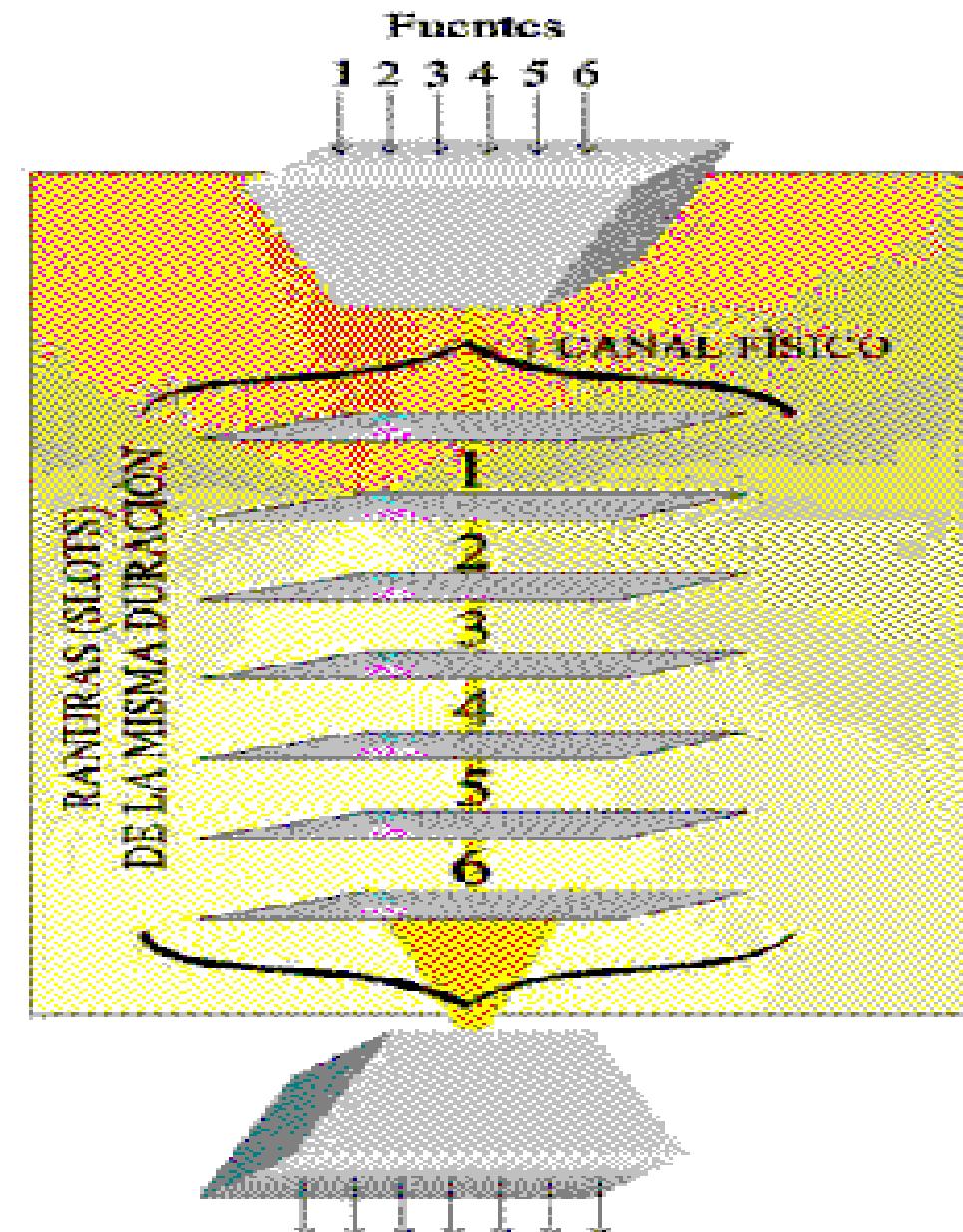
Los principales métodos de realizar este proceso son:

- La multiplexación de división de frecuencia (FDM: Frequency Division Multiplexing),
- La multiplexación por división de código (CDM: Coded Division Multiplexing),
- La multiplexación por división de longitud de onda (WDM: Wavelength Division Multiplexing), y
- La multiplexación por división de tiempo (TDM: Time Division Multiplexing).

FUNDAMENTOS DE TELECOMUNICACIONES

Multiplexación por División de Tiempo (TDM)

- La **multiplexación por división de tiempo** es una técnica para compartir un canal de transmisión entre varios usuarios.
- Consiste en asignar a cada usuario, durante unas determinadas "ranuras de tiempo", la totalidad del ancho de banda disponible.



FUNDAMENTOS DE TELECOMUNICACIONES

Características del TDM

- Consiste en ocupar un canal de transmisión a partir de distintas fuentes, mejor aprovechamiento del medio de transmisión.
- El ancho de banda total del medio de transmisión es asignado a cada canal durante una fracción del tiempo total (intervalo de tiempo).

FUNDAMENTOS DE TELECOMUNICACIONES

Ventajas de TDM

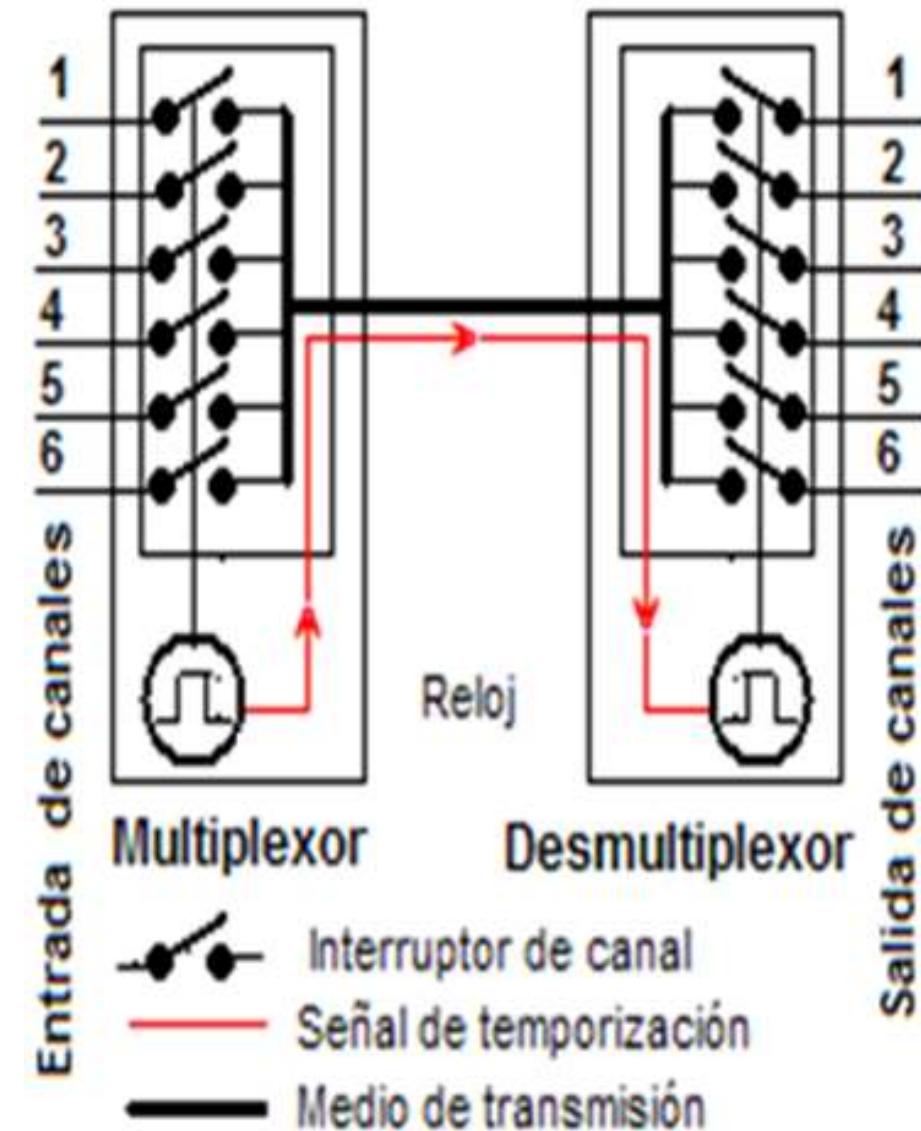
- ❖ El uso de la capacidad es alto.
- ❖ Cada uno para ampliar el número de usuarios en un sistema en un coste bajo.

Desventajas de TDM

- ❖ La sensibilidad frente a otro problema de usuario es alta.
- ❖ El coste inicial es alto.
- ❖ La complejidad técnica.

FUNDAMENTOS DE TELECOMUNICACIONES

Esquema de cómo se realiza la Multiplexación – Desmultiplexación

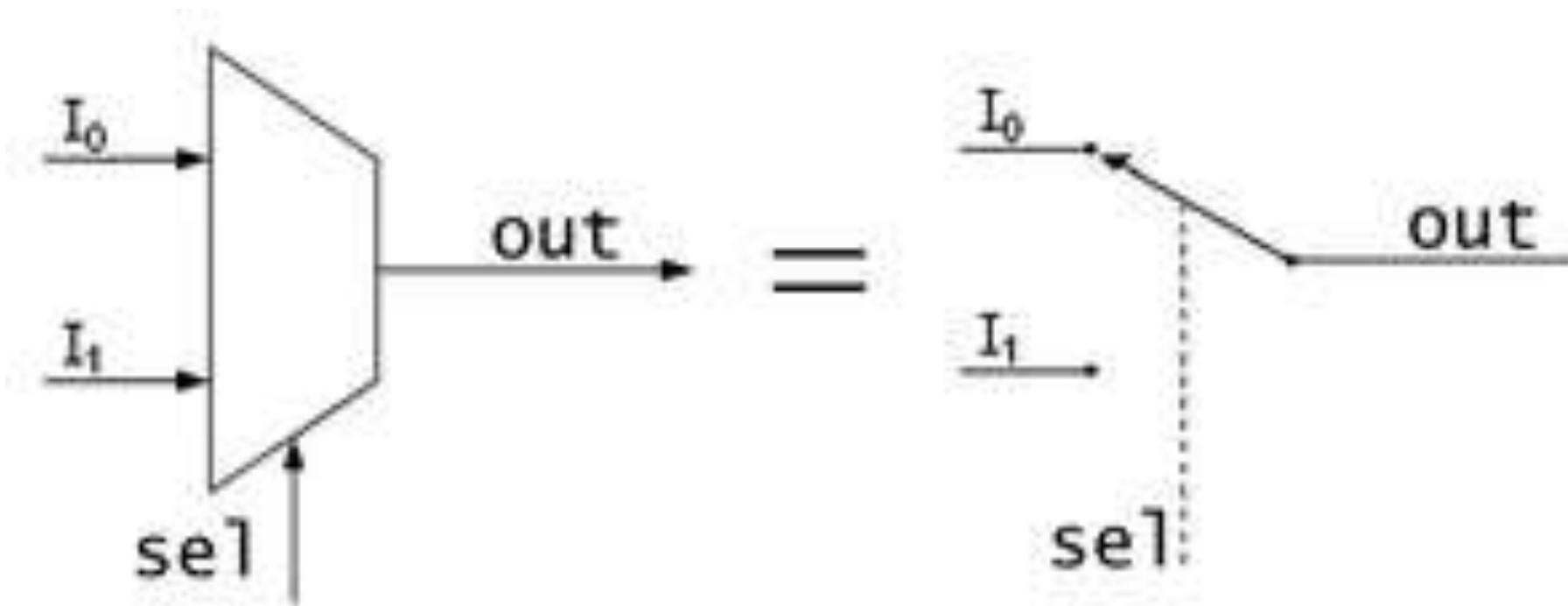


- Las entradas de 6 canales llegan a los interruptores de canal controlados por una señal de reloj, de manera que cada canal es conectado al medio de Tx durante un tiempo determinado por la duración de los impulsos del reloj.
- El Desmultiplexor realiza la función inversa: conecta al medio de Tx, secuencialmente con la salida de cada uno de los 5 canales mediante interruptores controlados por el reloj del D.
- El reloj del extremo receptor funciona de forma sincronizada con el del Multiplexor mediante señales de temporización que son transmitidas a través del propio medio de Tx.

FUNDAMENTOS DE TELECOMUNICACIONES

Que es un Multiplexor

- Es un dispositivo que puede recibir varias entradas y transmitirlas por un medio de transmisión compartido; es decir, divide el medio de transmisión en múltiples canales, para que varios nodos puedan comunicarse al mismo tiempo.



FUNDAMENTOS DE TELECOMUNICACIONES

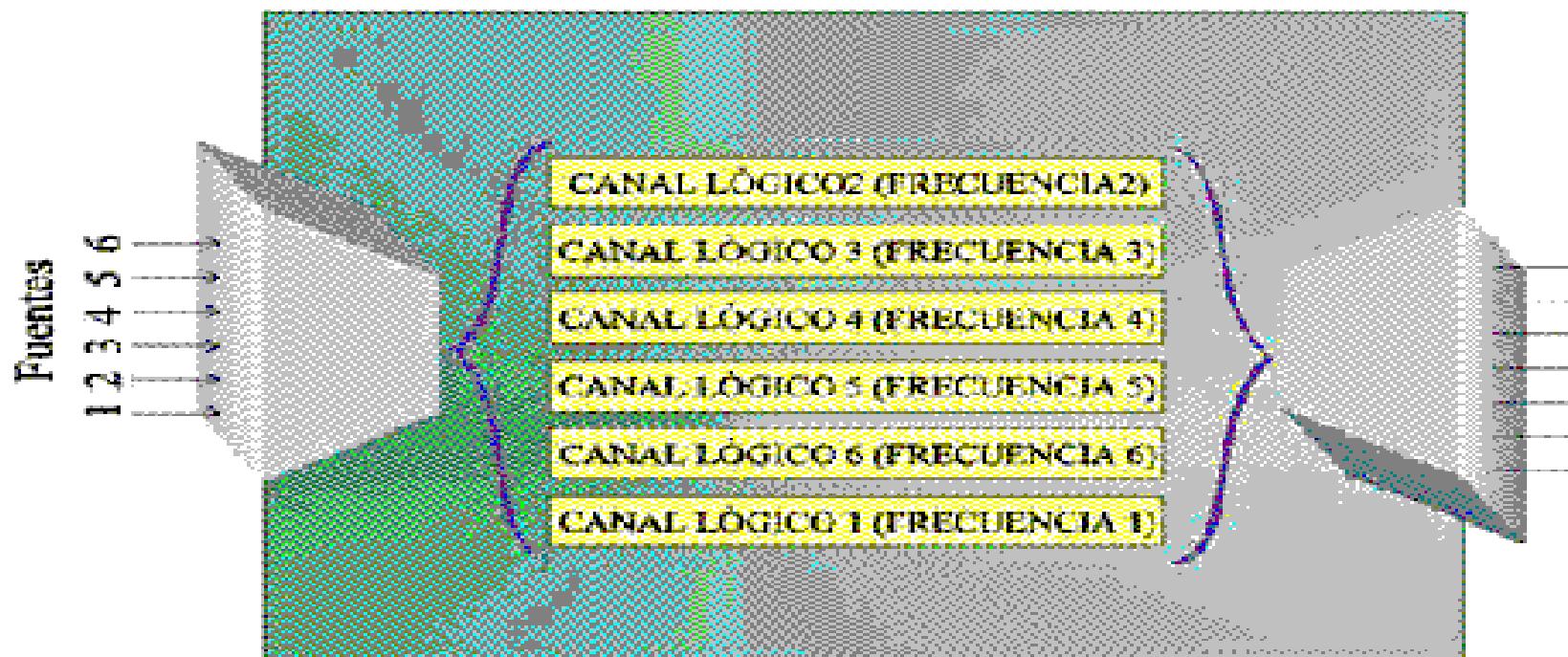
La función de un multiplexor da lugar a diversas aplicaciones:

- Selector de entradas.
- Serializador: Convierte datos de paralelo a serial.
- Transmisión multiplexada: En una misma línea de conexión, transmite diferentes datos de distinta procedencia.
- Realización de funciones lógicas: Utilizando inversores y conectando a 0 ó 1 las entradas se puede diseñar funciones de un modo más compacto, que utilizando puertas lógicas.

FUNDAMENTOS DE TELECOMUNICACIONES

División de frecuencia (FDM)

- Esta técnica que consiste en dividir mediante filtros el espectro de frecuencias del canal de transmisión y desplazar la señal a transmitir dentro del margen del espectro correspondiente mediante modulaciones, de tal forma que cada usuario tiene posesión exclusiva de su banda de frecuencias.



FUNDAMENTOS DE TELECOMUNICACIONES

- Para optimizar la utilización del medio de transmisión, se ha desarrollado la multiplexación, que es un conjunto de técnicas que permite la transmisión simultanea de múltiples señales a través de un único enlace.



FUNDAMENTOS DE TELECOMUNICACIONES

Características del FDM

- El ancho de banda del medio debe ser mayor que le ancho de banda de la señal transmitida.
- Capacidad de transmisión de varias señales a la vez.
- La señal lógica trasmisita a través del medio es analógica.
- La señal recibida puede ser analógica o digital.
- Para la comunicación análoga el ruido tiene menos efecto.

FUNDAMENTOS DE TELECOMUNICACIONES

VENTAJAS DE FDM

- El sistema de FDM apoya el flujo de dúplex total de información que es requerido por la mayor parte de la aplicación.
- El problema del ruido para la comunicación análoga tiene menos el efecto.
- Aquí el usuario puede ser añadido al sistema por simplemente añadiendo otro par de modulador de transmisor y modulador receptor.

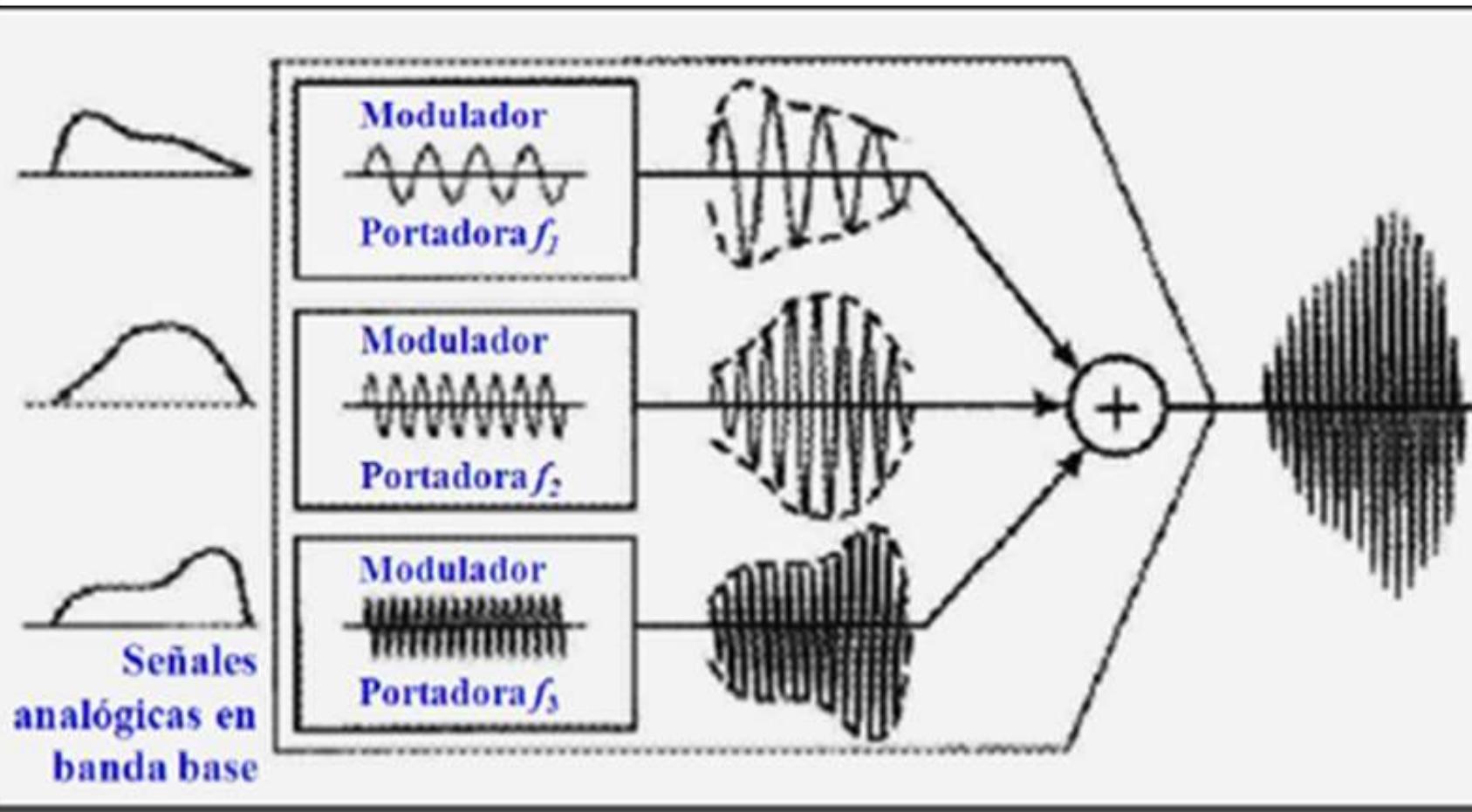
FUNDAMENTOS DE TELECOMUNICACIONES

DESVENTAJAS DE FDM

- En el sistema FDM, el coste inicial es alto. Este puede incluir el cable entre los dos finales y los conectores asociados para el cable.
- En el sistema FDM, un problema para un usuario puede afectar a veces a otros.
- En el sistema FDM, cada usuario requiere una frecuencia de portador precisa.

FUNDAMENTOS DE TELECOMUNICACIONES

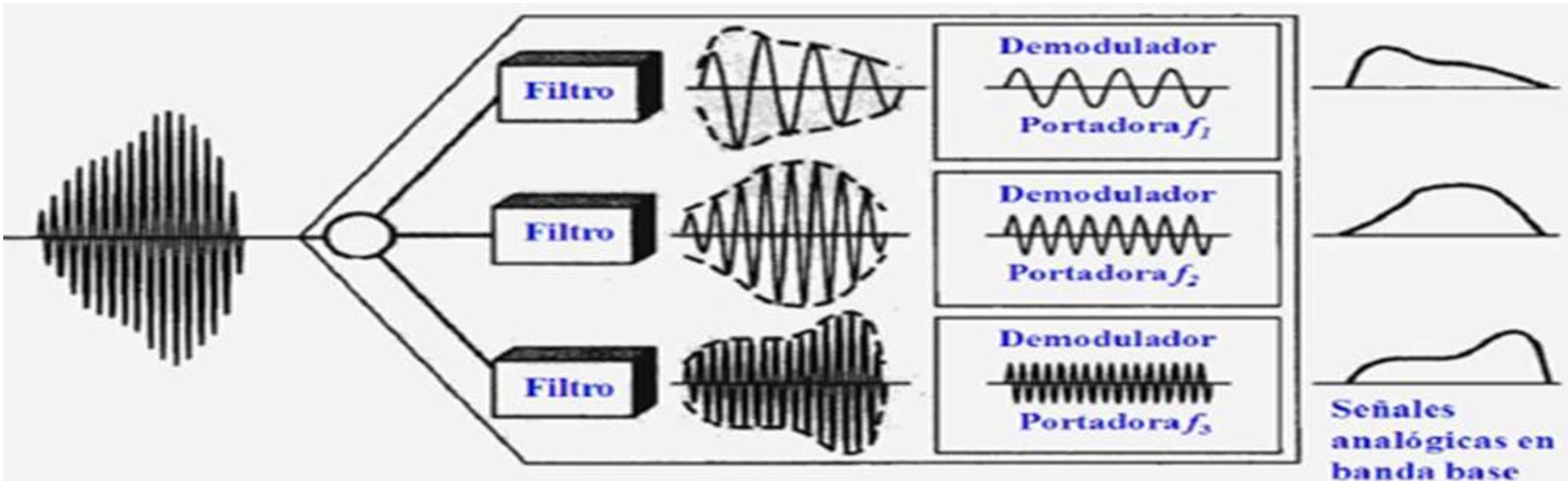
Proceso de Multiplexación



- Cada fuente genera una señal con un rango de frecuencia similar. Dentro del MUX, estas señales similares se modulan sobre distintas frecuencias portadoras (f_1, f_2, f_3 , etc.)
- Las señales moduladas se combinan en un única señal compuesta que se envía sobre un enlace.

FUNDAMENTOS DE TELECOMUNICACIONES

Proceso de Desmultiplexación



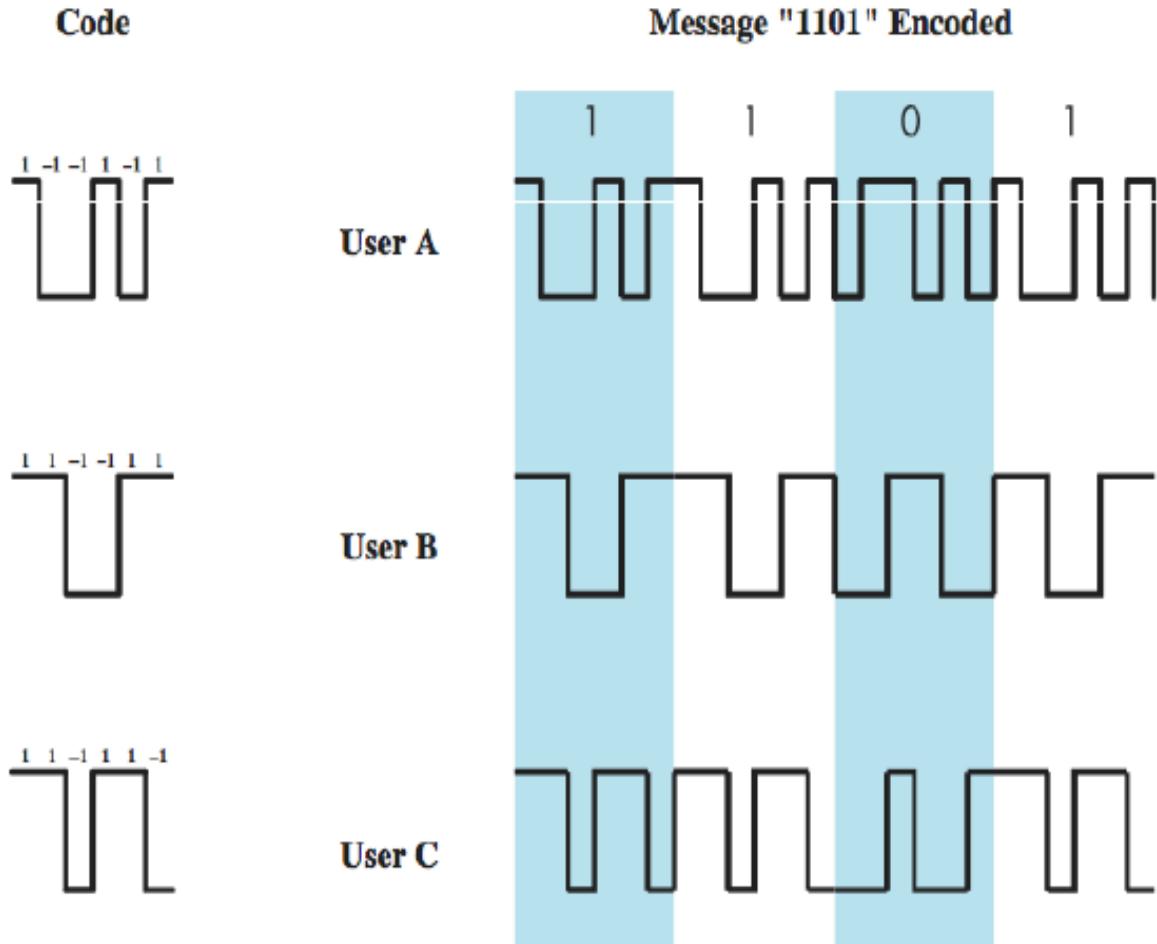
- El DEMUX usa filtros para descomponer la señal multiplexada en las señales que la constituyen.
- Las señales individuales se pasan después a un demodulador que las separa de sus portadoras y las pasa a la línea de salida

FUNDAMENTOS DE TELECOMUNICACIONES

La multiplexación por División en Código o CDM

- La **multiplexación por división de código, acceso múltiple por división de código o CDMA**
- Es un término genérico para varios métodos de multiplexación o control de acceso al medio basado en la tecnología de espectro expandido.

FUNDAMENTOS DE TELECOMUNICACIONES

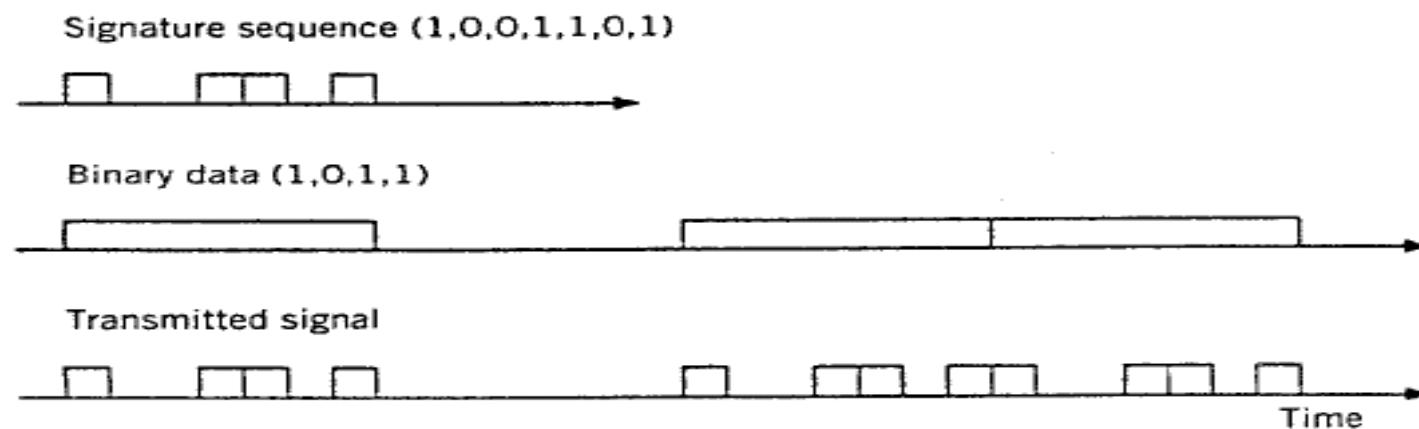


- CDMA emplea una tecnología de espectro expandido y un esquema especial de codificación, por el que a cada transmisor se le asigna un código único, escogido de forma que sea ortogonal respecto al del resto
- En CDMA, la señal se emite con un ancho de banda mucho mayor que el precisado por los datos a transmitir; por este motivo, la división por código es una técnica de acceso múltiple de espectro expandido.

FUNDAMENTOS DE TELECOMUNICACIONES

Tipos de multiplexación División de Código

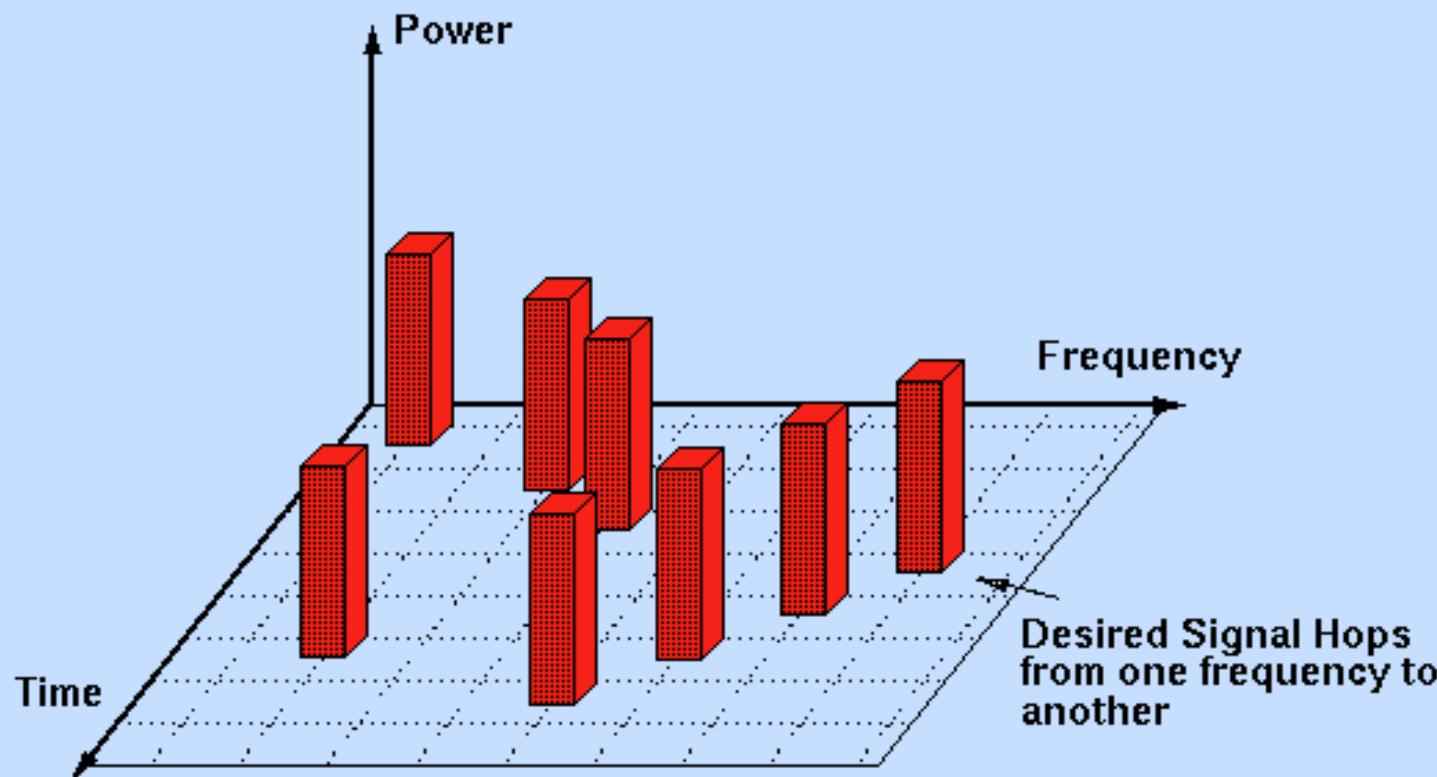
ESQUEMA CDMA



- Ensanchamiento espectral dado por la clave.
- Las claves las conocen el Rx y Tx.

FUNDAMENTOS DE TELECOMUNICACIONES

Espectral



- Esparcimiento espectral se hace con saltos en frecuencia (frequency hopping).
- Los saltos de frecuencia están dados por el código.

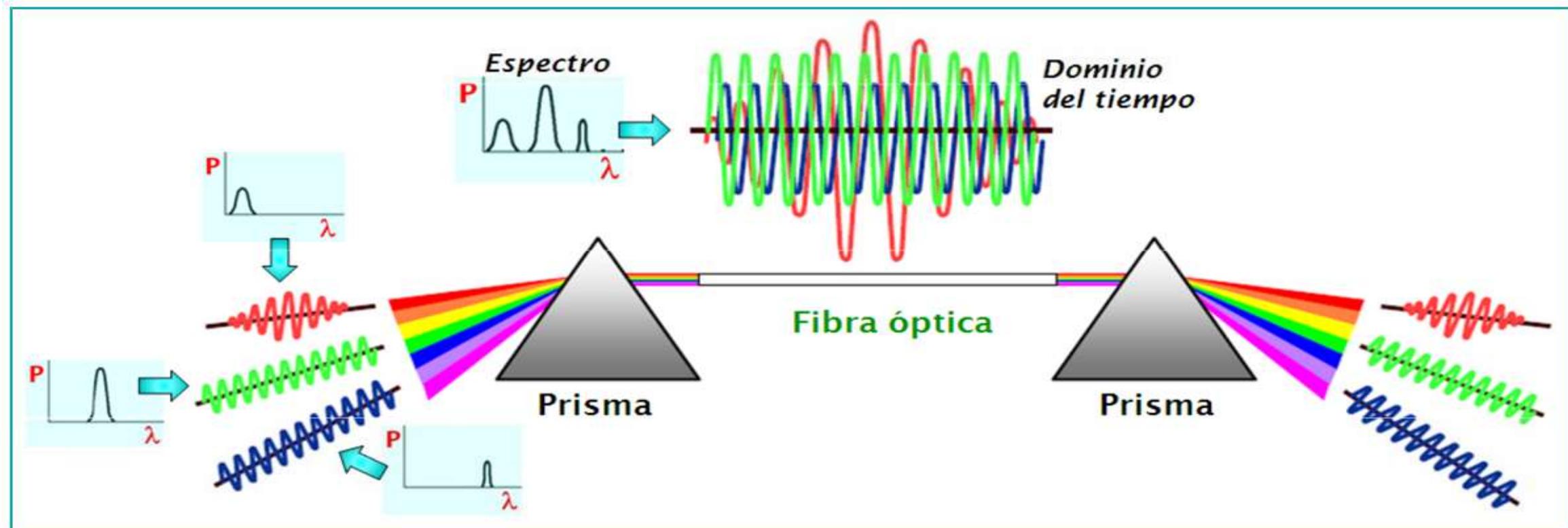
FUNDAMENTOS DE TELECOMUNICACIONES

División de Onda o División de Longitud (WDM)

- En las comunicaciones de fibra óptica, multiplexación por división de longitud de onda es una tecnología que multiplexa un número de señales portadoras ópticas en una sola fibra óptica mediante el uso de diferentes longitudes de onda de la luz láser.
- Esta técnica permite comunicaciones bidireccionales sobre una hebra de fibra, así como la multiplicación de la capacidad.

FUNDAMENTOS DE TELECOMUNICACIONES

- Se diseñó para utilizar la capacidad de alta tasa de datos de la fibra.
- Conceptualmente es la misma que FDM, excepto que involucra señales luminosas de frecuencias muy altas.



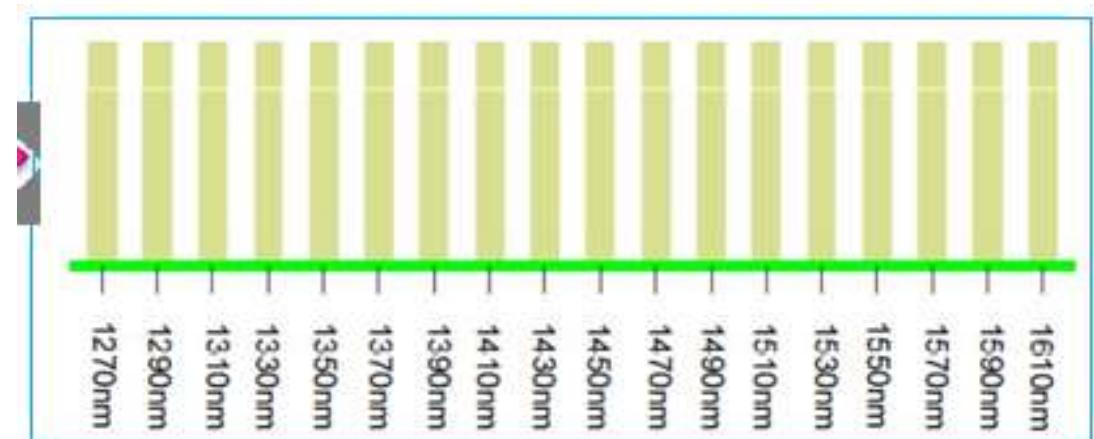
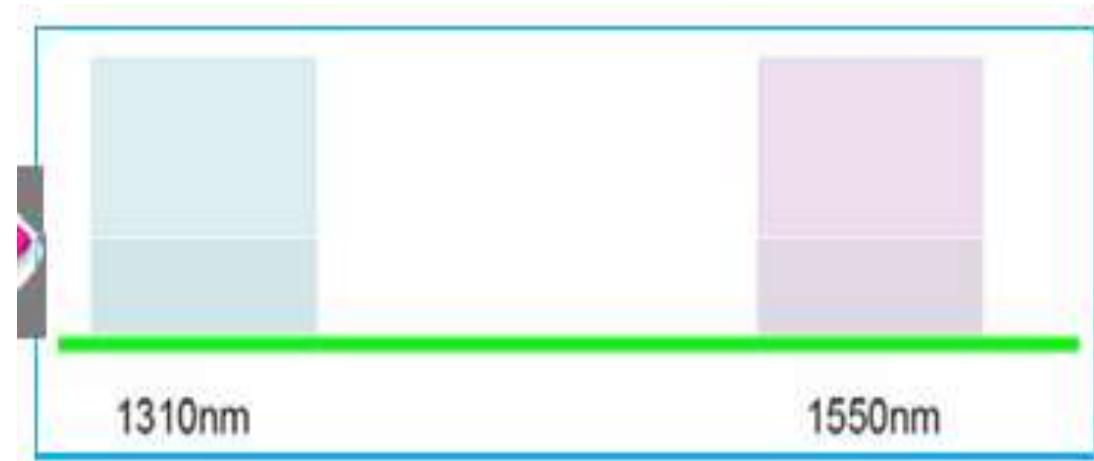
FUNDAMENTOS DE TELECOMUNICACIONES

- El término multiplexación por división de longitud de onda se aplica comúnmente a una portadora óptica, mientras que la multiplexación por división de frecuencia típicamente se aplica a una portadora de radio.
- Dado que la longitud de onda y la frecuencia están unidas entre sí a través de una simple relación directamente inversa, los dos términos en realidad describen el mismo concepto.

FUNDAMENTOS DE TELECOMUNICACIONES

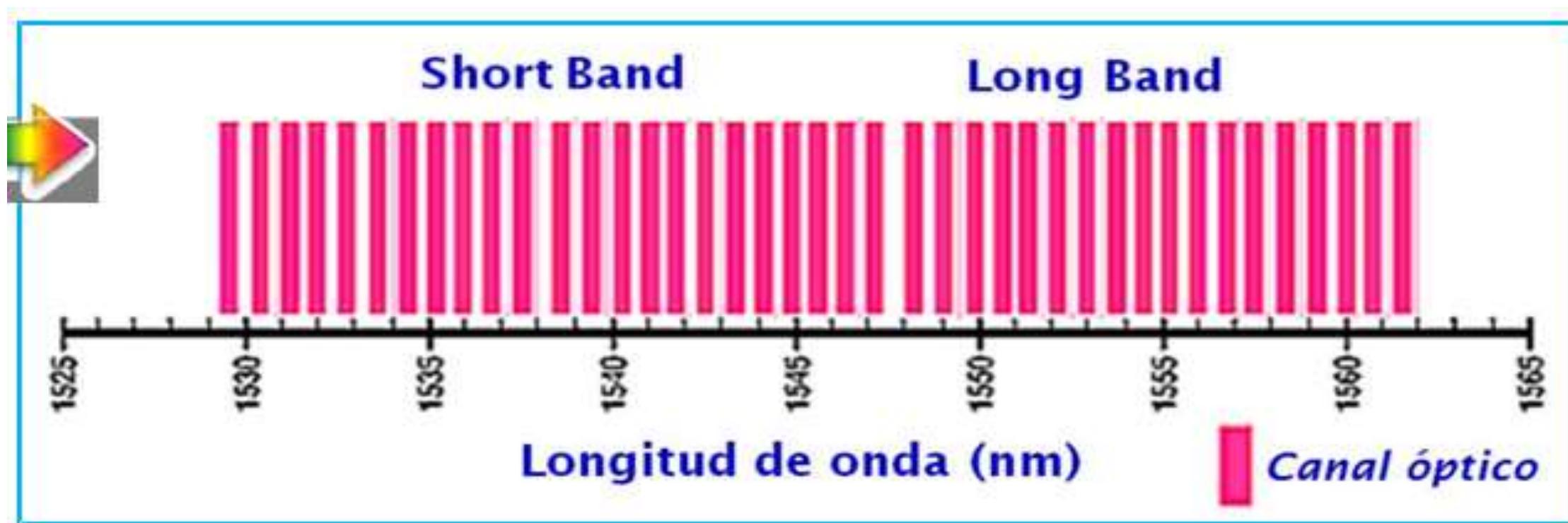
Tipos de sistemas WDM

- Los primeros sistemas WDM usaron 2 longitudes de onda centradas en las ventanas de 1310 nm y 1550 nm.
- Después fue CWDM (*Coarse WDM*) . La ITU (G.694.2) define una banda óptica de 18 l's, entre 1270 y 1610 nm, espaciadas entre ellas 20 nm.



FUNDAMENTOS DE TELECOMUNICACIONES

- Alrededor de 1.400 nm existe una atenuación alta debido al pico de absorción. Se fabrican fibras con este pico de absorción compensado.
- Luego fue DWDM (*Dense WDM*) . La ITU (G.692) define una banda óptica de 20 a 40 l's , entre 1530 y 1570 nm.



FUNDAMENTOS DE TELECOMUNICACIONES

- *Se usan 2 separaciones:*
 - 200 GHz (1.6 nm)
 - 100 GHz (0.8 nm)
- Ya hay disponibles sistemas UWDM (*Ultradense WDM*) con separaciones más densas.
 - 50 GHz (0.4 nm)
 - 25 GHz (0.2 nm)

FUNDAMENTOS DE TELECOMUNICACIONES

La idea es simple:

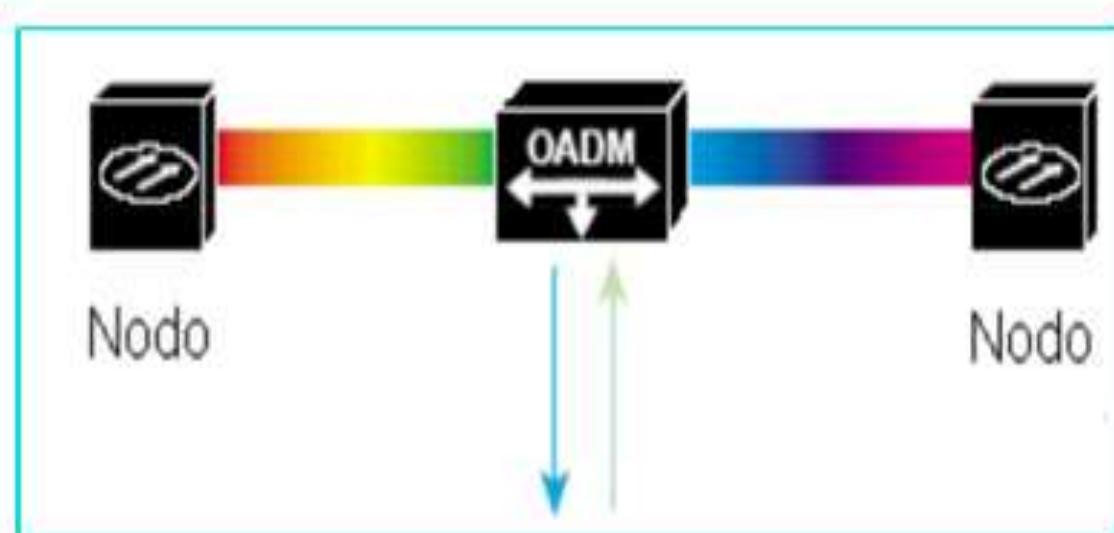
- Se quieren combinar múltiples áces de luz dentro de una única luz en el multiplexor, Hacer a operación inversa en el desmultiplexor.
- Combinar y dividir haces de luz se resuelve fácilmente mediante un prisma. Un prisma curva un rayo de luz basándose en el ángulo de incidencia y la frecuencia.

FUNDAMENTOS DE TELECOMUNICACIONES

Topologías para DWDM

Topología punto a punto

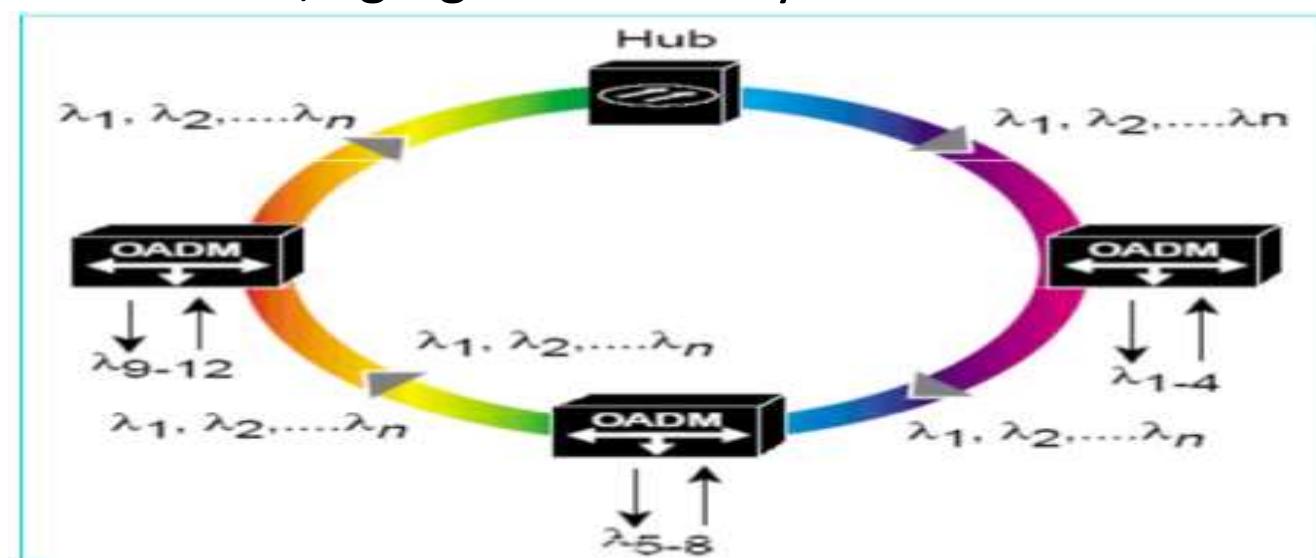
- La fibra y el tráfico son lineales.
- Se usan en redes de transporte WAN y de acceso metropolitano.
- Con o sin multiplexor óptico OADM. Son de alta velocidad; actualmente hasta 160 Gbps.
- Pueden cubrir varios cientos a miles de km, con menos de 10 amplificadores.
- En redes de acceso metropolitano no se necesitan amplificadores.



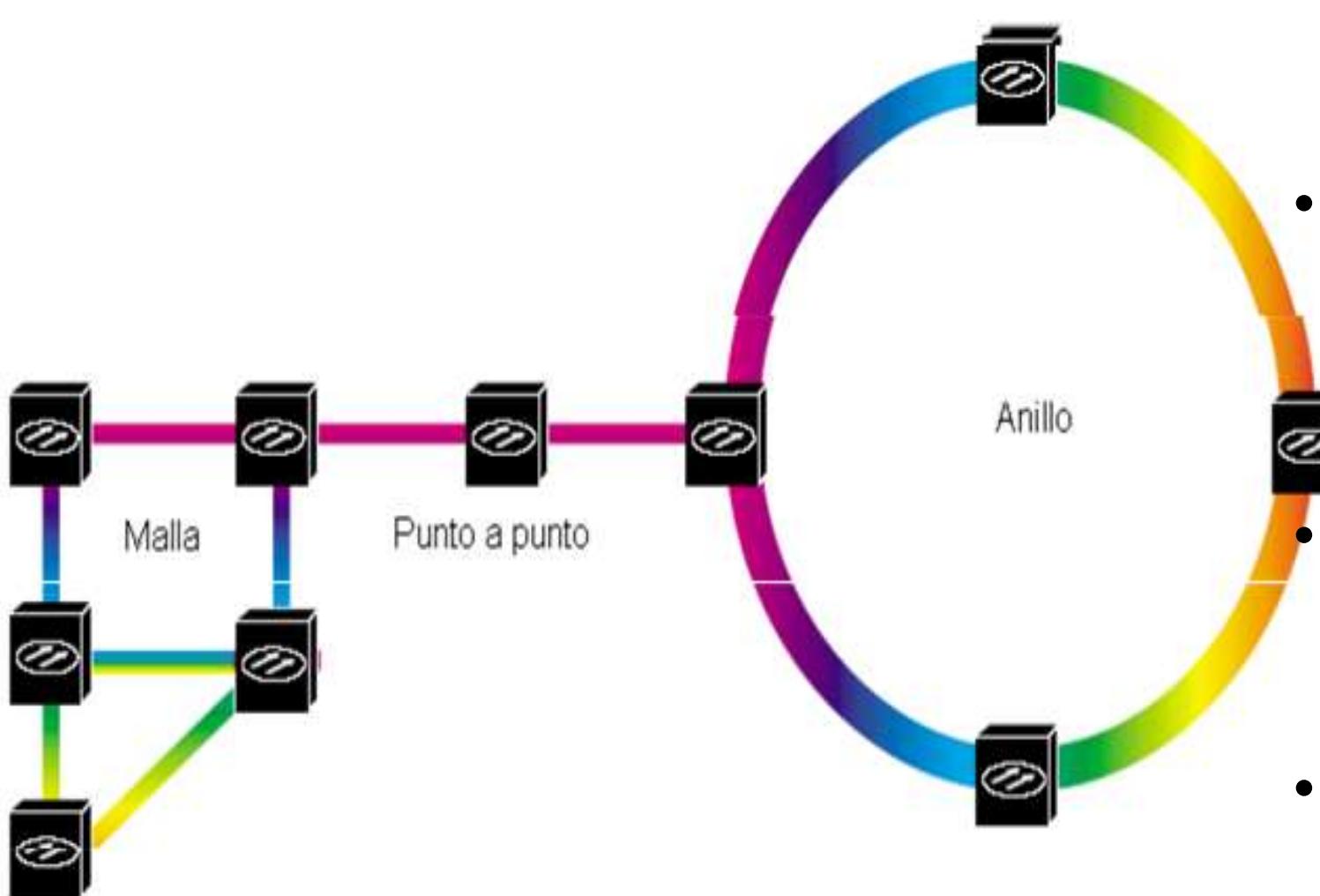
FUNDAMENTOS DE TELECOMUNICACIONES

Topología en anillo

- La fibra se instala en anillo. Los canales de tráfico se transmiten a través de los OADM hasta alcanzar su destino.
- Se usa en redes de acceso metropolitano; es típico que existan menos nodos que canales.
- La velocidad de tráfico está en el rango de 622 Mbps a 10 Gbps por canal, y pueden cubrir decenas de km sin amplificación.
- Las topologías en anillo permiten a los nodos OADM proporcionar el acceso para conectar routers, switches o servidores, agregando o ex rayen o canales en el dominio óptico.



FUNDAMENTOS DE TELECOMUNICACIONES



Topología en Malla

- Todos los nodos ópticos se interconectan entre sí. Se usan en redes de acceso metropolitano.
- Requiere esquemas de protección con redundancia al sistema, tarjeta o nivel de fibra.
- La arquitectura en malla es el futuro de las arquitecturas en redes ópticas.

FUNDAMENTOS DE TELECOMUNICACIONES

Tarea

Realizar el cuadro sinóptico correspondiente a la unidad



REDES DE COMUNICACIONES Y CÓMPUTO

Unidad V

Redes I



REDES DE COMUNICACIONES

REDES DE INFORMACIÓN MASIVA



- Una red de información masiva suele generalizarse como:
 - *Medios masivos de comunicación o mass media*
 - Conjunto de recursos que transmiten información a una gran cantidad de personas de forma simultanea
 - *Algunos ejemplos de medios masivos de comunicación son:*
 - Prensa escrita
 - Radio
 - Televisión
 - Internet
 - Imprenta
 - *Los medios masivos de comunicación influyen en la opinión pública*



CONEXIÓN EN RED INFORMÁTICA

- La conexión en red, o interconexión informática
 - *Proceso de conectar dos o más dispositivos informáticos para permitir la transmisión y el intercambio de información y recursos, como:*
 - Computadoras de escritorio
 - Dispositivos móviles
 - Routers
 - Aplicaciones
 - Etc.
 - Los dispositivos en red se basan en protocolos de comunicaciones para compartir información mediante conexiones físicas o inalámbricas.

EVOLUCIÓN DE LAS REDES

- Desplazamiento físico de las computadoras para compartir datos entre dispositivos
 - *El Departamento de Defensa financió la creación de la primera red informática en funcionamiento (que finalmente se llamó ARPANET) a fines de la década de 1960.*
- Desde entonces, las prácticas de trabajo en red han evolucionado en gran medida.
- Las redes informáticas actuales facilitan la comunicación a gran escala entre dispositivos para todos los fines:
 - *Empresariales*
 - *De entretenimiento*
 - *De investigación.*
 - *Internet, las búsquedas en línea, el correo electrónico, el intercambio de audio y video, el comercio en línea, las retransmisiones en directo y las redes sociales existen gracias a los avances de las redes informáticas.*

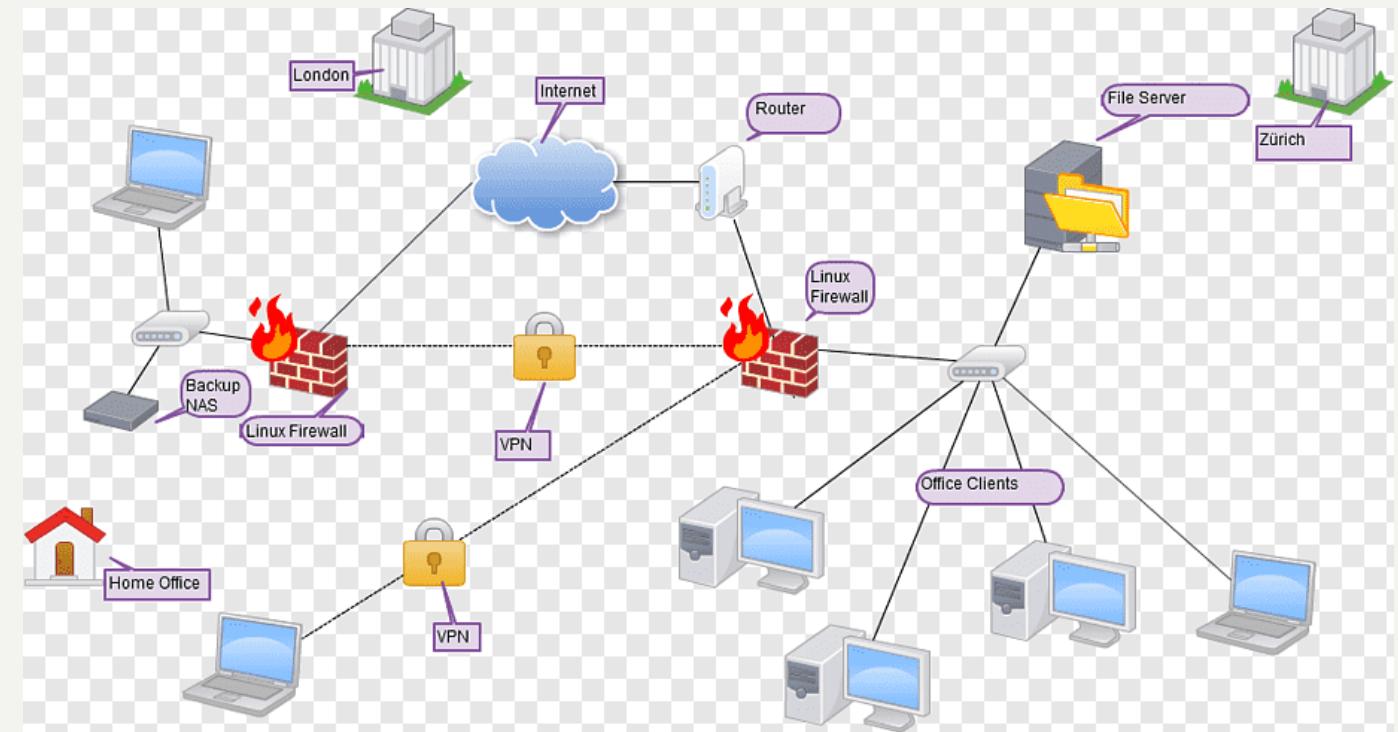
- Una red de información masiva es una red de computadoras interconectadas a nivel mundial que permite el acceso público a información. Un ejemplo de red de información masiva es Internet
 - *Internet es una red descentralizada de redes de comunicaciones que utiliza protocolos para garantizar que las redes físicas se conecten en una sola red lógica.*
 - *Internet se puede acceder a través de diferentes medios de conexión como:*
 - Líneas telefónicas
 - Cables
 - 4G
 - 5G
 - Fibra óptica
 - ADSL.
 - *Para acceder a los sitios web, se utilizan navegadores.*
- La primera conexión de computadoras, conocida como ARPANET, se estableció en 1969 entre tres universidades de California.

REDES DE INFORMACIÓN MASIVA



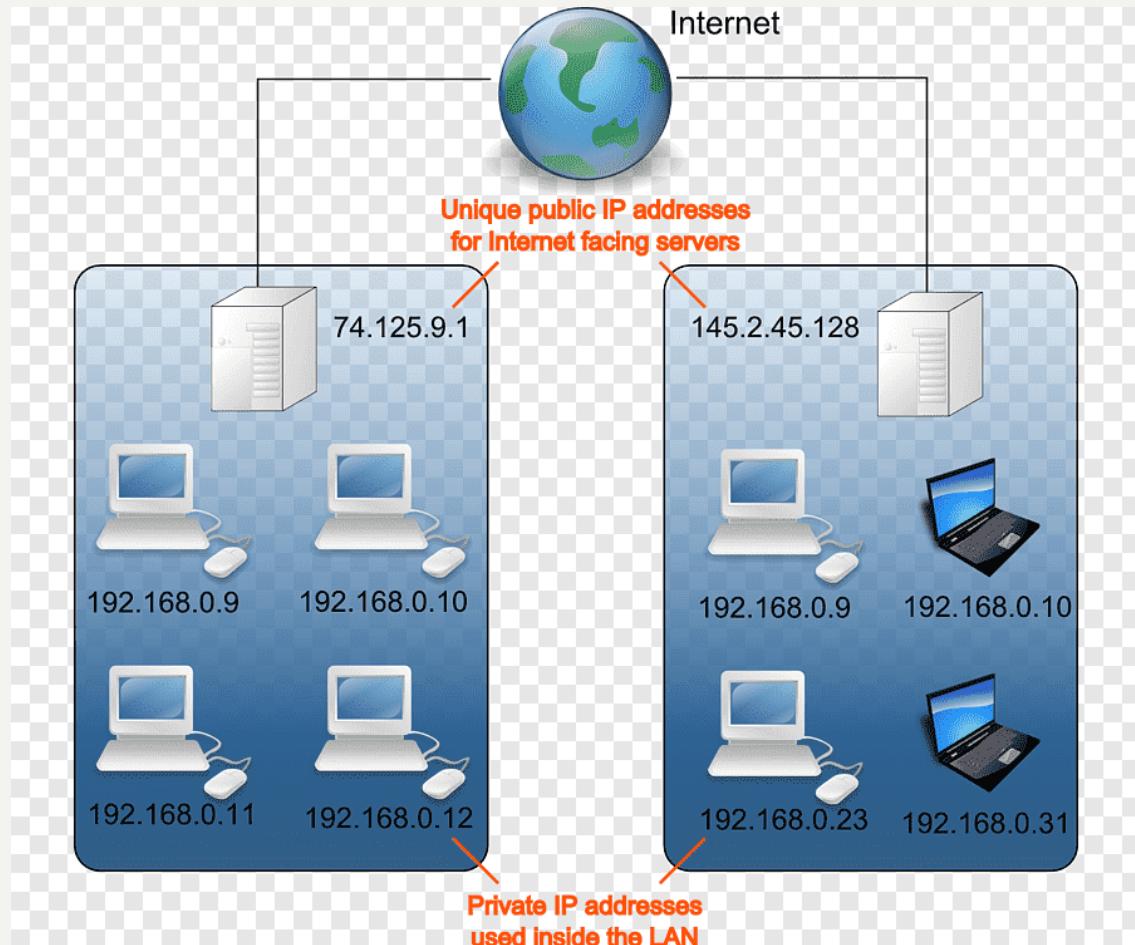
- Una red de información privada es una red segura que solo pueden utilizar ciertos dispositivos o usuarios, y que está aislada de la red pública de internet.
- Las redes privadas se utilizan para conectar equipos de una organización (intranet) o de varias organizaciones (extranet). Son muy utilizadas en entornos empresariales, ya que permiten compartir recursos entre los usuarios conectados a ella
- La mayoría de nosotros estamos conectados a redes privadas
 - *Es posible que no sepamos que lo estamos.*

REDES DE INFORMACIÓN PRIVADAS

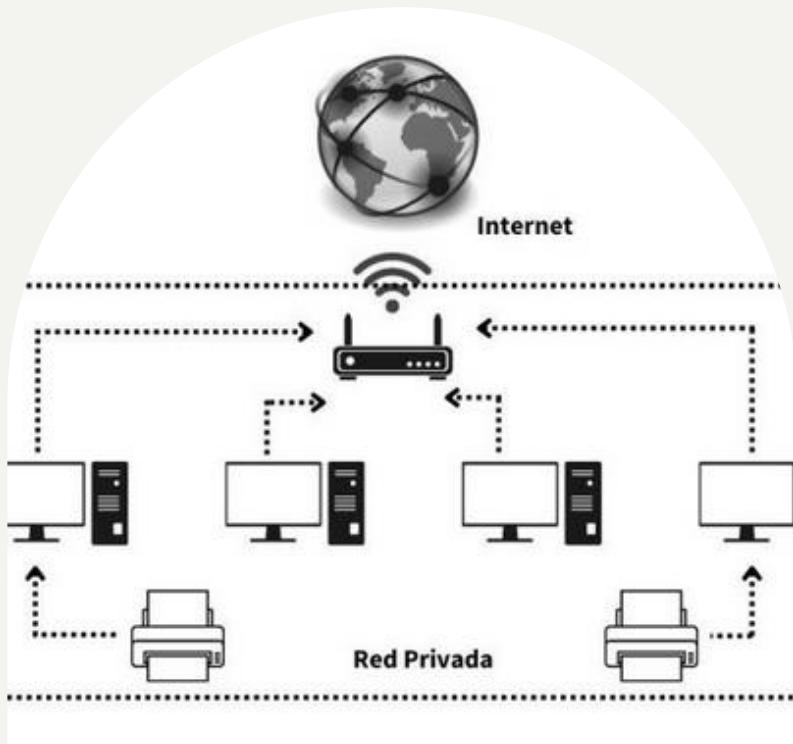


- ¿Qué son las redes privadas?
 - Las redes privadas son redes formadas por varios dispositivos conectados entre sí, pero que no están conectados a una red pública (como es internet).
 - Una red privada es una red operada y gestionada por un administrador encargado de configurar, mantener y gestionar la red, su seguridad y el acceso a la misma.
 - Una red privada es una red a la que pueden conectarse varios equipos computacionales o dispositivos electrónicos a través de un router, que puede o no estar conectado a internet a través de una puerta enlace.

REDES DE INFORMACIÓN PRIVADA



REDES DE INFORMACIÓN PRIVADAS



- Red privada en casa
 - Nosotros somos el administrador de la misma
 - Se conectan, a través del router, diferentes dispositivos (uno o más ordenadores, móviles, Smart TV, una impresora, la consola, etc.)
- Red privada de una empresa
 - A la que están conectados varios equipos, incluso de manera remota, gracias a una VPN.
- Los dispositivos conectados a una red privada tienen asignada una IP (Homologada o no Homologada)
 - Generalmente se conectan a Internet a través de una puerta de enlace (Gateway)
- Los archivos e información de los dispositivos de la red privada no son visibles en la red pública
 - Son vulnerables

REDES DE INFORMACIÓN PRIVADAS

¿Cómo funcionan las redes privadas?

En la actualidad conexión a un punto de interconexión

- Modem/Router
- Switch
- Access Point

De manera que podrán comunicarse entre ellos

- Intercambiando archivos
- Usar recursos, etc.

Al conectarse a la red privada, cada dispositivo recibe una dirección IP privada

Si el router no está conectado a internet

- La red será de área local (LAN) y no tendrá acceso a internet.

Si el router tiene una puerta de enlace a internet, los dispositivos conectados a esa red privada podrán acceder a internet a través de su ISP.

Si bien, los dispositivos de la red privada pueden acceder a internet nadie que no esté conectado a la red privada podrá acceder a los dispositivos conectados a ella (salvo que logre vulnerar la seguridad de la red privada).

Dentro de la red privada, es posible dejar como no visibles los dispositivos conectados a la misma, de manera que los diferentes usuarios de la red no pueden verlos o tratar de acceder a ellos.

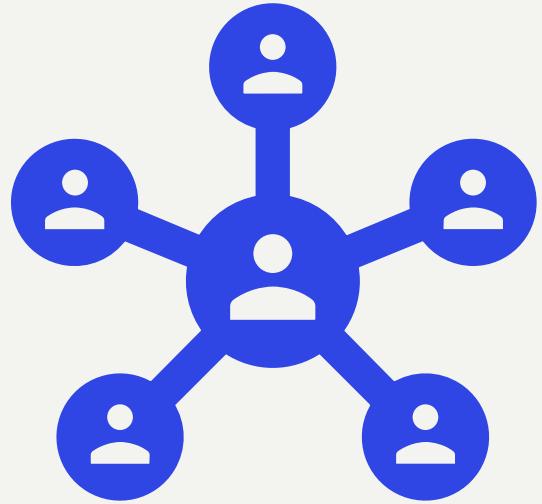
REDES DE INFORMACIÓN PRIVADAS

- Así, las principales ventajas de las redes privadas son:
 - Mayor seguridad, ya que se trata de una red cerrada, a la que, en principio, no se puede acceder sin estar conectados a ella o sin los permisos otorgados por el administrador.
 - Mayor control de los administradores de la red, que pueden dar o quitar privilegios a los usuarios en función de sus roles o tareas.
 - Mayor tasa de transferencia de datos entre los dispositivos interconectados a la red privada.
 - Permite compartir determinados recursos, como las impresoras o las bases de datos, desde diferentes dispositivos.
 - Si está conectado a internet, permite el acceso remoto a la misma, normalmente a través de una VPN para seguir manteniendo los niveles de seguridad y privacidad necesarios.

REDES DE INFORMACIÓN PRIVADAS

- La red privada estará gestionada por un administrador y tendrá varios usuarios, si bien su número será limitado (por ejemplo, en una empresa, el número de empleados con acceso a un equipo informático). El administrador es quien puede incluir nuevos usuarios y gestionar la seguridad de la red.
- En cuanto a las redes privadas virtuales (VPN), estas tienen como particularidad que los dispositivos de la misma se conectan entre sí a través de una red pública. Es decir, que usando un servicio de VPN, podemos crear una red privada «dentro» de la red pública, para conectar de manera remota diferentes dispositivos, que se comunicarán entre ellos como si estuvieran en una red local.
- Cabe señalar que no debemos confundir la navegación privada con el uso de redes privadas, puesto que la primera requiere del acceso a internet que una red privada puede o no tener, pero que por el hecho de estar conectados a una red privada conectada a su vez a una red pública, la navegación no será privada (esto es algo que debemos activar en el navegador).

REDES PRIVADAS VIRTUALES



- Algunos **ejemplos** **redes** **privadas** **son**:
 - Una *red privada doméstica*, es decir, aquella que creas al conectar todos los dispositivos electrónicos al router.
 - La *red privada de una empresa*, o *red interna*, a la que están conectados ordenadores, impresoras y, en algunos casos, dispositivos móviles. Esta red puede limitarse a una sola localización o interconectar varias de ellas.
 - La *red privada* que se puede habilitar en convenciones y eventos, muchas veces a través de WiFi (WLAN).
 - La *red privada* o *intranet* de una universidad.