Redes de computadores

Transparencias de la asignatura

Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Curso 2024–2025

Este documento se realiza gracias al apoyo institucional de la Convocatoria competitiva de Proyectos de Innovación de la Universidad de Zaragoza (PI_DTOST) en el año 2023 y con referencia 4655 con título «Curso OCW para la asignatura Redes de computadores». Concretamente, el apoyo institucional ha consistido en una financiación de 0 €.

Índice general

Redes de Computadores

Presentación de la asignatura

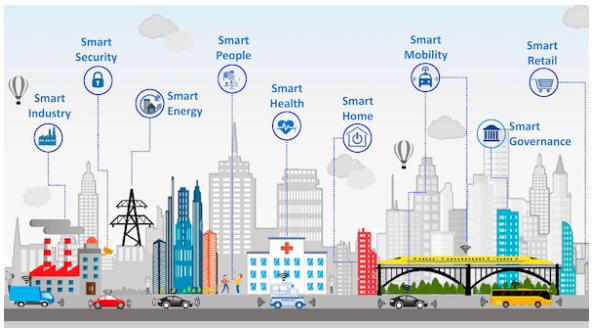
Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción

- Internet lleva 50 años funcionando ininterrumpidamente, adaptándose a cambios tecnológicos y funcionales
- Todo está conectado y lo va a estar aún más



1.1 Contexto de las redes

Actualmente, las redes de computadores son el nexo de unión de prácticamente todas las ramas de la informática

- Expectativas de trabajo prometedoras
  - Cualquier empresa necesita trabajar en red
  - Cualquier empresa tecnológica necesita que sus productos funcionen en red
  - Empresas especializadas en redes (Telefónica, Google, etc.) necesitan expertos
  - …y que todo siga funcionando durante la transición a la nueva versión de Internet (IPv6)
- Los expertos son muy valorados
  - Programador de aplicaciones y servicios
  - Diseñador/administrador de redes
  - Seguridad informática
  - etc.

2 Objetivos y contenido

Conocer las tecnologías y la arquitectura software que permiten construir una red de computadores de forma eficiente:

- Introducción y arquitecturas de red ..... Tema 1
- Fundamentos de las comunicaciones digitales ..... Tema 2
- Comunicación punto-a-punto ..... Tema 3
- Interconexión de redes ..... Tema 4
- Comunicación extremo-a-extremo ..... Tema 5
- Calidad de servicio y control de congestión ..... Tema 6
- Servicios específicos de aplicación ..... Tema 7

2 Objetivos y contenido (II)

Prácticas de laboratorio y trabajo práctico:

- Análisis de tráfico en redes ..... Práctica 1
- Programación de aplicaciones en red ..... Prácticas 2 y 3
- Programación de protocolos ..... Práctica 4 y trabajo
- Topología y organización de Internet ..... Práctica 5
- Administración básica de redes ..... Práctica 6

2 Objetivos y contenido (III)

Propósito de los materiales:

Diapositivas de teoría: guía/resumen de elementos teóricos del curso

Colección de ejercicios: enunciados de ejercicios donde aplicar los elementos teóricos aprendidos

Enunciados de prácticas y trabajo: actividades prácticas en las que aplicar los elementos teóricos aprendidos

3 Profesores

- Juan Segarra Flor
- Jesús Alastruey Benedé

4 Bibliografía

- Larry L. Peterson y Bruce S. Davie. *Computer Networks, a systems approach.* Morgan Kaufmann. https://book.systemsapproach.org
- James F. Kurose y Keith W. Ross. *Computer Networking, a top-down approach.* Pearson.
- Andrew S. Tanenbaum, David J. Wetherall. *Computer Networks.* Pearson.
- William Stallings. *Comunicaciones y Redes de Computadores.* Prentice Hall.
- W. Richard Stevens, Bill Fenner, Andrew M. Rudoff. *UNIX Network Programming Volume 1: The Sockets Networking API.* Addison Wesley.

Créditos de material reutilizado

Imagen «Ciudad inteligente» (p. 3): Ricardo Estévez, www.ecointeligencia.com

Redes de Computadores
Sistemas Operativos

Curso rápido de C

Juan Segarra

(parcialmente basado en material de, en orden alfabético,
N. Ayuso, J.L. Briz, P. Ibáñez, T. Monreal y J. Segarra)

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

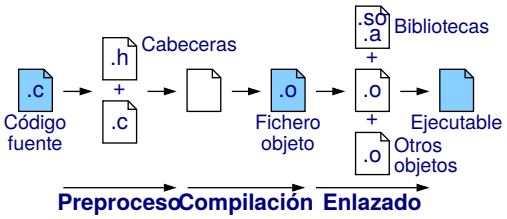
1 Introducción

- Lenguaje imperativo desarrollado junto con UNIX
- Muy ligado al Sistema Operativo
  - Todas las funciones del SO accesibles desde C
  - Todos los SO proporcionan compilador (`$ cc`) y manual (`$ man`) de C
- En este curso rápido asumimos:
  - Nivel básico de programación imperativa
  - Conocimientos básicos de C/C++
    (sintaxis, estructuras de control y operadores)

2 Compilación

Compilador: programa (gcc, clang, icc…) que traduce de alto nivel a código máquina

1. Fase de **preproceso**: modifica texto en código fuente
2. Fase de **compilación**: traduce a código máquina (específico para el procesador de ese equipo)
3. Fase de **enlazado**: genera un ejecutable con lo compilado más las funciones de biblioteca necesarias



2 Compilación (II)

- Compilación básica:
  `$ cc fichero.c` → `a.out` (ejecutable)
- Compilación con destino (outfile):
  `$ cc -o fich fich.c` → `fich` (ejecutable)
- Compilación de múltiples fuentes:
  `$ cc -o fich fichero1.c fichero2.c` → `fich` (ejec.)
- Compilación por pasos:
  `$ cc -c fich1.c` → `fich1.o` (compilado, no ejecutable)
  `$ cc -c fich2.c` → `fich2.o` (compilado, no ejecutable)
  `$ cc -o fich fich1.o fich2.o` → `fich` (ejecutable)
  - Último paso, con fuentes ya compilados, realiza enlazado
  - Evita compilar todo cuando sólo se modifica una parte
  - Automatizado en fichero `Makefile` (`$ make` / `$ gmake`)

2 Compilación (III)

Opciones de compilación interesantes:

`-Wall/-Wextra`: muestra más warnings

`-S`: genera ensamblador, no código máquina

`-l<biblio>`: incluye la biblioteca `<biblio>` al enlazar

`-O`: optimiza según nivel (-O0,1,2,3,s)

`-g`: incluye información para depurar

`-pg`, `--coverage`, etc.: al ejecutar el programa compilado, generará datos de rendimiento (veces que ejecuta cada instrucción, tiempo que pasa en cada subrutina, etc.), legibles desde otros programas (e.g. `$ gprof`, `$ gcov`)

`-fsanitize=<grupo>`: incluye código para detectar errores, según `<grupo>` (e.g. `-fsanitize=address` detecta accesos fuera de rango)

3 Estructura de programa

Elementos en el código fuente:

- Inclusión de ficheros (# preprocesador)
  - Estándar entre <> (`#include <stdio.h>`)
  - Propios entre ¨ ¨ (`#include "misdefiniciones.h"`)
- Definición de constantes (# preprocesador)
  - `#define FALSO 0`
- Definición de nuevos tipos de datos (`typedef`)
- Cabecera/definición de funciones
- Declaración de funciones (todas al mismo nivel)
- Función `main()` como programa principal

Programas divididos en programa principal y módulos (.h + .c)

- Ficheros .h (*header*) con definiciones
- Ficheros .c (*c code*) con declaraciones

3.1 Entrada/salida básica

```
printf("formato",variables);
scanf("formato",variables);
getchar();
putchar(char);
```

Elemento	Formato
carácter	%c
decimal	%d
dec. sin signo	%u
real	%f
string	%s
salto	\n
tabulador	\t
otros	…

(En C++: `cin>>` y `cout<<`)

```
#include <stdio.h>

char v[6]="LUNES";
int i=245;
float r=3.1416;

printf("%c\n",v[3]); // E
printf("%d\n",i); // 245
printf("%d\n",v[3]); // 69
printf("%x\n",i); // f5
printf("%f\n",r); // 3.1416
printf("%s\n",v); // LUNES
```

3.2 Ejemplo

```c
/* fichero prog.c */
#include <stdio.h>

#define CONST 3
#define ELEM_IDENTIDAD_SUMA 0

int suma(int a, int b);

main() {
  int i=7, j=CONST, k=CONST; // resultado
  if (i != ELEM_IDENTIDAD_SUMA) { k=suma(i,j); }
  printf("%d+%d=%d",i,j,k);
}

int suma(int a, int b) {
  int aux;
  aux=a+b;
  return aux;
}
```

```
$ cc prog.c -o prog
```


3.2 Ejemplo (II)

```c
/* fichero prog.c */
#include <stdio.h>

#define CONST 3
#define ELEM_IDENTIDAD_SUMA 0

int suma(int a, int b);

main() {
  int i=7, j=CONST;
  int k=CONST; // resultado

  if (i!=ELEM_IDENTIDAD_SUMA) {
    k=suma(i,j);
  }
  printf("%d+%d=%d",i,j,k);
}
```

Separación de función «suma» del programa principal

```c
/* fichero suma.c */
int suma(int a, int b) {
  int aux;
  aux=a+b;
  return aux;
}
```

```
$ cc -c suma.c
$ cc -c prog.c
$ cc -o prog prog.o suma.o
```


3.2 Ejemplo (III)

Módulo «suma» como .c/.h

```c
/* fichero prog.c */
#include <stdio.h>
#include "suma.h"

#define CONST 3

main() {
   int i=7, j=CONST;
   int k=CONST; // resultado

   if (i!=ELEM_IDENTIDAD_SUMA) {
     k=suma(i,j);
   }
   printf("%d+%d=%d",i,j,k);
}
```

```c
/* fichero suma.h */

#define ELEM_IDENTIDAD_SUMA 0

int suma(int a, int b);
```

```c
/* fichero suma.c */
#include "suma.h"

int suma(int a, int b) {
  int aux;
  aux=a+b;
  return aux;
}
```


3.2 Ejemplo (IV)

Comentarios para la generación automática de documentación

```c
/* fichero suma.h */

#define ELEM_IDENTIDAD_SUMA 0

int suma(int a, int b);
```

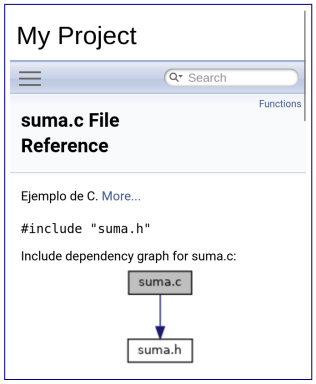
```
$ doxygen -g
$ doxygen
```

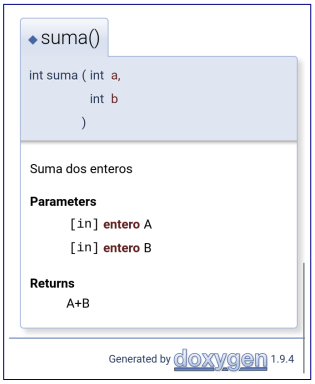
```c
/**
 * @file suma.c suma.h
 * @brief Ejemplo de C
 * @author Profesores UZ
 */

/**
 * Elemento identidad
 */
#define ELEM_IDENTIDAD_SUMA 0

/**
 * Suma dos enteros
 *
 * @param[in] entero A
 * @param[in] entero B
 * @return A+B
 */
int suma(int a, int b);
```


3.2 Ejemplo (V)





4 Tipos de datos

	Tipo	Bits	Declaración	Uso
entero	char	≥ 8	`char c;`	`c=97; c='a';`
	int	≥ 16	`int i;`	`i=023; //oct`
	short (int)	≥ 16	`short int i;`	`i=0x30A; //hex`
	long (int)	≥ 32	`long int i; long j;`	`i=-5L;`
	long long (int)	≥ 64	`long long i;`	
	int$N$_t	N	`int64_t i;`	$N \in \{8, 16, 32, 64\}$
	unsigned uint$N$_t		`unsigned long i;` `uint16_t i;`	$i \in \mathbb{N}$
	size_t		`size_t tam;`	$tam \in \mathbb{N}$
	ssize_t		`ssize_t tam;`	$tam \in \mathbb{Z}$
	puntero	*	`int *p; char *q;`	
real	float	32	`float f;`	`f=-5.3e8;`
	double	64	`double f;`	`f=-5.3;`
	long double	≥ 80	`long double f;`	`f=-5;`

4.1 Conversión de tipos

- Implícita por *widening*:
  `short i=2; long j; j=i;` //rellena con ceros a la izquierda
- Implícita a tipo de operando más genérico:
  `num=3*2.1;` // float 2.1 → mult. float → 3 float
- Implícita por asignación:
  `int i; i=2.8;` // trunca a 2
- Explícita (cast):
  `num= 3.0 + (float)1/2;`
- **¡Cuidado!**
  `num=3.0+1/2;` // división (entera) antes que suma
  `num=3.0+(float)(1/2);` // división antes que cast
  `int i=2.8;` // pérdida de información (trunca)
  `char j=300;` // pérdida de información (módulo)
  `unsigned char c=20; int i=-c; c=-c;` // ¿i==c?

4.2 Atributos de variables

Global: variable **declarada fuera** de funciones (visible desde varias funciones)
- auto (por defecto): visible desde todas
- static: visible sólo desde las del propio fichero
- extern: especifica variables declaradas en otros ficheros

Local: variable **declarada dentro** de una función (visible sólo dentro la función)
- auto (por defecto): se reserva espacio en cada llamada y se libera (pierde su valor) al salir
- static: sólo se inicializa para la primera llamada y mantiene su valor de una llamada a otra

4.2 Atributos de variables (II)

Otros atributos:

register: indicación al compilador para que mapee la variable sobre un registro físico de la máquina (si es posible)

volatile: indicación al compilador para evitar optimizaciones sobre esta variable; se usa para variables cuyo valor puede cambiar de forma ajena al programa

const: indicación al compilador para que dé error de compilación si detecta código que la modifica

4.3 Ejemplo

```
#include <stdio.h>

int i=0;
int j=0;

void incrementar() {
  int j=0; // ojo! hay una j global
  static int k=0;

  i++;
  j++;
  k++;
  printf("i=%d, j=%d, k=%d\n",i,j,k);
}

int main() {
  incrementar(); incrementar(); incrementar();
}
```


5 Vectores

Vector: conjunto de variables del mismo tipo que se referencian por un nombre común

Declaración: `char v[5];` // Declaración de 5 elementos (v[0]...v[4]) que ocupan posiciones consecutivas de memoria

Inicialización: `char v[5]={'a','e','i','o','u'};`

Uso: `v[4]='z';` // Asigno el valor 'z' al **5º** elemento del vector v

Cálculo de posición de memoria:

$$dirBase + índice \cdot tamañoElemento$$

- Nombre de un vector ≡ Dirección de su primer elemento
- Elemento `v[3]` en dirección `v+3*sizeof(char)`
- `char M[2][5];` // 10 elementos consecutivos por filas: `M[0][0], M[0][1]... M[0][4], M[1][0]... M[1][4]`

5.1 Strings

String: vector de caracteres acabado con el carácter `'\0'` (no existe el tipo «string» de C++)

Inicialización: `char v[6] = {'L','U','N','E','S','\0'};`
`char v[6] = {'L','U','N','E','S',0};`
`char v[6] = "LUNES";` // Una constante string es una cadena de caracteres entre comillas dobles

¡Cuidado!

```
char v[6]; v="LUNES"; // Mal: operador = no copia vectores
char v[6]="LUNES"; v[2]=0;          // Bien: "LU" en string v
char v[6]="LUNES"; v[5]='a';  // Ojo: v ya no es un string
char v[6]="LUNES"; v[8]='a';        // Ojo: fuera de rango
```

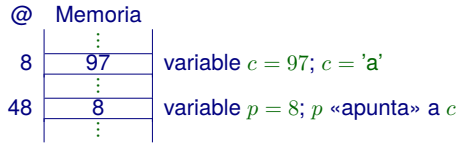

5.2 Punteros

Puntero: variable que contiene la dirección de memoria de una variable de un tipo dado

Declaración: `char *p;` // declaración (sin inicializar) de un puntero a una variable de tipo carácter

Operador `&`: devuelve la dirección de memoria del operando: `char c; p = &c;`

Operador `*`: devuelve el valor almacenado en la dirección especificada: `char c; c=*p;`



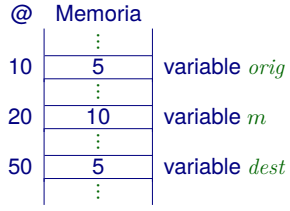
Reserva/liberación de memoria dinámica: `malloc()/free()` (en C++: `new/delete`)

5.3 Ejemplo

```
#include <stdio.h>

main() {
  int dest, orig;
  int *m;

  orig=5;
  m=&orig;
  dest=*m;
  printf("%d %d",orig,dest);
  *m=6;
  printf("%d %d",orig,dest);
  dest=orig;
  printf("%d %d",orig,dest);
}
```



Salida

```
5 5
6 5
6 6
```


5.4 Operaciones con vectores y punteros

Recordar: $dirBase + índice \cdot tamañoElemento$

- Nombre de vector (sin corchetes) ≡ dirección (constante) de su primer elemento ($dirBase$):
  `char v[10]; v ≡ &v[0]`
- Cualquier variable puntero puede indexarse con corchetes como un vector
- Todas las operaciones con vectores y punteros se hacen de acuerdo a su tipo base:
  `int *m; m+1 ≡ m+(1*sizeof(int))`
  `int *m,i; m+i ≡ m+(i*sizeof(int))`
  `int *m; m[3] ≡ *(m+(3*sizeof(int)))`
  `int v[3]={1,5,12}, *p=v;` ¿`*(p+2)`? ¿`*p+2`?

6 Paso de parámetros

Paso por valor: la llamada copia un valor, especificado como parámetro, a una variable dentro de la función

- La modificación de la copia no modifica el original

En cualquier computador, el paso de parámetros es **siempre por valor**

Paso por referencia: paso por valor de la dirección de memoria (puntero) donde reside un dato

- La modificación del contenido de esa dirección modifica el dato referenciado
- Si un parámetro ocupa muchos bytes puede ser recomendable pasarlo por referencia, incluso si no va a modificarse, para evitar copiarlo

6 Paso de parámetros (II)

paso por valor: misma sintaxis en C y C++

```
int incrementa(int var) {
  return var++;
}
```

```
int a=10; a=incrementa(a); a=incrementa(10);
```

paso por referencia en C++: sintaxis abreviada

```
void incrementa(int & var) {
  var++; return;
}
```

```
int a=10; incrementa(a); // incrementa(10) MAL
```

paso por referencia en C: sintaxis literal (punteros)

```
void incrementa(int * var) {
  (*var)++; return;
}
```

```
int a=10; incrementa(&a); // incrementa(&10) MAL
```


7 Structs

struct: estructura de datos compuesta por elementos individuales (llamados campos o miembros) que pueden ser de distinto tipo

Operador = copia

Operador . acceso a campo

Operador -> acceso a campo desde puntero a struct

```
struct fecha {
  int mes;
  int dia;
  int anyo;
};

struct fecha hoy,ayer;
struct fecha *manyana;

hoy.dia=1;
hoy.mes=2;
hoy.anyo=3;

ayer=hoy;
manyana=&hoy;

(*manyana).dia=2;
manyana->dia=2;
```

Redes de Computadores

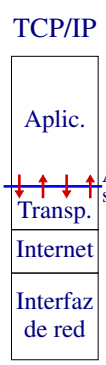
Introducción a la programación con sockets

Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción



Aplicación en red: programa que se comunica con otro mediante el mismo **protocolo de aplicación** a través de una red de computadores

Application Program Interface (API) socket: interfaz de programación entre capa de aplicación y capa de transporte originalmente desarrollado para BSD (*Berkeley Software Distribution*), un derivado de Unix

socket (enchufe/conector): extremo de un canal de comunicación entre dos procesos. Cada **sistema operativo** puede implementar distintas **familias/dominios** de sockets. En la asignatura estudiaremos los dominios **Internet**

2 Tipos de socket en Internet

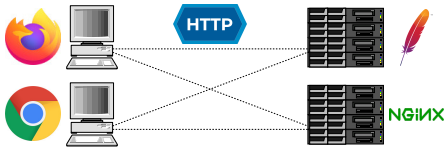
SOCK_STREAM: socket de tipo flujo (*stream*) o TCP
- Transmisión **bidireccional continua**
- **Fiable** (los datos se reciben ordenados, sin errores, sin pérdidas y sin duplicados)
- **Con conexión** (el emisor se pone en contacto con el receptor antes de enviar datos)
- Prot. capa de transporte: **TCP** (*Transmission Control Protocol*)

SOCK_DGRAM: socket de tipo datagrama (*datagram*) o UDP
- Transmisión **bidireccional** de datos de **tamaño limitado**
- **No fiable**
- **Sin conexión**
- Prot. capa de transporte: **UDP** (*User Datagram Protocol*)

Otros: dependiendo del sistema o familia utilizado

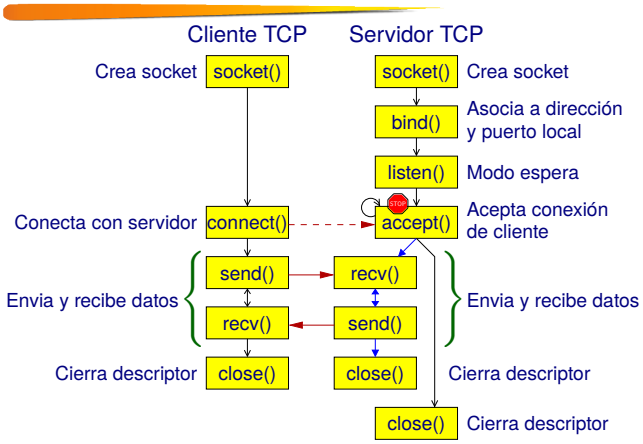
3 Modelo cliente/servidor

- Programación basada en **modelo cliente/servidor**
  - **Cliente**: proceso que inicia la comunicación para solicitar un servicio (e.g. navegador web solicitando una página)
  - **Servidor**: proceso que recibe y atiende solicitudes de varios clientes (e.g. servidor web enviando la página solicitada)

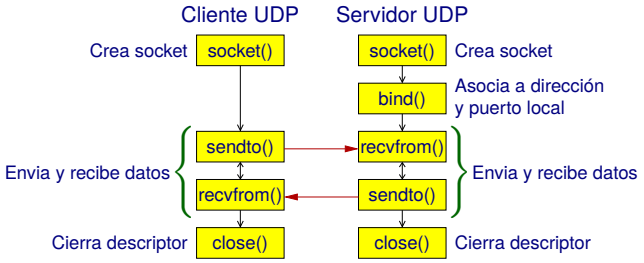


- Dependiendo del tipo de socket utilizado (TCP o UDP), el procedimiento para manejar las funciones varía ligeramente

3.1 Modelo cliente/servidor TCP



3.2 Modelo cliente/servidor UDP



- En UDP no se usan las llamadas de establecimiento de conexión (listen/connect/accept)
- En UDP las llamadas de envío/recepción de datos (sendto/recvfrom) incluyen destinatario/remitente

4 Llamadas al sistema

socket=**socket**(dominio, tipo, protocolo) crea un extremo de la comunicación

err=**bind**(socket, dir_local, long_dir) enlaza socket a la dirección y puerto locales (dir_local)

err=**listen**(socket, long_cola) pone socket en modo de espera y establece su límite máximo de conexiones en espera

err=**connect**(socket, dir_remota, long_dir) inicia una conexión en socket con dir_remota, que debe estar en modo de espera

sock2=**accept**(socket, ↑ dir_remota, ↕ long_dir) crea una conexión (sock2) a partir de una petición (**connect**) a socket (modo espera) e informa sobre el cliente (dir_remota)

núm=**send**(socket, mensaje, long_msj, flags) envía mensaje a través de socket

4 Llamadas al sistema (II)

núm=**recv**(socket, mensaje, long_msj, flags) recibe mensaje a través de socket

núm=**sendto**(socket, mensaje, long_msj, flg, dir_remota, long_dir) envía mensaje a dir_remota a través de socket

núm=**recvfrom**(socket, msj, long_msj, flg, ↑ dir_remota, ↕ long_dir) recibe mensaje e informa sobre el remitente (dir_remota)

err=**shutdown**(socket, sentido) cierra socket en uno o ambos sentidos; automático al llamar a **close** (obligatorio)

.....

núm=**read**(descriptor, mensaje, long_msj) lee mensaje de descriptor (socket/fichero)

núm=**write**(descriptor, mensaje, long_msj) escribe mensaje en descriptor (socket/fichero)

err=**close**(descriptor) cierra descriptor (socket/fichero)

4.1 Llamadas y funciones auxiliares

htons/htonl/ntohs/ntohl convierte (⋆**to**⋆) enteros cortos/largos (**s**hort/**l**ong, 16/32 b) entre formato local y red (**h**ost y **n**etwork)

getaddrinfo traduce servicios y direcciones de red

freeaddrinfo libera mem. dinámica de estructuras de dirección

inet_ntop/inet_pton analiza/crea estructuras de dirección de red

getsockopt/setsockopt obtiene/establece opciones en sockets

select bloquea el proceso hasta poder realizar un read/write sin bloqueo sobre alguno de los descriptores especificados

5 Estructuras de direcciones

- Las direcciones se guardan en un `struct addrinfo`

```
struct addrinfo {
 int      ai_flags;          //AI_PASSIVE,AI_CANONNAME,etc
 int      ai_family;         //AF_INET,AF_INET6,AF_UNSPEC
 int      ai_socktype;       //SOCK_STREAM, SOCK_DGRAM
 int      ai_protocol;       //use 0 for "any"
 size_t   ai_addrlen;        //size of ai_addr in bytes
 struct sockaddr *ai_addr;   //struct sockaddr_in or _in6
 char    *ai_canonname;      //full canonical hostname

 struct addrinfo *ai_next;   //linked list, next node
};
```

- El formato y tamaño de las direcciones (`struct sockaddr *ai_addr`) depende de la familia/dominio

5 Estructuras de direcciones (II)

- Estructura de direcciones para el protocolo IPv4 de Internet (familia `AF_INET/PF_INET`)

```
struct sockaddr_in {
 short          sin_family;   // AF_INET
 unsigned short sin_port;     // e.g. htons(3490)
 struct in_addr sin_addr;     // see struct in_addr below
 char           sin_zero[8];  // zero this if you want to
};
```

```
struct in_addr {
 unsigned long s_addr;        // 32bit X.X.X.X
};
```


5 Estructuras de direcciones (III)

- Estructura de direcciones para el protocolo IPv6 de Internet (familia `AF_INET6/PF_INET6`)

```
struct sockaddr_in6 {
 u_int16_t       sin6_family;   // addr family AF_INET6
 u_int16_t       sin6_port;   // port, Network Byte Order
 u_int32_t       sin6_flowinfo; // IPv6 flow information
 struct in6_addr sin6_addr;     // IPv6 address
 u_int32_t       sin6_scope_id; // Scope ID
};
```

```
struct in6_addr {
 unsigned char   s6_addr[16];  // 128bit X:X:X:X:X:X:X:X
};
```


5 Estructuras de direcciones (IV)

- Estructura de direcciones para cuando se desconoce la familia específica (reserva el tamaño suficiente)

```
struct sockaddr_storage {
    sa_family_t  ss_family;     // address family

//all this is implementation specific padding, ignore it
    char       __ss_pad1[_SS_PAD1SIZE];
    int64_t    __ss_align;
    char       __ss_pad2[_SS_PAD2SIZE];
};
```

- Las estructuras de direcciones (`sockaddr`, `sockaddr_in`, `sockaddr_in6` y `sockaddr_storage`) son intercambiables
- Cualquier función con alguna de las estructuras anteriores como parámetro sabrá tratar cualquiera de ellas

6 Cambio explícito de tipos

Ejemplo con parámetro tipo `struct sockaddr` de salida:
`int accept(int s, struct sockaddr *addr, socklen_t *addrlen);`

- Si esperamos sólo direcciones IPv4:
  `struct sockaddr_in *addr_ipv4;`
  `accept(sock, addr_ipv4, &len)`
- Si esperamos sólo direcciones IPv6:
  `struct sockaddr_in6 *addr_ipv6;`
  `accept(sock, addr_ipv6, &len)`
- Si esperamos direcciones IPv4 e IPv6:
  `struct sockaddr_storage addr_ip;`
  `accept(sock, addr_ip, &len)`

Sólo lo vamos a usar **sobre punteros** para evitar sus *warnings*

7 Serialización de datos

- En TCP/IP no hay capa de presentación de datos
- El programador decide el formato de intercambio y **debe garantizar** que ambos extremos lo interpretan igual
- send/recv transmiten **bytes contiguos en memoria**
  Tipos básicos: ya almacenados en bytes contiguos
  Enteros: htons/htonl/ntohs/ntohl
  Estructuras de datos dispersos en memoria: programar funciones para recorrer y enviar (o recibir y generar) la estructura (listas, tablas *hash*, ¿structs?, etc.)

7 Serialización de datos (II)

Ejemplo de envío (tipo `void` no da *warnings*):

```
ssize_t send(int s, const void *msg, size_t len, int
flags);
```

- Si queremos enviar una cadena/vector:

```
char cadena[10];
send(s,        , sizeof(      ), flags);
```

- Si queremos enviar un entero largo, e.g. 20:

```
long numero;

send(s,          , sizeof(      ), flags);
```

- Si queremos enviar un `struct` con datos **contiguos** y **correctamente formateados**:

```
struct mensaje mistruct;
send(s,            , sizeof(        ), flags);
```

Redes de Computadores

Tema 1 – Introducción y arquitecturas de red

Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción

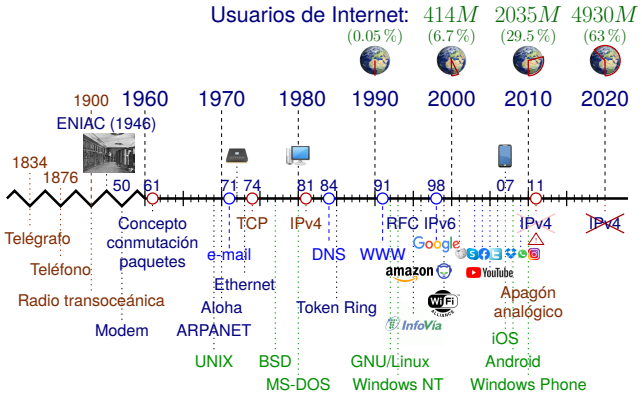

Paloma mensajera


Líneas telégrafo


Ordenador conectado

- Las **redes de comunicaciones** permiten superar limitaciones geográficas «rápido»
  - Servicios: reservas, televenta, banca telefónica, etc.
- Las **redes de computadores**, al prescindir de emisor/receptor humanos, además permiten:
  - Información voluminosa bidireccional → más servicios
  - Supercomputación (cluster/grid)
  - Independencia del equipo de trabajo (nube/cloud)
  - Trabajo en grupo / redes sociales
  - etc.

1.1 Evolución histórica



2 Terminología

Red de computadores: sistema de comunicación que permite intercambiar información entre equipos informáticos

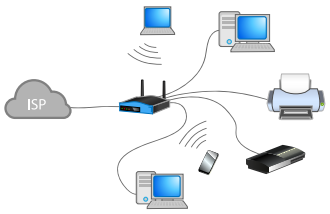
Enlace: conexión física o lógica entre dispositivos

- Enlace **punto-a-punto** (*point-to-point*): **conexión física** directa entre dos o más dispositivos, e.g. cable, aire
- Enlace **extremo-a-extremo** (*end-to-end*): **conexión lógica** entre dos dispositivos, normalmente a través de dispositivos de interconexión

Nodo: dispositivo conectado capaz de enviar/recibir

- Dispositivo de interconexión: nodo que retransmite la información recibida, e.g. encaminador (*router*)
- *Host*/estación: nodo que hospeda aplicaciones/servicios
  - Cliente: estación extremo que solicita servicios
  - Servidor: estación extremo que proporciona servicios
  - *Peer* (par): cliente + servidor

2 Terminología (II)



Ejemplos:

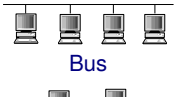
- Enlace punto-a-punto entre portátil y encaminador
- Enlace extremo-a-extremo entre móvil (cliente) e impresora (servidor) al imprimir a través del encaminador
- Enlace extremo-a-extremo entre portátil (cliente) y encaminador (servidor) al configurar encaminador desde portátil

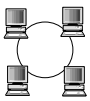
2.1 Tipos de red, por extensión

Tipo de red	Distancia	Cobertura
Área personal (PAN)	1 m	Escritorio/Persona
Área local (LAN)	10 m	Habitación
	100 m	Edificio
	1000 m	Complejo/Campus
Área metropolitana (MAN)	10 km	Ciudad
Área amplia (WAN)	100 km	Región
	1000 km	País
	10000 km	Continente
Internet	>10000 km	Tierra+ISS

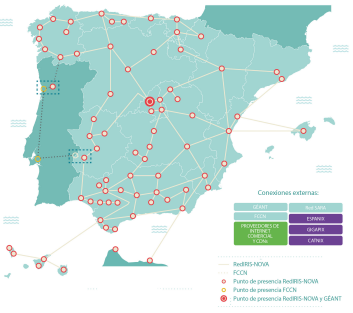
2.2 Topología de red

Topología de red: disposición en que se encuentran los elementos de la red. Ejemplos:


Bus


Anillo


Estrella/Árbol


Topología de RedIRIS

2.3 Tipos de envío, por destino

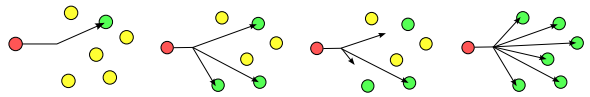
Dependiendo del destino, un mensaje puede ser:

Unicast: un único destino

Multicast/Multidestino: un conjunto de destinos

Anycast: un destino cualquiera (el más cercano) de un conjunto

Broadcast/Difusión: todos los destinos



Unicast Multicast Anycast Broadcast

No todos los tipos de envío son siempre posibles

2.4 Unidades y prefijos

Bit (bit o b): binary digit (1/0) Byte (B): vector de 8 bits

Prefijo decimal (SI)				P. bin. (ISO/IEC)		
Valor	**Prefijo**	**Valor**	**Prefijo**	**Valor**	**Prefijo**	**Dif.**
10^{-3}	mili (m)	10^{3}	kilo (k)	2^{10}	kibi (Ki)	2.4 %
10^{-6}	micro (μ)	10^{6}	mega (M)	2^{20}	mebi (Mi)	4.9 %
10^{-9}	nano (n)	10^{9}	giga (G)	2^{30}	gibi (Gi)	7.4 %
10^{-12}	pico (p)	10^{12}	tera (T)	2^{40}	tebi (Ti)	10.0 %
10^{-15}	femto (f)	10^{15}	peta (P)	2^{50}	pebi (Pi)	12.6 %
10^{-18}	atto (a)	10^{18}	exa (E)	2^{60}	exbi (Ei)	15.3 %
10^{-21}	zepto (z)	10^{21}	zetta (Z)	2^{70}	zebi (Zi)	18.1 %
10^{-24}	yocto (y)	10^{24}	yotta (Y)	2^{80}	yobi (Yi)	20.9 %
10^{-27}	ronto (y)	10^{27}	ronna (R)	2^{90}		
10^{-30}	quecto (y)	10^{30}	quetta (Q)	2^{100}		

3 Estándares

- Al principio cada fabricante tenía especificaciones propias
  - E.g. SNA (IBM), IPX/SPX (Novell), Appletalk (Apple)
  - **Problema**: interoperatividad limitada entre fabricantes
- **Solución**: establecer especificaciones públicas entre todos, aprobadas por organismos internacionales, que todos puedan seguir
- ¿Por qué siguen existiendo especificaciones privadas?
  - Forzar la compra de productos del mismo fabricante (*SMB*)
  - Retener a los usuarios para vender la información que generan (*Whatsapp*)
  - Coste de salida elevado: cambiar todos los equipos a la vez, aislarse de los contactos…

3 Estándares (II)

Principales organizaciones de estándares:

ISOC: *Internet Society*
- IAB: *Internet Architecture Board*
- IETF: *Internet Engineering Task Force*
  - Request For Comments (RFCs)
- IRTF: *Internet Research Task Force*
- IESG: *Internet Engineering Steering Group*

ANSI: *American National Standards Institute*

IEEE: *Institute of Electrical and Electronics Engineers*

ISO: *International Organization for Standardization*

ITU-T: *International Telecommunication Union - Telecommunications Sector*

W3C: *World Wide Web Consortium*

3 Estándares (III)

Algunas versiones dentro del estándar IEEE 802.11 (Wi-Fi):

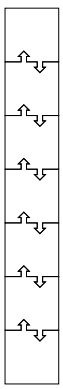
802.11a: 54 Mbit/s, 5 GHz (1999-2001)
802.11b: 5.5 y 11 Mbit/s, 2.4 GHz (1999)
802.11d: International roaming extensions (2001)
802.11e: QoS, including packet bursting (2005)
802.11g: 54 Mbit/s, 2.4 GHz standard (b-compatible) (2003)
802.11i: Enhanced security (2004)
802.11n: Wi-Fi 4, MIMO (2009)
802.11ac: Wi-Fi 5, 256-QAM (2013)
802.11ad: Very High Throughput 60 GHz (2012)
802.11ai: Fast Initial Link Setup, secure link in <100 ms (2016)
802.11ax: Wi-Fi 6, OFDMA, 1024-QAM (2020)
802.11be: Wi-Fi 7, 4096-QAM (borrador 2021)

4 Arquitectura de red

Arquitectura de red: patrón común al que han de ceñirse los elementos de red para mantener compatibilidad entre sí

- En la interconexión de computadores intervienen muchos elementos hardware/software de distintos fabricantes
- Especificar detalladamente todo el problema de forma conjunta no es factible
  - Un «firefox» específico para cada tipo de tarjeta de red
- Mejor dividir el problema mediante un **modelo de capas**
  - Permite describir el funcionamiento de las redes de forma modular y hacer cambios de manera sencilla
  - Modelo de referencia: *Open System Interconnection* (OSI) de ISO

4 Arquitectura de red (II)



El modelo de capas se basa en:
- Cada capa resuelve un problema concreto
- Un mismo problema puede resolverse de distintas formas, detalladas en protocolos
  - Una misma capa puede albergar varios protocolos
  - Dos capas en sistemas distintos se pueden comunicar si usan el mismo protocolo
- Cada capa/protocolo tiene una interfaz de comunicación con sus capas superior e inferior (e.g. *API socket*)

Pila de protocolos: conjunto de protocolos (uno por capa) usados en una comunicación concreta

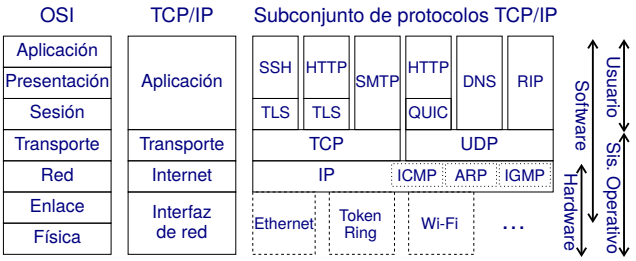
4.1 Modelo OSI

Capa	Descripción
7. Aplicación	Protocolos específicos para aplicaciones
6. Presentación	Conversión entre formatos de datos
5. Sesión	Control/coordinación de comunicaciones
4. Transporte	Comunicación extremo-a-extremo
3. Red	Búsqueda de caminos + llevar mensajes a destino
2. Enlace de datos	Control de acceso al medio de transmisión (MAC) + comunicación punto-a-punto
1. Física	Uso del medio de transmisión (parámetros mecánicos/eléctricos/funcionales)

4.2 Modelo TCP/IP

- Anterior (1972) al modelo OSI (1984)
- Diseñado por el departamento de defensa de los EE.UU. (ARPANET)
- Objetivo: proporcionar comunicaciones con tolerancia a fallos (conmutación de paquetes)
- Diseñado sin las perspectivas de su uso actual (Internet)
- Comunicación entre procesos
- Capa de interfaz de red aglutina las capas física y de enlace de datos
- Capa de aplicación aglutina las capas por encima de la capa de transporte

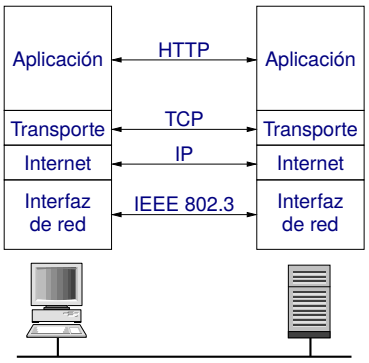
4.3 Comparativa OSI-TCP/IP



- A menudo se sigue un modelo híbrido entre ambos

4.3 Comparativa OSI-TCP/IP (II)

- E.g. acceso a un servidor web en la misma LAN



4.4 Encapsulado de protocolos

- Para enviar una carta hace falta un sobre
  - Dirección destino, destinatario, remitente, sellos (limitan la distancia), acuse de recibo, etiquetas (e.g. frágil), etc.

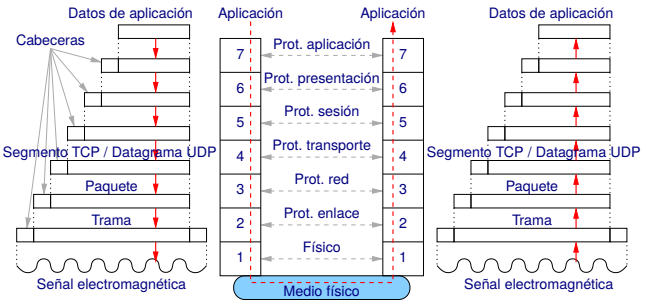
Mensaje: secuencia de bytes que conforma la unidad de información de un protocolo
Cabecera (*header*): información de control
Datos (cuerpo/*payload*): información para el destinatario
Cola (*footer*/*trailer*): control adicional (en pocos protocolos)

Byte 1	Byte n
Cabecera	Datos para el destinatario

4.4 Encapsulado de protocolos (II)

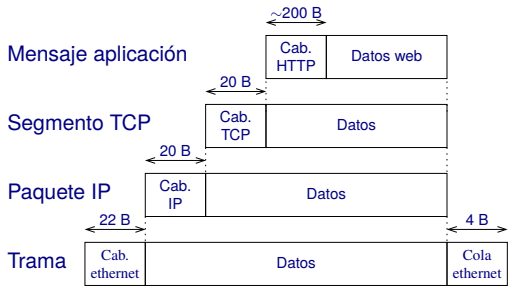
- **Encapsulado**: la «cabecera+datos» en una capa son los «datos» en la capa inferior

4.4 Encapsulado de protocolos (III)



4.4 Encapsulado de protocolos (IV)

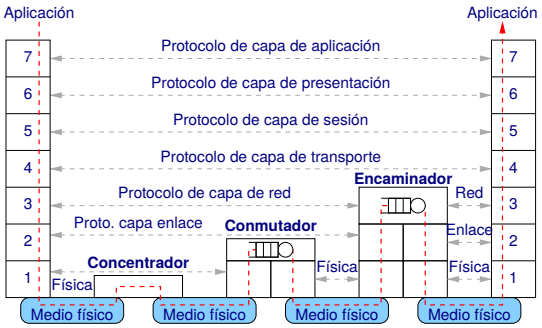
- La información de control añadida reduce la eficiencia



Eficiencia: bytes datos / bytes mensaje
Sobrecarga: bytes control / bytes mensaje ($= 1 -$ Eficiencia)

5 Dispositivos de interconexión

- Física: amplificador, repetidor, concentrador (*hub*) ¡transp.!
- Enlace: conmutador (*switch*), puente (*bridge*) ¡transparente!
- Red: encaminador (*router*)

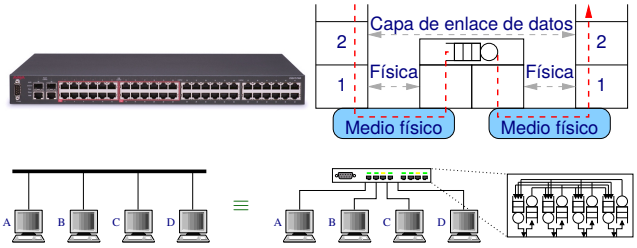


5.1 …en capa física

- Amplificador, repetidor, concentrador (*hub*)
- Función principal: extender rango
- Transparente: los equipos no saben si hay repetidor o no

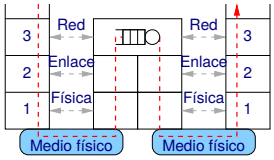
5.2 …en capa de enlace

- Conmutador (*switch*), puente (*bridge*)
- Función principal: reducir colisiones en una red
- Transparente: los equipos no saben si hay conmutador o no



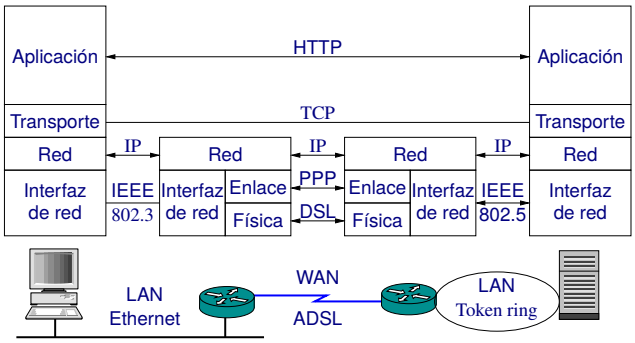
5.3 …en capa de red

- Encaminador (*router*)
- Función principal: interconectar redes
- No transparente: los equipos conocen a su encaminador e interactuan explícitamente con él



5.3 …en capa de red (II)

- E.g. acceso web atravesando tres redes físicas



Créditos de material reutilizado

Imagen «Paloma mensajera» (p. 3): CC BY Wakana Sasaki, DataBase Center for Life Science (DBCLS)
Imagen «Líneas telégrafo» (p. 3): PD Internet Archive Book Images
Imagen «Ordenador conectado» (p. 3): CC BY Someone's Moving Castle
Imagen «CPU, PC (iconos Oxygen)» (p. 4): GNU LGPL, The Oxygen Team, KDE
Imagen «Smartphone (iconos Tango)» (p. 4): CC BY SA Rickterto
Imagen «ENIAC» (p. 4): PD U.S. Army
Imagen «Logo Google 2015» (p. 4): PD Google
Imagen «Logo Wikipedia» (p. 4): CC BY SA Nohat
Imagen «Logo Amazon» (p. 4): PD Amazon.com, Inc.
Imagen «Icono Napster» (p. 4): CC BY SA Dj tronick
Imagen «Logo Skype 2019» (p. 4): PD Skype Technologies
Imagen «Logo Facebook 2019» (p. 4): PD Facebook, Inc.
Imagen «Icono Twitter» (p. 4): CC BY David Ferreira
Imagen «Icono Dropbox» (p. 4): PD Dropbox
Imagen «Logo Whatsapp» (p. 4): PD WhatsApp
Imagen «Icono Instagram» (p. 4): CC BY SA Instagram

Créditos de material reutilizado (II)

Imagen «Logo YouTube 2017» (p. 4): PD YouTube
Imagen «Logo Wi-fi alliance» (p. 4): PD Wi-fi Alliance
Imagen «Logo InfoVía» (p. 4): PD Desconocido
Imagen «Red doméstica» (p. 6): CC BY SA Milesjpool (etiquetas y bordes eliminados)
Imagen «Puntos de presencia RedIRIS» (p. 8): CC BY SA Gobierno de España
Imagen «Unicast» (p. 9): PD
Imagen «Multicast» (p. 9): PD
Imagen «Anycast» (p. 9): PD
Imagen «Broadcast» (p. 9): PD
Imagen «Logo Wi-fi alliance» (p. 13): PD Wi-fi Alliance
Imagen «Sobre» (p. 20): CC BY Windell Oskay (fondo eliminado)
Imagen «Matriosca rusa» (p. 21): CC BY SA Greyhood
Imagen «Repetidor» (p. 25): CC BY SA Calips
Imagen «Conmutador» (p. 26): CC BY SA Geek2003
Imagen «Encaminador doméstico» (p. 27): CC BY SA Raimond Spekking (fondo eliminado)
Imagen «Cisco CRS-1» (p. 27): CC BY SA Cisco Systems Inc.

Redes de Computadores

Tema 2 – Capa Física

Juan Segarra y Jesús Alastruey

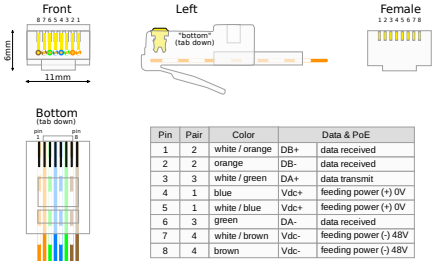
Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción

La capa física se encarga de la **interfaz física** entre las tecnologías de transmisión de la red:

- Especificaciones **mecánicas** de conectores y cables
- Especificaciones **electromagnéticas** de la señal
- Especifica cómo **emitir** los bits e **interpretar** la señal



Pin	Pair	Color	Data & PoE
1	2	white / orange	DB+ data received
2	2	orange	DB- data received
3	3	white / green	DA+ data transmit
4	1	blue	Vdc+ feeding power (+) 0V
5	1	white / blue	Vdc+ feeding power (+) 0V
6	3	green	DA- data received
7	4	white / brown	Vdc- feeding power (-) 48V
8	4	brown	Vdc- feeding power (-) 48V

2 Señales y ondas electromagnéticas

Señal: función que transmite información

- **Continua** o **discreta**
- **Periódica** (se repite un patrón a lo largo del tiempo) o **no periódica**

Señal continua periódica

2.1 Señal periódica

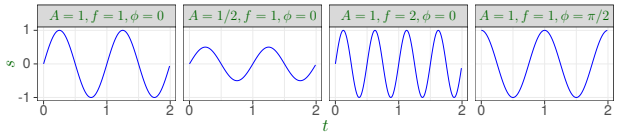
$$s(t) = A \cdot \sin(2\pi f t + \phi)$$

Amplitud (A): valor máximo de la señal, en unidades dependientes del tipo de onda (V, Pa, m, etc.)

Frecuencia (f): tasa a la que se repite la señal, en Hertzs o Hercios (Hz), o ciclos dividido por segundos (c/s)

Periodo ($T = 1/f$): tiempo transcurrido entre dos repeticiones consecutivas de la señal, en segundos (s)

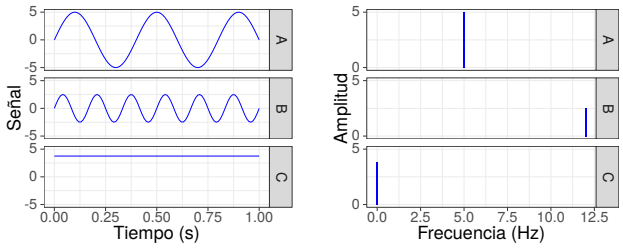
Fase (ϕ): posición relativa de la señal dentro de un periodo



2.2 Análisis de Fourier

- Toda señal periódica puede representarse como la suma de múltiples señales sinusoidales

$$s(t) = \frac{1}{2}A_0 + \sum_{n=1}^{\infty}\left[A_n \sin(2\pi n f_0 t) + B_n \cos(2\pi n f_0 t)\right]$$

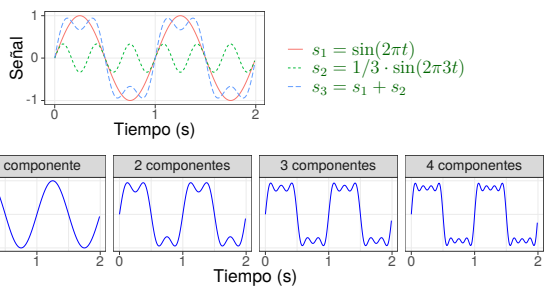


Componente continua (DC): aquella con frecuencia cero

2.2 Análisis de Fourier (II)

- Ejemplo: onda cuadrada (f impares, $A = 1/f$)

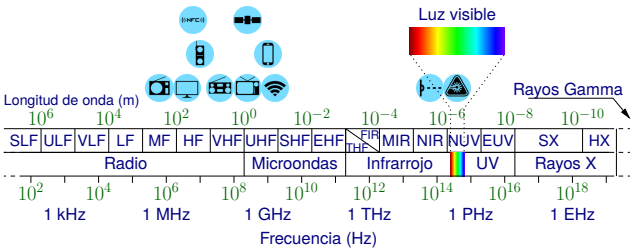
$$s(t) = \sum_{n=1}^{\infty} \frac{1}{2n-1} \sin\left(2\pi(2n-1)t\right)$$



2.3 Espectro electromagnético

Onda electromagnética: forma de propagación de la radiación electromagnética

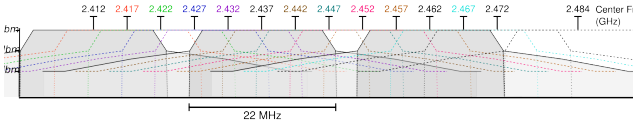
Espectro electromagnético: distribución energética del conjunto de ondas electromagnéticas



2.3 Espectro electromagnético (II)

Ancho de banda de frecuencias (B): rango de frecuencias que conforman una onda o que pueden atravesar un canal

- Voz: 100–7000 Hz $\rightarrow B = 6900\ Hz$
- Teléfono: 300–3400 Hz $\rightarrow B = 3100\ Hz$
- Wi-Fi (802.11g): canales de $B = 22\ MHz$ en banda 2.4 GHz



- Cualquier sistema sólo puede transmitir por una banda limitada de frecuencias
  - Por limitaciones físicas/tecnológicas
  - Por legislación vigente

2.4 Decibelio (dB)

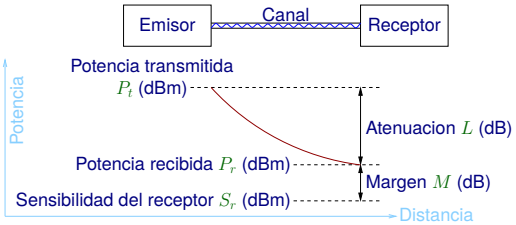
- Los **decibelios** (dB) se usan para expresar la **relación** (logarítmica) entre dos magnitudes M_1 y M_2

$$10 \cdot \log_{10}\left(\frac{M_1}{M_2}\right)$$

- Relación entre potencia de señal y ruido (SNR): $M_1 = Signal$; $M_2 = Noise$
- Ganancia (G) o atenuación (L) de potencia de un sistema: $M_1 = P_{Salida}$; $M_2 = P_{Entrada}$
- Relación respecto a 1 mW (dBm): $M_1 = P$; $M_2 = 1$ mW
- Ganancias/atenuaciones se pueden sumar/restar ($G = -L$)

2.4 Decibelio (dB) (II)

- Esquema de atenuación:



$$P_t - L = P_r$$
$$P_t - S_r = L + M$$
$$P_r - S_r = M \geq 0$$

3 Medios de transmisión

Medio de transmisión: soporte a través del cual se propaga la señal transmitida

- Guiado o línea de transmisión:
  - Par trenzado
  - Cable coaxial
  - Fibra óptica
- No guiado (inalámbrico):
  - Radiofrecuencia
  - Infrarrojos (IrDA) (obsoleto)
  - Láser (no usado en redes de computadores)

3.1 Par trenzado

- Dos conductores: señal + referencia
- Cada cable está compuesto por una serie de pares trenzados (4, 25, 50, 100, 200 y 300)
- Se trenzan para reducir las interferencias externas y entre pares adyacentes
- El ancho de banda depende de la sección y longitud
- Bajo coste y facilidad de instalación

U/UTP F/UTP S/FTP

3.1 Par trenzado (II)

UTP (*unshielded twisted pair*): no apantallado
- Cable más utilizado en redes locales

STP (*shielded twisted pair*): cubierta metálica trenzada
- Reduce tasa de error y emisiones electromagnéticas

FTP (*foiled twisted pair*): cubierta de papel de aluminio
- Características similares a STP con costes inferiores

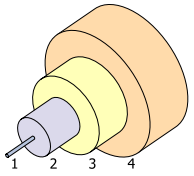
- Designación ISO/IEC 11801 según tipo de cubierta global / de cada par (U/UTP, U/FTP, SF/UTP, S/FTP…)
- Designación por categoría según prestaciones, e.g. cat 5: 100 Mb/s U/UTP, cat 7: 10 Gb/s F/FTP o S/FTP

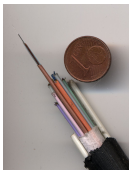
3.2 Cable coaxial

- Hilo conductor central de cobre rodeado por malla de cobre
- Conductores separados por aislante
- Puede estar apantallado
- Buen ancho de banda (1 GHz) y excelente inmunidad al ruido
- Coste elevado
- Uso más común: TV y cableado final en accesos domésticos de fibra óptica
- Está siendo sustituido por fibra óptica

3.3 Fibra óptica

- Hilo muy fino de vidrio o plástico por el que se envían pulsos de luz
- El haz de luz queda confinado y se propaga por el interior de la fibra con cierto ángulo de reflexión





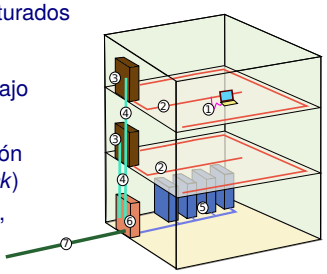
1: fibra 8-10 µm; 2: contenedor 125 µm; 3,4: cubiertas protectoras

3.3 Fibra óptica (II)

- Ventajas
  - Ancho de banda elevado
  - Baja atenuación $\rightarrow$ largas distancias
  - Pequeñas dimensiones, flexibilidad, ligereza
  - Inmunidad total a perturbaciones electromagnéticas
  - Seguridad: no emite radiaciones y es difícil de «pinchar»
  - Resistencia mecánica, calor, corrosión
- Inconvenientes
  - Fragilidad de las fibras
  - Emisores y receptores caros
  - Velocidad condicionada por la electrónica de emisión y recepción (10 Gb/s)
  - No pueden transmitir electricidad para alimentar receptores o repetidores
  - Empalmes difíciles de realizar en campo

3.4 Cableado estructurado

- Objetivo: optimizar la gestión y el mantenimiento
- Division en tramos estructurados



1. Cableado de área de trabajo
2. Cableado horizontal
3. Cableado de administración (armario de cableado, *rack*)
4. Cableado vertical (central, *backbone*)
5. Centro de cálculo
6. Cableado de equipamiento (armario de entrada al edificio)
7. Cableado del campus (acometida, cableado entre edificios)

3.5 Radiofrecuencia (RF)

- Ondas de radio (30 MHz-1 GHz)
  - No usado en redes de computadores
- Microondas terrestre (2-40 GHz)
  - Wi-Fi y telefonía móvil
- Microondas satélite (2-40 GHz)
  - No usado en redes de computadores

3.6 Velocidad de propagación

Velocidad de propagación (V_p): rapidez con la que una onda electromagnética viaja a través del medio de transmisión

Medio	V_p (m/s)	% c
Espacio libre	$c = 3.0 \cdot 10^8$	100
Cobre (S/FTP, cat. 7a)	$2.4 \cdot 10^8$	80
Fibra óptica	$2.0 \cdot 10^8$	67
Cobre (UTP, cat. 5e)	$1.9 \cdot 10^8$	63

4 Transmisión digital

Dada una secuencia binaria a transmitir, hay que convertir los bits en ondas compatibles con el canal:

- Transmisión en banda base: **modulación por pulsos** o **modulación en banda base**
  - E.g. transmisión ethernet sobre par trenzado
- Transmisión en canal paso banda: **modulación por** señal sinusoidal **portadora**
  - E.g. transmisión WiFi por aire

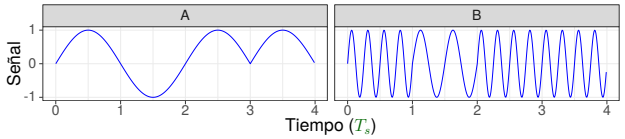
4.1 Símbolos y bits

Símbolo: forma de representar cierta información

Alfabeto: conjunto de símbolos

Si hablamos de ondas electromagnéticas…

Símbolo: señal modulada con ciertos parámetros durante un **tiempo de símbolo** T_s



- Dos valores (0/1) necesitan dos símbolos
- Una señal con un alfabeto de M símbolos podrá transmitir $k = \log_2(M)$ **bits/símbolo**, con $k \in \mathbb{N}$

4.1 Símbolos y bits (II)

Tasa de símbolos (R_s): número de símbolos en la señal modulada por unidad de tiempo, en **baudios** (Bd, símbolos/s)

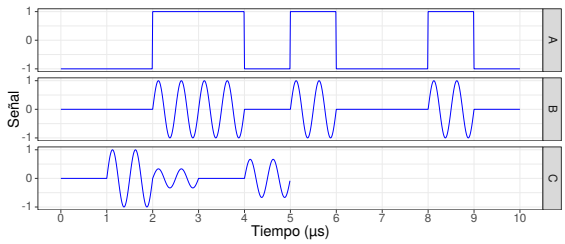
$$R_s = \frac{símbolos}{tiempo} = \frac{1}{T_s} \quad \text{(Bd)}$$

Tasa de bits (R_b): número de bits en la señal modulada por unidad de tiempo, en **bits/segundo** (b/s, bps)

$$R_b = \frac{bits}{tiempo} = R_s \cdot k = R_s \cdot \log_2(M) \quad \text{(b/s)}$$

4.1 Símbolos y bits (III)

✍ Se envía la secuencia de bits 0011010010 mediante distintas señales con el mismo tiempo de símbolo $T_s = 1$ µs. Para cada señal, indica el número de símbolos M, los bits/símbolo k, la tasa de bits R_b y la tasa de símbolos R_s.

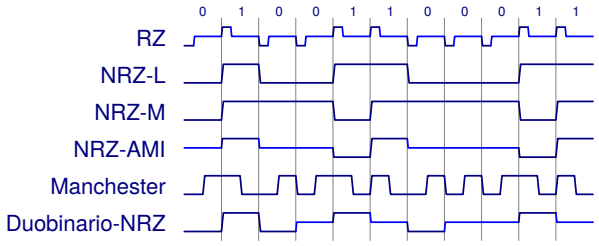


4.2 Modulación por pulsos

- Bits transmitidos mediante pulsos por un canal banda base
- Existen multitud de codificaciones como por ejemplo:
  - RZ (Retorno a cero): pulso ($0 \equiv V^-$, $1 \equiv V^+$) y vuelta a reposo
  - NRZ-L (No retorno a cero, por nivel): $0 \equiv V^-$, $1 \equiv V^+$
  - NRZ-M (NRZ diferencial): 0 mantiene V, 1 cambia V
  - NRZ-AMI (NRZ inversión alterna): $0 \equiv 0\,V$, 1 alterna $V^{+/-}$
  - Manchester: $0 \equiv V^- \rightarrow V^+$, $1 \equiv V^+ \rightarrow V^-$ (señal de reloj integrada en la codificación)
  - Duobinario-NRZ: mismo valor en bit $\equiv 0\,V$, distinto valor $\equiv$ alterna $V^{+/-}$
- Cada codificación tiene sus aplicaciones

4.2 Modulación por pulsos (II)

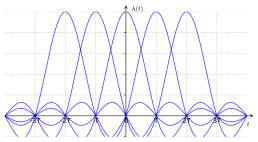
- Ejemplos:



4.3 Tasa de Nyquist

Distorsión: alteración de una señal causada porque frecuencias distintas se propagan a velocidades distintas
- Los pulsos se solapan (interferencia entre símbolos)

Nyquist[1]: el ancho de banda mínimo teórico para recibir adecuadamente R_s es $B = R_s/2$ Hz



Tasa de Nyquist

$$R_s \leq 2B$$

Eficiencia espectral: $E = R_b/B$

4.4 Modulación por portadora

Modulación por portadora: variación de la amplitud, frecuencia o fase de una señal sinusoidal (portadora) según los símbolos a transmitir

$$s(t) = A \cdot \cos(2\pi f t + \phi)$$

- Mayor eficiencia espectral que modulación por pulsos
- Hardware más sencillo: antenas, filtros, amplificadores…
- Módem:
  - Modulador: modifica alguna característica de la portadora
  - Demodulador: elimina la señal portadora

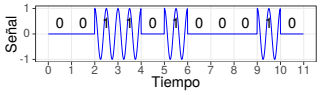
4.4.1 ASK (amplitude shift keying)

- Los símbolos se representan mediante diferentes amplitudes (A_i) en la señal portadora

$$s(t) = A_i \cdot \cos(2\pi f t + \phi),\ i = 1, \ldots, M$$

- Por ejemplo, ASK binario (OOK, *on-off keying*):

$$s(t) = \begin{cases} A\cos(2\pi f t) & 1 \\ 0 & 0 \end{cases}$$



- Sensible a cambios repentinos de la ganancia (ruido impulsivo)
- Se usa en fibras ópticas

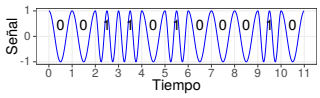
4.4.2 FSK (frequency shift keying)

- Los símbolos se representan mediante diferentes frecuencias (f_i) en la señal portadora

$$s(t) = A \cdot \cos(2\pi f_i t + \phi),\ i = 1, \ldots, M$$

- Por ejemplo, FSK binario (BFSK):

$$s(t) = \begin{cases} A\cos(2\pi f_1 t) & 1 \\ A\cos(2\pi f_2 t) & 0 \end{cases}$$



- Se utiliza en telefonía analógica y digital para identificación de llamada

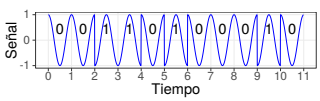
4.4.3 PSK (phase shift keying)

- Los símbolos se representan mediante diferentes fases (ϕ_i) en la señal portadora

$$s(t) = A \cdot \cos(2\pi f t + \phi_i),\ i = 1, \ldots, M$$

- Por ejemplo, PSK binario (BPSK):

$$s(t) = \begin{cases} A\cos(2\pi f t + \pi) & 1 \\ A\cos(2\pi f t) & 0 \end{cases}$$

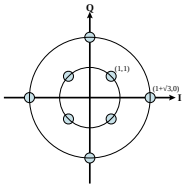


- PSK diferencial (DPSK): fase depende del símbolo anterior
- Amplio uso: 802.11b, Bluetooth…

4.4.4 QAM (quadrature amplitude mod.)

- Combinación de ASK y PSK
- Usa distintas amplitudes (A) y fases (ϕ) para codificar varios bits por símbolo
- E.g. 8-QAM circular: fases de $45°$ y dos amplitudes (A y B)

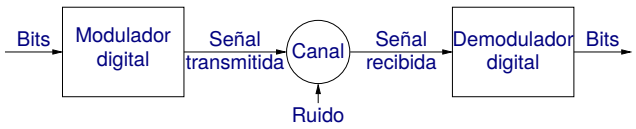
$000 \rightarrow s(t) = A\sin(2\pi f t + 0°)$
$001 \rightarrow s(t) = B\sin(2\pi f t + 45°)$
$010 \rightarrow s(t) = A\sin(2\pi f t + 90°)$
$011 \rightarrow s(t) = B\sin(2\pi f t + 135°)$
$100 \rightarrow s(t) = A\sin(2\pi f t + 180°)$
$101 \rightarrow s(t) = B\sin(2\pi f t + 225°)$
$110 \rightarrow s(t) = A\sin(2\pi f t + 270°)$
$111 \rightarrow s(t) = B\sin(2\pi f t + 315°)$



- Uso WiFi:

802-11n	802.11ac	802.11ax	802.11be
64-QAM	256-QAM	1024-QAM	4096-QAM

5 Capacidad de un canal con ruido



Cualquier canal tiene perturbaciones:

Atenuación: pérdida de energía de la señal al propagarse

Distorsión de retardo: frecuencias distintas viajan a velocidades distintas

Ruido: señales insertadas entre emisor y receptor

- Ruido térmico: agitación de los electrones
- Intermodulación: varias frecuencias en el mismo medio
- Diafonía: acoplamiento entre líneas
- Ruido impulsivo: pulsos cortos e irregulares

5.1 Teorema de Shannon-Hartley

Las perturbaciones pueden hacer que el receptor confunda los símbolos de la señal

- El tipo de modulación, el número de símbolos M y su tasa R_s vendrán limitados por la capacidad para distinguirlos

Shannon[i] modela la **capacidad** de un canal con ruido

$$C = B \cdot \log_2\left(1 + \frac{S}{N}\right) \quad \text{(b/s)}$$

- Pocos errores al transmitir a una tasa $R_b \leq C$
  - **Bit Error Ratio** (BER): bits erróneos/transmitidos
- Recordar: $\log_2(x) = \frac{\ln(x)}{\ln(2)} = \frac{\log_{10}(x)}{\log_{10}(2)}$

5.2 Resumen

Capacidad de un canal (b/s):

- $C = B \cdot \log_2\left(1 + \frac{S}{N}\right)$

Tasa de bits de una señal (b/s):

- $R_b = \frac{bits}{segundo} = \underbrace{\frac{R_s}{\frac{símbolos}{segundo}}} \cdot \underbrace{\log_2(M)}_{\frac{bits}{símbolo}}$

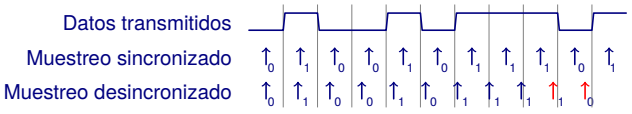
Para que todo funcione:

- $R_s \leq 2B$ (Nyquist)
- $R_b \leq C$ (Shannon)

…con ancho de banda B, tasa de símbolos R_s, M símbolos y relación señal-a-ruido $\frac{S}{N}$ (recordar dB: $SNR = 10 \cdot \log_{10}\left(\frac{S}{N}\right)$)

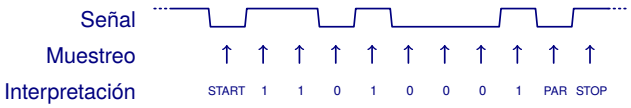
6 Sincronismo

- El receptor debe saber cuándo muestrear cada símbolo
- Alta velocidad → relojes con precisión de µs/ns
- Todos los relojes se adelantan/atrasan → la precisión requerida no se puede garantizar durante mucho tiempo



6.1 Transmisión asíncrona

- Envío de caracteres entre 5 y 8 bits + pausa
- Sincronización de relojes al inicio de cada carácter
- Sencilla y barata
- Velocidad de transmisión baja, sobrecarga alta

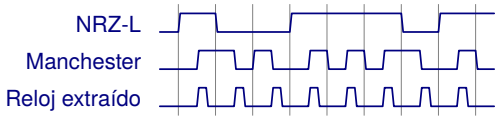


Caracter transmitido: 10001011

- Deben especificarse todos los parámetros, por ejemplo:
  - 9600/8N1 (9600 b/s, 8 bits sin paridad y 1 bit de stop)
  - 9600/7E2 (9600 b/s, 7 bits p. par (*even*) y 2 bits de stop)
  - 4800/7O1 (4800 b/s, 7 bits p. impar (*odd*) y 1 bit de stop)

6.2 Transmisión síncrona

- Contexto: flujo constante de grandes bloques de bits o altas velocidades de transmisión
- Solución 1: **línea adicional** que transmite la señal de reloj
  - Funciona bien en distancias cortas
- Solución 2: incorporar **transiciones frecuentes** en señal de datos para autosincronizarse
  - E.g. Manchester codifica una transición a mitad de bit



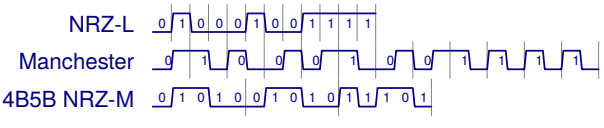
6.2 Transmisión síncrona (II)

- **Códigos de sustitución** garantizan transiciones, e.g. 4B5B

Datos	4B5B	Datos	4B5B	Datos	4B5B	Datos	4B5B
0000	11110	0100	01010	1000	10010	1100	11010
0001	01001	0101	01011	1001	10011	1101	11011
0010	10100	0110	01110	1010	10110	1110	11100
0011	10101	0111	01111	1011	10111	1111	11101

4B5B: nunca más de tres 0s consecutivos

- Ejemplo NRZ-L (no sincronizable), Manchester, 4B5B sobre NRZ-M (Ethernet 100 Mb/s), con igual tiempo de símbolo

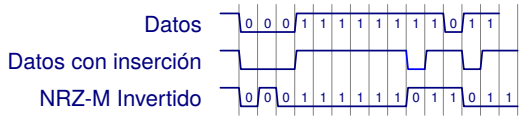


- Velocidad de transmisión [Tema 3]: $V_t(\text{NRZ-L}) = R_b(\text{NRZ-L})$
  $V_t(\text{4B5B-NRZ-M}) = \frac{4}{5}R_b(\text{NRZ-L})$ $V_t(\text{Manchester}) = \frac{1}{2}R_b(\text{NRZ-L})$

6.2 Transmisión síncrona (III)

- **Inserción de bits** (*bit stuffing*): ante señales largas sin transiciones se inserta un bit que provoque transición

Ejemplo: Emisor 111111 → 1111110
Receptor 1111110 → 111111



7 Tipos de transmisión

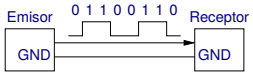
Simplex: comunicación en un único sentido

Half-duplex: en ambos sentidos pero no a la vez

Full-duplex: en ambos sentidos simultáneamente

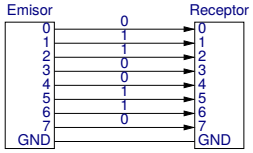
Serie: los símbolos se transmiten por el mismo canal **secuencialmente**

- Dos cables: señal y referencia



Paralelo: distintos símbolos se transmiten por diferentes canales **simultáneamente**

- Más caro (requiere más cables)
- Menor distancia (diafonía, sincronización entre canales)



8 Ejemplo especificaciones Ethernet

Especificaciones físicas ethernet (IEEE 802.3): identificadores cortos con tasa de bits, señalización y medio físico/codificación

10BASE-T: 10 Mb/s, banda base, par trenzado

100BASE-T (fast ethernet): 100 Mb/s, banda base, p. trenzado

- 100BASE-TX: 100 Mb/s, banda base, par trenzado cat. 5, full-duplex, 4B5B/NRZ-I, para LAN
- 100BASE-T1: 100 Mb/s, banda base, par trenzado cat. 5e, full-duplex, PAM3, para automoción
- …

1000BASE-T: 1000 Mb/s, banda base, par trenzado

10GBASE-SR: 10 Gb/s, banda base, fibra óptica

…

Créditos de material reutilizado

Imagen «conectores RJ45» (p. 3): Hieke
Imagen «especificaciones RJ45» (p. 3): jobefox
Imagen «Espectro electromagnético» (p. 8): Duarte Farrajota Ramos (obra derivada)
Imagen «Peligro, láser» (p. 8): Lorc
Imagen «2.4 GHz Wi-Fi channels (802.11g WLAN)» (p. 9): Michael Gauthier, Wireless Networking in the Developing World
Imagen «Par trenzado U/UTP» (p. 13): Baran Ivo
Imagen «Par trenzado F/UTP» (p. 13): Baran Ivo
Imagen «Par trenzado S/FTP» (p. 13): Ru wiki
Imagen «Cable coaxial RG-6» (p. 15): User:Ronibd0
Imagen «Conectores IEC 169-2» (p. 15): Josutus
Imagen «Conectores coaxiales» (p. 15): Adamantios
Imagen «Sección fibra monomodo» (p. 16): Benchill
Imagen «Cable fibra» (p. 16): Christophe Merlet
Imagen «Conectores fibra óptica» (p. 16): Elsanto510.ULE
Imagen «Cableado estructurado» (p. 18): PePeEfe
Imagen «Raised-cosine-ISI» (p. 27): Chris828
Imagen «8QAM circular» (p. 32): Life of Riley
Imagen «Radio» (p. 41): Stefan Kühn
Imagen «Walkie-talkies» (p. 41): Jaysin Trevino (fondo eliminado)
Imagen «Videollamada» (p. 41): methodshop.com (texto eliminado)

Redes de Computadores

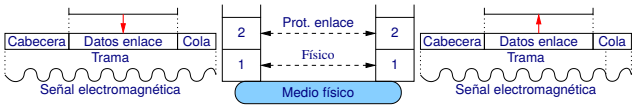
Tema 3 – Capa de enlace de datos

Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción



Funciones típicas de la capa de enlace:

- Reglas para que varias entidades compartan un mismo canal de transmisión: **control de acceso al medio (MAC)**
- **Comunicación fiable** entre dos entidades:
  - Control de errores
  - Secuenciación de datos
- Unidad básica de nivel de enlace: **trama** (*frame*)

2 Métricas

Longitud de trama (L): número de bits en una trama (bits, b)

Velocidad de transmisión (V_t): tasa a la que se inyecta una trama en el medio de transmisión (bits/segundo, b/s, bps)

- Excluye la sobrecarga de la capa física, por ejemplo, bits insertados para la sincronización
  - 4B5B-NRZ-M: $R_b = 100$ Mb/s, $V_t = \frac{4}{5}R_b = 80$ Mb/s
  - 802.11g: $R_b = 54$ Mb/s, $V_t = 22$ Mb/s

Tiempo de transmisión (T_t): tiempo necesario para inyectar una trama en el medio de transmisión (segundos, s)

$$T_t = \frac{L}{V_t}$$

2 Métricas (II)

Distancia (D): longitud del enlace (metros, m)

Velocidad de propagación (V_p): rapidez con que una onda viaja por el enlace (metros/segundo, m/s) [Tema 2]

Tiempo de propagación (T_p): tiempo necesario para que una onda viaje de emisor a receptor (segundos, s)

$$T_p = \frac{D}{V_p} \qquad V_p \approx \begin{cases} 3.0 \cdot 10^8\ m/s & \text{en espacio libre} \\ 2.4 \cdot 10^8\ m/s & \text{en cobre, cat. 7a} \\ 1.9 \cdot 10^8\ m/s & \text{en cobre, cat. 5e} \\ 2.0 \cdot 10^8\ m/s & \text{en fibra óptica} \end{cases}$$

2 Métricas (III)

Tiempo de acceso al canal (T_a): tiempo desde que un nodo quiere transmitir hasta que empieza a transmitir la trama «definitiva» (esperas por colisiones, por turnos, etc.)

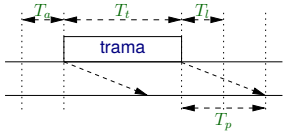
Tiempo de liberación del canal (T_l): mínimo tiempo de espera obligatorio entre transmisiones consecutivas

Tiempo de procesamiento: tiempo de toma de decisiones

Velocidad efectiva de datos (V_e): tasa de datos (no cabeceras) transferidos por unidad de tiempo

Utilización efectiva de datos del canal (U_e): porcentaje de tiempo en el que se transmiten datos por el canal

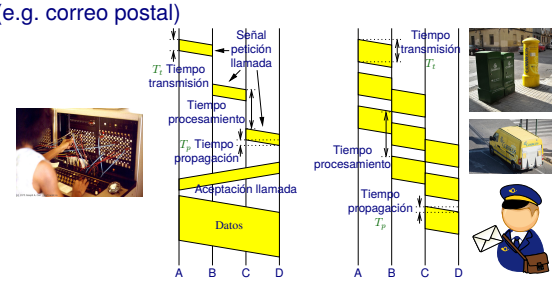
$$V_e = \frac{datos}{T_a + T_t + T_l} \qquad U_e = \frac{V_e}{V_t} \times 100$$



3 Modelos de conmutación

Conmutación de circuitos: la información viaja por un circuito establecido antes de iniciar la transmisión, sin almacenarse durante su recorrido (e.g. teléfono)

Conmutación de paquetes: la información se divide y cada bloque se almacena y reexpide entre conmutadores (e.g. correo postal)



3 Modelos de conmutación (II)

- Un medio de transmisión suele ser compartido
- Si varios emisores transmiten a la vez, sus señales pueden colisionar (superponerse) y no ser recibidas correctamente
- ¿Cómo se gestiona el uso compartido?
- El **control de acceso al medio (MAC)** establece las reglas de uso del canal para evitar/minimizar colisiones

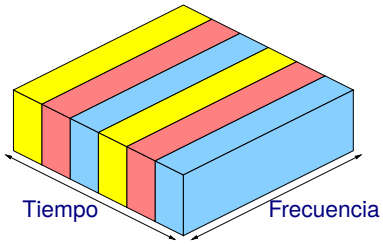
4 MAC en conmutación de circuitos

- Se **establece un circuito** antes de iniciar la transmisión
- El establecimiento va asociado con una reserva de recursos en los nodos (frecuencias, tiempo, etc.)
- **Calidad de servicio** (QoS) implícita al reservar recursos
- **Control de admisión**: cuando se agotan los recursos no se permiten más circuitos y no se admiten más conexiones
- Una vez establecido el circuito, todos los datos viajan por él
- Si se corta el circuito, la conexión se corta
- **Conmutación por circuito virtual**: uso de circuitos lógicos sobre conmutación de paquetes [Tema 4]
- Ejemplos: https://www.spectrummonitoring.com/standards/

4.1 TDMA

Time division multiple access

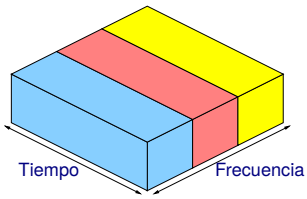
- Necesita *buffers*
- Necesita sincronización
- E.g. GSM (*Global System for Mobile communications*), Bluetooth



4.2 FDMA

Frequency division multiple access

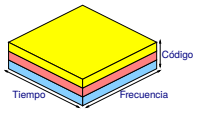
- Señales moduladas con portadoras a distinta frecuencia (sin solapar)
- WDMA (*wavelength*) en fibra óptica
- E.g. Bluetooth, 3G, ADSL (*Asymmetric Digital Subscriber Line*), PON (*Passive Optical Network*, fibra doméstica)

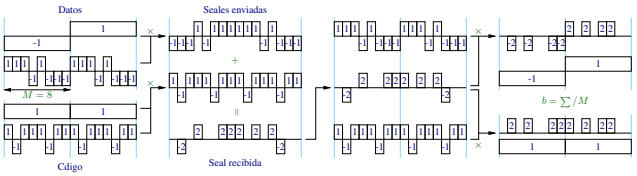


4.3 CDMA

Code division multiple access

- Usado en comunicaciones inalámbricas
- Mayor tolerancia ante interferencias
- E.g. GPS (*Global Positioning System*), UMTS (*Universal Mobile Telecommunication System*)





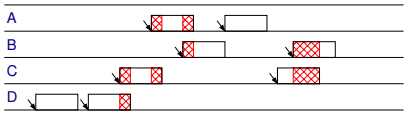
5 MAC en conmutación de paquetes

- Más **tolerante a fallos** que conm. circuitos: ante fallos los paquetes pueden ir por rutas alternativas [Tema 4]
- Suele implicar **más sobrecarga**: información de control por paquete y no por canal como en conmutación de circuitos
- Ocupación del canal bajo demanda
  - Si no hay datos a transmitir no se ocupa el canal
  - Adecuado para tráfico en ráfagas (usual en Internet)
- Multiplexación estadística **con colisiones**: Aloha, CSMA/CA (IEEE 802.11), CSMA/CD (IEEE 802.3), etc.
- Multiplexación estadística **sin colisiones**: Token-Ring (IEEE 802.5), etc.

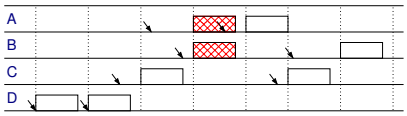
5.1 Aloha

Pensado para comunicaciones de radio entre islas

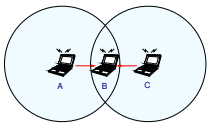
- Aloha puro: pueden iniciarse transmisiones en cualquier momento



- Aloha ranurado: sólo pueden iniciarse transmisiones al inicio de ranuras/*slots* periódicas



5.2 CSMA/CA



Carrier Sense Multiple Access / Collision Avoidance

1. Escucha el canal hasta que esté libre
2. Transmite
3. Espera la confirmación de la recepción (trama ACK)
4. Si no se recibe confirmación, espera un tiempo aleatorio y vuelve al paso 1

Detalles:

- Tramas separadas por un espacio mínimo (*gap*)
- Usado en IEEE 802.11 Wi-Fi

5.2.1 *Backoff* exponencial

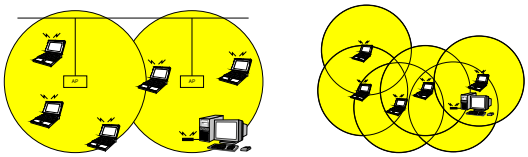
- Si el **tiempo de espera** fuera determinista → nueva colisión
- Siendo aleatorio se reduce la probabilidad de colisión
- Se adapta aleatoriedad a probabilidad de colisión
  - Empezar asumiendo poca probabilidad
  - Ante colisiones consecutivas, aumentar exponencialmente el intervalo de tiempo máximo
- $TiempoEspera = backoff \cdot T_{slot}$. E.g. $T_{slot} = 9\ \mu s$ en 802.11n
- Ejemplo:

1 col: $backoff \in \{0, 1\}$
2 col: $backoff \in \{0, 1, 2, 3\}$
3 col: $backoff \in \{0, 1, 2, 3, 4, 5, 6, 7\}$
4 col: $backoff \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$
…

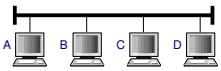
- Inconveniente: efecto LIFO (*last-in-first-out*)

5.2.2 Ejemplo de uso

- Puntos de acceso (AP)
  - Los AP conforman una red ya desplegada
  - Los equipos se conectan a uno de los APs
  - Los AP proporcionan configuración (DHCP [Tema 4])
  - Los equipos envían/reciben siempre a través del AP
- Redes ad-hoc
  - No hay una red desplegada a la que conectarse
  - Los equipos se autoorganizan funcionando como retransmisores



5.3 CSMA/CD



Carrier Sense Multiple Access / Collision Detection

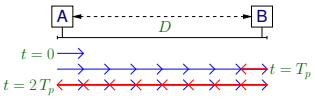
1. Escucha el canal hasta que esté libre
2. Transmite
3. Si detecta colisión, sustituye la transmisión por una señal corta (32 bits) de alerta (*jam*) para enfatizar la colisión, espera un tiempo aleatorio y vuelve al paso 1

Detalles:

- No hay ACK → **el emisor debe detectar colisiones**
- *Backoff* igual que en CSMA/CA, con $T_{slot} \approx T_t(TramaMin)$
- Tramas separadas por un espacio mínimo (*gap*)
- Usado en IEEE 802.3 Ethernet

5.3.1 Detección de colisiones

Si hay colisión, el emisor la debe detectar **siempre**



- Colisión detectada si la señal en el canal no coincide con la transmitida en ese instante
- Distancia del cable máxima D establecida por protocolo
- Colisiones detectables en $t \leq 2\,T_p$, con $T_p = \frac{D}{V_p}$
- Para detectar colisión hay que estar aún transmitiendo en $t = 2\,T_p$, es decir, $T_t \geq 2\,T_p$, con $T_t = \frac{L}{V_t}$
- Dado que $L \geq 2\,T_p \cdot V_t$, el protocolo establece la longitud mínima de trama $L_{min} = 2\,T_p \cdot V_t$ para garantizar detección

5.3.2 Ejemplo trama

Ejemplo: trama Ethernet II (72–1530 bytes)

7	1	6	6	(4)	2	46–1500	4
Preámbulo	SFD	Id. destino	Id. origen	VLAN	Tipo	Datos	FCS

Preámbulo: 10101010 × 7

SFD (*Start-of-Frame-Delimiter*): 10101011

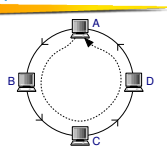
Origen/destino: identificadores MAC de las tarjetas de red origen y destino (e.g. lab000: 50:65:f3:42:43:37)

VLAN (opcional): identificador de VLAN 802.1Q [p. 27]

Tipo: identificador (demultiplexor) del protocolo encapsulado en el campo «Datos»

FCS (*Frame Check Sequence*): CRC de 32 bits [p. 37]

5.4 Token Ring (IEEE 802.5)



- Estaciones conectadas en **anillo unidireccional**
- Cada estación propaga las tramas que recibe (con 1 bit de retardo para procesar bits particulares)
- Una **trama especial** (testigo/*token*) circula por el anillo
- Cuando se recibe el *token*, se puede capturar o propagar
- Para transmitir hay que estar en posesión del *token*:
  - Esperar a que llegue el *token* y capturarlo
  - Transmitir la trama de datos en un sentido del anillo
  - Quitar la trama de datos cuando llegue por el otro lado
  - Transmitir el *token*

5.4 Token Ring (IEEE 802.5) (II)

- Transmisiones por **turnos** → no hay colisiones
- **Prioridades**: mensajes más prioritarios se transmiten antes
  - Las estaciones marcan su prioridad sobre la trama propagada
  - El nuevo token se genera con la máxima prioridad marcada
  - Solo quien tiene prioridad suficiente puede capturar el token
- **Control distribuido** del anillo (todas las estaciones deben acordar qué estación monitoriza el funcionamiento):
  - que el turno vaya pasando
  - que ninguna trama de datos se quede dando vueltas indefinidamente
  - que no se «atasque» la prioridad
  - que quien monitoriza el funcionamiento no se bloquee

5.4.1 Ejemplo tramas

Tramas de token y datos en Token Ring (IEEE 802.5)

1	1	1
SD	AC	ED

1	1	1	2–6	2–6	n	4	1	1
SD	AC	FC	Id. destino	Id. origen	Datos	FCS	ED	FS

SD (*Starting Delimiter*): JK0JK000 (Manchester, J: HH, K: LL)

AC (*Access Control*): PPPTMRRR
(P: prioridad, T: token, M: monitor, R: reserva)

FC (*Frame Control*): tipo trama (demultiplexor) y bits de control

Origen/destino: identificadores MAC origen y destino

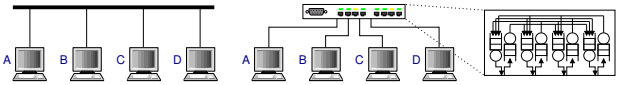
FCS (*Frame Check Sequence*): CRC de 32 bits [p. 37]

ED (*Ending Delimiter*): JK1JK1IE (trama Intermedia, Error)

FS (*Frame Status*): ACrrACrr
(A: dir. reconocida, C: trama copiada, r: reservado)

6 Conmutación en Ethernet

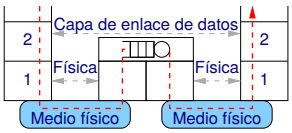
- LANs ethernet (CSMA/CD) fueron ganando terreno
  - Fácil mantenimiento, buenas prestaciones con baja carga
- Más dispositivos en medio compartido → más colisiones → menos prestaciones
- Solución: conmutador (*switch*) que **almacena y reexpide** las tramas a través de enlaces no compartidos



- Cada equipo dispone de un **cable dedicado** → todos pueden transmitir a la vez → n veces más capacidad pico
  - *Fast ethernet* (100Base-TX): un par trenzado para enviar y otro para recibir → canal full-duplex sin colisiones

6 Conmutación en Ethernet (II)

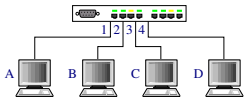
- Cambio de comunicación directa vía medio compartido a indirecta vía conmutador



- El conmutador es **transparente** para los dispositivos
- ✕ **Congestión**: pérdida de tramas por falta de memoria en conmutador [Tema 6]
- ✕ **Contención**: latencia adicional y variable en colas de conmutador
- ✕ El conmutador sólo une **redes del mismo tipo o muy similares**, e.g. ethernet y WiFi (mismo esquema MAC)

6.1 Conmutadores aprendices

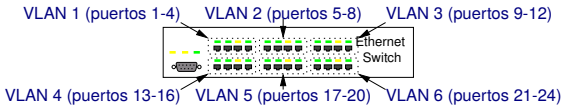
- Sin conmutadores aprendices, aunque no se comparta el canal, el tráfico es el mismo → **no escala**
- Los conmutadores aprendices ven que las tramas con identificador MAC de origen x llegan por interfaz y
  - Mantienen tabla con parejas $\langle x, y \rangle$ que expiran con el tiempo
- Al recibir tramas dirigidas a x, se reexpiden sólo hacia y
  - Se evita tráfico innecesario
  - Se dificulta la monitorización del tráfico de otros
  - Decide por dónde enviar → **No es función de capa 2**
- Tramas con destinos no conocidos o de difusión total (*broadcast*) se reexpiden a todos



Estacin	Puerto HW
A	1
B	2
C	3
D	4

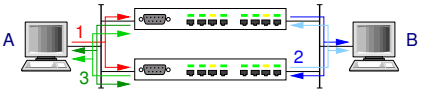
6.2 LANs virtuales (VLANs)

- Las tramas de difusión (*broadcast*) han de llegar a todos → no escala para redes muy grandes
- División de LAN en dominios de difusión (VLANs)
  - Tramas de difusión no se reexpiden a otras VLANs
  - Id. VLAN asociado a interfaces o id. MAC
  - IEEE 802.1Q: modifica la cabecera ethernet para incluir campo id. VLAN [p. 20]
- Decide a quién enviar → **No es función de capa 2**



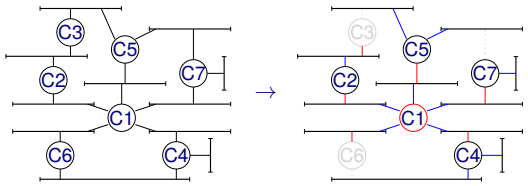
6.3 *Spanning tree*

- Tolerancia a fallos → conmutadores y enlaces redundantes
- Bucles dan problemas (ejemplo: 1 2 3 2 3 2 3 …)
  - Tramas duplicadas y dando vueltas indefinidamente
  - Imposible asociar parejas id.origen–interfaz



- Hay que deshabilitar conexiones redundantes… ¿cuáles?
  - Teoría de grafos: **un *spanning tree*** es un árbol que cubre (sin bucles) todos los nodos (conmutadores) del grafo
  - Los conmutadores deben acordar qué conexiones deshabilitar en cada momento → **algoritmo distribuido**
- Decidir si reexpedir o no → **no es función de capa 2**

6.3 *Spanning tree* (II)



- ◯ Raíz: conmutador con el menor id. (único)
- — Puerto raíz de cada conmut.: el más cercano a raíz (empates → menor id.)
- — Conmut. designado de cada segmento de red: el más cercano a raíz (empates → menor id.)
- ··· Conmut. dejan de reexpedir desde/hacia puertos no raíz y segmentos no designados

6.3 *Spanning tree* (III)

- Cada conmutador tiene un id. obtenido concatenando un valor de prioridad (configurable) y su id. MAC (único)
- Inicialmente, todos los conmutadores creen ser raíz
- Quien cree ser raíz genera mensajes de configuración $\langle id\text{-}propio, id\text{-}raíz, distancia\text{-}a\text{-}raíz \rangle$
- Quien aprende que no es raíz (recibido $id\text{-}raíz < id\text{-}propio$) no crea más mensajes, pero sí reexpide ($distancia + 1$) los que llegan desde raíz

Como resultado:

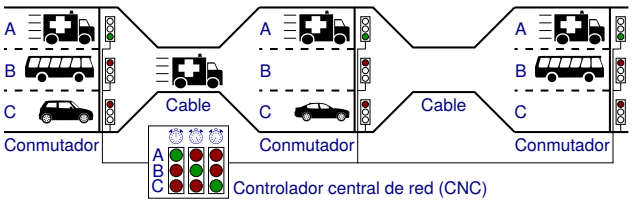
- Sólo el raíz genera mensajes de configuración
  - Si no se reciben durante cierto tiempo, se reinicia el proceso
- Todos aprenden cuáles son sus puertos redundantes

Inconveniente: **las tramas no siguen el camino óptimo**

6.4 Redes sensibles al tiempo (TSN)

Sustituye CSMA/CD por TSN (802.1Q) sobre cable ethernet

- Objetivo: integrar tráfico de tiempo real (🏭, ✈, 💓)
- Tramas etiquetadas con identificador de prioridad
- Colas por prioridad en cada interfaz
- Reloj compartido entre dispositivos
- Reexpedición de colas preprogramada (TDMA [p. 10])



7 Fiabilidad

Fiabilidad: garantía de que los datos se reciben correctamente, ordenados, sin pérdidas y sin duplicados

- **Control de errores** para detectar errores en tramas recibidas
- **Secuenciación de datos** para ordenar datos recibidos

Un protocolo ofrece un servicio de transmisión fiable si:

a) Detecta tramas erróneas y las retransmite
b) Detecta tramas perdidas y las retransmite
c) Descarta tramas duplicadas
d) Ordena las tramas que llegan desordenadas

8 Control de errores

- Valores típicos de *bit-error-ratio* (BER) (tramas $\approx 10^4$ bits):
  - Fibra óptica: $\sim 10^{-12}$
  - Cable de cobre: $\sim 10^{-9}$ (muy pocos errores)
  - Aire (medio inalámbrico): $\sim 10^{-5}$ (1/10 tramas erróneas)
- Detección: paridad, checksum, CRC, SHA1, etc.
  - Añade redundancia: alteraciones $\leftrightarrow$ inconsistencias
  - Sobrecarga «baja»
  - **Ninguna detección es 100 % fiable**
- Ante error detectado:
  - **Descartar** trama $\rightarrow$ no hay fiabilidad (e.g. Ethernet, IP, UDP)
  - **Retransmitir** trama $\rightarrow$ necesita secuenciación (e.g. TCP)
  - **Corregir** error $\rightarrow$ sobrecarga elevada (e.g. Hamming) pero interesante ante coste de retransmisión muy alto

8.1 Paridad

- Cada m bits se añade un bit de paridad: 1010100 **P**
- Paridad Par (P: $\oplus$): número total de unos es par: 1010100 **1**
- Paridad Impar (P: $\overline{\oplus}$): número impar de unos: 1010100 **0**
- Detecta cualquier número impar de bits erróneos
- Ejemplo:

Datos	Enviado ($\oplus$)	Recibido	Resultado	
100011	1000111	1000111	ok (unos par): 100011	✓
100011	1000111	0000111	error (unos impar)	✓
100011	1000111	0010111	ok (unos par): 001011	✗

8.1 Paridad (II)

Errores en ráfaga: secuencia de bits cuyo primer y último bit son erróneos

Paridad por bloques:

- Toma el mensaje como una matriz de m bits de ancho
- Calcula una nueva fila de m bits de paridad vertical
- Detecta errores en ráfaga de longitud $\leq m$

Paridad en 2 dimensiones:

- Caso particular donde cada fila de la matriz tiene a su vez un bit de paridad
- Permite corregir errores de 1 bit

8.2 Checksum

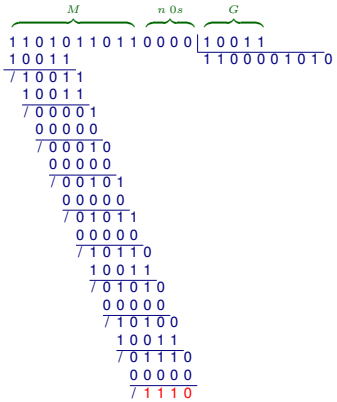
- Suma todo y comprueba si el resultado es correcto
  - E.g. emisor: $3+7+5+1+3=19$ (checksum: 19)
  - Comprobación: ¿$3+7+5+1+3-19=0$? ($\neq 0$: error)
- Toma el mensaje como secuencia de números de m bits
- Calcula una nueva fila de m bits como la suma de la secuencia de números
  - Cualquier tipo de suma sirve, aunque distintos tipos de suma tendrán distintas propiedades de detección
  - En TCP/IP: suma de enteros de 16 bits en complemento a 1 con inversión de bits en el resultado
  - Paridad por bloques: caso particular de checksum con $\oplus$ bit a bit como operación de suma
- Detecta errores en ráfaga de longitud $\leq m$

8.3 Cyclic Redundancy Check (CRC)

- CRC: resto de la división del mensaje
- Ejemplo: letra DNI = número DNI módulo 23 (con secuencia de restos T R W A G M Y F P D X B N J Z S Q V H L C K E)
- Dado un mensaje M y un polinomio generador G de grado n, generar un mensaje M' divisible por G
  - CRC más común: **aritmética finita binaria**: $+\equiv-\equiv\oplus$
  - G tiene grado n si tiene $n+1$ bits y su bit de más peso es 1
  - $CRC = (M \times 2^n) \mod G$ (añadir n 0s, dividir y coger resto)
  - $M' = M \times 2^n + CRC$ (concatenar M y CRC; CRC en «cola»)
- El receptor asume mensaje correcto si $M' \mod G = 0$
- Propiedades básicas de detección:
  - Número impar de errores si $G \mod 11_2 = 0$
  - Errores en ráfaga de longitud $\leq n$ si bit de menor peso es 1

8.3 Cyclic Redundancy Check (CRC) (II)

M: 1101011011
$(x^9+x^8+x^6+x^4+x^3+x+1)$
G: 10011 (x^4+x+1)
n: 4
CRC: 1110 (x^3+x^2+x)
M': 11010110111110
$(x^{13}+x^{12}+x^{10}+x^8+x^7+x^5+x^4+x^3+x^2+x)$



8.4 Corrección de errores

Distancia de *Hamming*: número mínimo de bits que deben cambiar en un código válido para convertirse en otro código válido

- La corrección de errores se basa en establecer códigos válidos lo suficientemente distantes como para que:
  1. Si hay error, se reciba un código no válido
  2. Al recibir un código no válido, el código válido más cercano (en distancia Hamming) sea el código original
- Con distancia d se pueden corregir $\lfloor (d-1)/2 \rfloor$ bits
- Ejemplo: 0000000000, 0000011111, 1111100000, 1111111111
  - Distancia de Hamming: 5
  - Puede corregir 2 bits, e.g. 0000010011 $\rightarrow$ 0000011111
- E.g. código Hamming, código Golay (NASA Voyager 1 y 2)

9 Secuenciación de datos

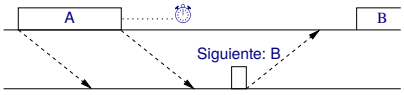
Basada en algoritmos ARQ (*Automatic Repeat reQuest*), donde el receptor solicita al emisor la siguiente trama que desea recibir en secuencia, lo cual confirma que las anteriores han llegado bien

Algoritmos:

- Parada y espera (*stop & wait*)
- Ventana deslizante
  - Vuelta atrás (*go-back-n*)
  - Repetición selectiva (*selective repeat*): más eficiente

Usados en ciertos protocolos de capas de enlace (enlace lógico punto-a-punto) y transporte (enlace lógico extremo-a-extremo)

9.1 Stop & wait

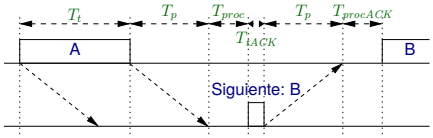


- Emisor envía trama con identificador i, inicia temporizador y espera solicitud de siguiente trama (ACK $i+1$)
  - Si llega ACK $i+1 \rightarrow$ parar temporizador y enviar trama $i+1$
  - Si vence el temporizador $\rightarrow$ reenviar trama i
- Receptor espera trama con identificador i
  - Si llega trama i correctamente $\rightarrow$ solicitar siguiente trama enviando ACK $i+1$ y esperar trama $i+1$
  - Si llega trama distinta de $i \rightarrow$ reenviar ACK i
  - Opcionalmente, si llega trama errónea $\rightarrow$ reenviar ACK i

9.1 Stop & wait (II)

Tiempo de ida y vuelta / *Round Trip Time* (RTT): tiempo necesario para enviar mensaje y recibir respuesta (segundos)

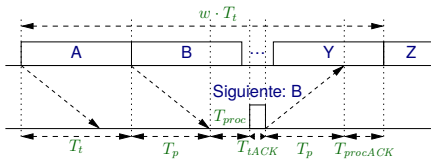
- RTT sin errores: transmisión $T_t = L/V_t$ + propagación $T_p = D/V_p$ + procesamiento T_{proc} de trama y ACK
- $RTT = T_t + T_p + T_{proc} + T_{tACK} + T_p + T_{procACK}$



- Utilización del enlace lógico (%): $U = 100 \cdot \frac{T_t}{RTT} \leq 100\,\%$

9.2 Ventana deslizante

- *Stop & wait* es poco eficiente si $T_t \not\gg T_p$
- Ventana deslizante permite enviar w tramas antes de bloquearse por esperar ACK:



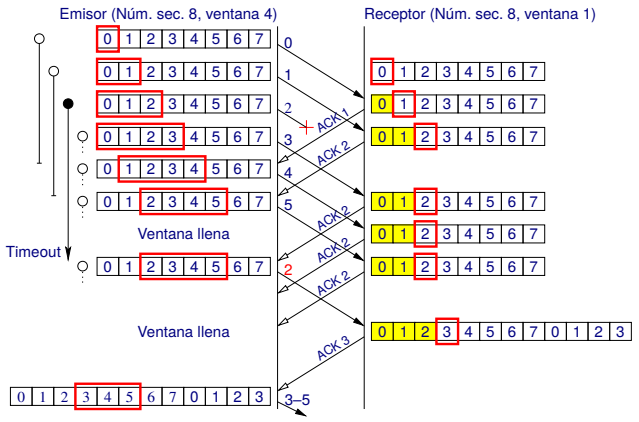
- Utilización del enlace lógico (%):

$$U = \begin{cases} 100\,\% & \text{si } w \cdot T_t \geq RTT \quad \checkmark \\ 100 \cdot \dfrac{w \cdot T_t}{RTT}\,\% & \text{si } w \cdot T_t < RTT \quad \times \end{cases}$$

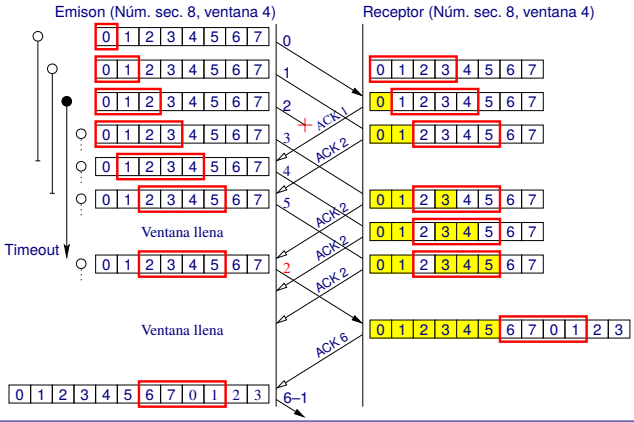
9.2 Ventana deslizante (II)

- Cada trama va etiquetada con un **número de secuencia** i
- El emisor mantiene una **ventana de emisión** de tamaño w con las tramas enviadas pero no confirmadas
- El receptor mantiene una **ventana de recepción** de tamaño w_r con «huecos» para tramas que está dispuesto a aceptar
- ACK i indica que se han recibido correctamente todas las tramas hasta la $i-1$ (incluida) y se espera la trama i

9.2 *Go-Back-n*: $w_r = 1$



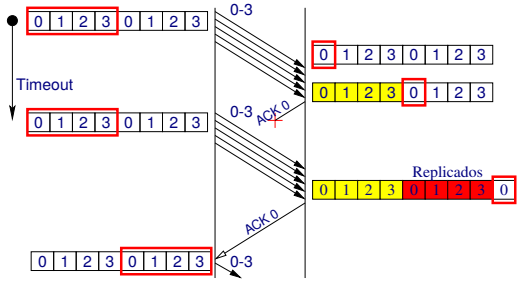
9.2 *Selective Repeat*: $w_r > 1$



9.2 Núm. secuencia y tamaño ventana

- Se necesitan $NS \geq w + w_r$ números de secuencia
  - $\lceil \log_2(NS) \rceil$ bits de «número de secuencia» en cabecera
- Contraejemplo *Go-back-n* ($w_r = 1$):

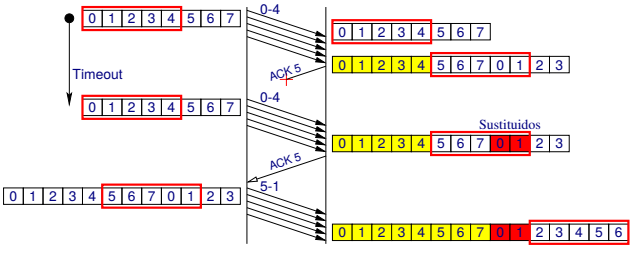
Emisor con $w = 4$ $NS = 4$ Receptor con $w_r = 1$



9.2 Núm. secuencia y tamaño ventana (II)

- $NS \geq w + w_r$ números de secuencia ($\lceil \log_2(NS) \rceil$ bits)
- Contraejemplo *Selective Repeat* ($w_r > 1$):

Emisor con $w = 5$ $NS = 8$ Receptor con $w_r = 5$



9.3 Ejemplo: HDLC

ISO High-Level Data Link Control (6 ó 7 bytes + datos)

1	1	1 ó 2	n	2	1
01111110	Dirección	Control	Datos	FCS	01111110

Dirección: Id. secundaria (no usada en punto-a-punto)

Control (8 bits): Número secuencia (3 bits), Núm. sec. esperada (3 bits), 2 bits control

Control (16 bits): Número secuencia (7 bits), Núm. sec. esperada (7 bits), 2 bits control

FCS: CRC de 16 bits

Créditos de material reutilizado

Imagen «Jersey Telecom switchboard and operator» (p. 7): Any purpose, attribution, Joseph A. Carr
Imagen «buzones de correos» (p. 7): CC BY Sergio
Imagen «furgoneta de correos» (p. 7): CC BY Dorieo
Imagen «cartero» (p. 7): PD Juhele
Imagen «Logo Wi-fi alliance» (p. 17): PD Wi-fi Alliance
Imagen «Band Aid Flat Icon Vector» (p. 26): CC BY VideoPlasty
Imagen «Clipart Coche» (p. 31): PD wellvieira
Imagen «Clipart Coche sedan» (p. 31): PD wellvieira
Imagen «Clipart Bus» (p. 31): PD
Imagen «Clipart Ambulancia» (p. 31): PD openclipart GDJ

Redes de Computadores

Tema 4 – Capa de red

Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción

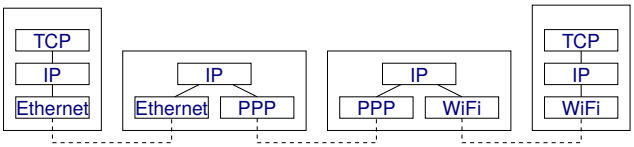
¿Por qué la capa 2 no sirve al aumentar el tamaño de la red?

- No puede interconectar redes físicas distintas
- Mensajes de difusión total (broadcast) saturarían la red
- Tamaño excesivo de tablas de reexpedición en conmutadores aprendices (una entrada por equipo)
- Rutas no óptimas (**spanning tree**)

1 Introducción (II)

Solución: pasar de red «física» (capa 2) a redes lógicas interconectadas → capa de red (capa 3)

- Abstrae diferencias de funcionamiento de redes físicas
- Todas las redes deben tener un protocolo común para:
  - Buscar caminos óptimos
  - Propagar mensajes hacia destino



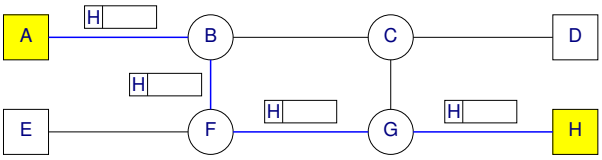
2 Modelos en conmutación de paquetes

Dependiendo de la información usada para decidir el camino, existen varios modelos dentro de la conmutación de paquetes:

- **Conmutación por datagrama**
  - En cada esquina preguntar a alguien por qué calle nos acercamos más al destino
- **Encaminamiento fuente**
  - Buscar la ruta y anotarla antes de iniciar el viaje para después seguirla
- **Conmutación por circuito virtual**
  - Especificar un destino a una agencia e iniciar el viaje con los trasbordos ya organizados
- Combinaciones de los modelos anteriores

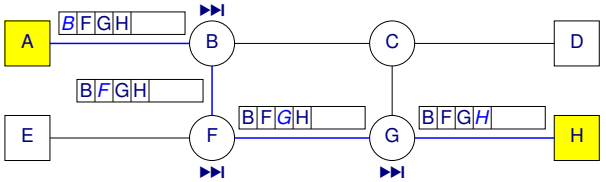
2.1 Conmutación por datagrama

- **Usado en Internet** en el protocolo IP
- Cada **encaminador** reexpide en función de la dirección destino de cada paquete
  - Debe conocer el camino a ¡cualquier destino! [p. 51]
  - Mantiene una **tabla de reexpedición** con esa información
  - La tabla de reexpedición de cada encaminador es distinta
- Cada paquete es reexpedido de forma independiente
  - Paquetes con igual origen-destino pueden ir por ruta distinta
  - **Alta tolerancia a fallos**



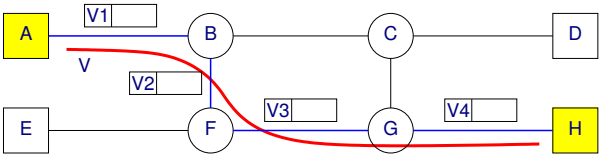
2.2 Encaminamiento fuente

- El propio paquete lleva la ruta a seguir
  - El emisor debe conocer la topología de la red
  - La información de ruta (campo «siguiente puerto») se modifica en cada encaminador (rotación, punteros, etc.)
  - La cabecera tiene un tamaño variable sin límite
- El protocolo IP permite usar encaminamiento fuente añadiendo ciertas opciones en la cabecera de los paquetes



2.3 Conmutación por circuito virtual

- Conmutación de circuitos vía software
  - Creación y destrucción explícita de circuitos (paquete inicial + paquete final)
  - Paquetes intermedios se reexpiden por mismo circuito
  - La creación permite reservar recursos (QoS) [Tema 6]
- Cada encaminador mantiene **tabla de circuitos virtuales**
- Si un encaminador o un enlace falla, el circuito también
- E.g. X.25, Frame relay, ATM



2.4 Combinación de modelos

Multi Protocol Label Switching (MPLS)

- Sobre conmutación por datagrama, los encaminadores solicitan que para ciertos destinos los paquetes incorporen ciertas etiquetas
- Necesita protocolo IP para crear rutas
- Es capaz de llevar cualquier protocolo de la capa de red
- Situado conceptualmente entre capas 2 y 3

3 Encaminadores

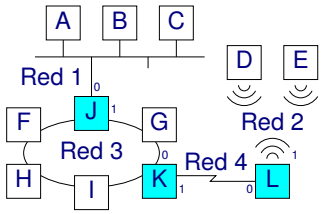
- Encaminador/*router*  : conmutador de capa de red
  - Interconecta (capa 3) redes (capa 2): inter-net
  - Trabaja con direcciones lógicas (dir. IP)
  - Varias interfaces o puertos hw., **distintos tipos de redes**
  - Conoce ruta óptima hacia cada red destino
  - Sólo reexpide hacia donde corresponde
  - Nunca reexpide tramas broadcast (MAC: ff:ff:ff:ff:ff:ff)
- Cada puerto: dir. lógica (IP) e id. físico (MAC)
- Ejemplo router doméstico: Ethernet + PON (fibra) + WiFi

3.1 Reexpedición desde encaminador

- Cada encaminador tiene en una tabla una **lista de destinos** y cómo acercarse/llegar a ellos
- Al recibir un paquete con dir. destino x, la busca en su tabla
  - Si está directamente conectado a la red destino → reexpedir hacia la estación destino x por interfaz correspondiente
  - Si no → reexpedir hacia siguiente encaminador según la tabla (sig. salto)

Tabla de reexpedición de **K**

Destino	Sig. salto	Interfaz
Red 1	J	0
Red 2	L	1
Red 3	—	0
Red 4	—	1

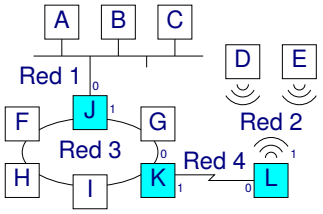


3.2 Reexpedición desde estación

- Cada estación tiene en una tabla su red y su **encaminador (*gateway*) por defecto**
- Para enviar un paquete hacia dir. destino:
  - Si está en la propia red → enviar a estación destino
  - Si no → enviar a encaminador por defecto
- ¡Mismo procedimiento que encaminador!

Tabla de reexpedición de **A**

Destino	Sig. salto	Interfaz
Red 1	—	0
default	J	0



3.3 Ejemplo tablas

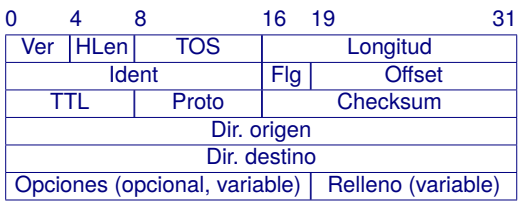
```
lab000:~$ /sbin/route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
155.210.152.0   *               255.255.255.0   U     0      0        0 br0
192.168.122.0   *               255.255.255.0   U     0      0        0 virbr0
link-local      *               255.255.0.0     U     1003   0        0 br0
default         155.210.152.254 0.0.0.0         UG    0      0        0 br0

lab000:~$ ip route
155.210.152.0/24 dev br0  proto kernel  scope link  src 155.210.152.177
192.168.122.0/24 dev virbr0  proto kernel  scope link  src 192.168.122.1
169.254.0.0/16 dev br0  scope link  metric 1003
default via 155.210.152.254 dev br0
```


4 Protocolo IP

- Internet Protocol (IP)
- Conmutación por datagrama [p. 6]
- Servicio **no fiable** (*unreliable*)
  - La red hace lo posible para la entrega de los paquetes, pero sin garantizarlo (*best effort*)
  - Pueden perderse paquetes
  - Pueden llegar desordenados respecto al envío
  - Se pueden entregar paquetes duplicados
  - El tiempo de entrega puede ser muy variable
- Actualmente convive la versión 4 del protocolo IP con IPv6 [p. 43]

4.1 Cabecera IPv4



Versión: 4, desde comienzos de los años 80

Hlen: longitud de la cabecera, medida en palabras de 32 bits (usualmente 5; 20 bytes)

Type Of Service (TOS): calidad de servicio (QoS) [Tema 6]

Longitud: Longitud total del paquete, en bytes

- 16 bits → paquetes de tamaño < 64 KiB (2^{16})

4.1 Cabecera IPv4 (II)

Identificador, Flags, Offset: campos para fragmentación [p. 17]

Time-To-Live: contador de saltos (encaminadores atravesados)

Protocolo: tipo de datos que contiene el paquete (TCP, UDP, ICMP, etc.); demultiplexor para capa superior

Checksum: verificación de la cabecera, no de datos

Direcciones: IP origen y destino, de 32 bits cada una

Opciones:

- Pedir que los encaminadores atravesados añadan información al paquete (su IP, la hora, etc.)
- Encaminamiento fuente: lista de encaminadores que pueden/deben ser atravesados
- etc.

Relleno/*Pad*: para que los datos empiecen en una posición múltiplo de 32 bits

4.2 Fragmentación y reensamblado

Maximum Transmission Unit (MTU): tamaño máximo de los datos de una trama
- Depende del protocolo de la capa de enlace, e.g.: 1 500 B en 802.3, 2 304 B en 802.11, 4 464 B en 802.5

- Un paquete generado en una red puede no caber en otra

Paquete IP	Cab. IP	Datos

Trama	Cab.	Datos (longitud $\leq$ MTU)

- Estrategia IPv4:
  - Fragmentar si MTU $<$ longitud del paquete IP
  - Fragmenta el encaminador directamente conectado a la red que requiere tramas más pequeñas
  - Es posible fragmentar fragmentos
  - Reensambla la estación destino

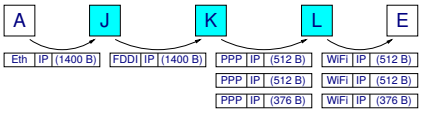
4.2 Fragmentación y reensamblado (II)

Ident (16 bits): identificador de paquete (igual en todos los fragmentos)

Flags (3 b): reservado (0), prohibido fragmentar, siguen más fragmentos

Offset (13 b): posición de los datos en el paquete original (unidades de 8 B)

Ejemplo fragmentación sobre red pág. 11:



Inicio cabecera
Ident x | 0 | 0 | 0 | Offset 0
Resto cabecera
1400 bytes datos

⇓

Inicio cabecera
Ident x | 0 | 0 | 1 | Offset 0
Resto cabecera
512 B datos ($/8 = 64$)

Inicio cabecera
Ident x | 0 | 0 | 1 | Offset 64
Resto cabecera
512 bytes datos

Inicio cabecera
Ident x | 0 | 0 | 0 | Offset 128
Resto cabecera
376 bytes datos

4.3 ICMP

- Internet Control Message Protocol
- Aunque IP sea no fiable, se notifican incidencias al emisor
  - Quien detecta incidencia (encaminador o destino) causada por un paquete IP crea mensaje ICMP dirigido a origen
  - No se notifican incidencias respecto a mensajes ICMP
- ICMP encapsulado dentro de un paquete IP
- Uso adicional: diagnóstico de red
  - ping: alcanzabilidad de destino y latencia
  - traceroute: ruta entre origen y destino

4.3 ICMP (II)

0	8	16	31
Tipo	Código	Checksum	
Resto cabecera			
Datos (>28 bytes)			

Tipo/Código (2 B): tipo/subtipo de mensaje ICMP, por ejemplo:
- Echo request (ping) / Echo reply (pong)
- Redirect (e.g. desde encaminador a estación fuente cuando hay otro encaminador mejor)
- Destino inalcanzable (red/estación/protocolo/puerto)
- TTL excedido (cuando se descarta por TTL=0)
- Error de checksum
- No se puede fragmentar / reensamblar

Checksum (2 B): del mensaje ICMP (cabecera + datos)

Resto cabecera (4 B): depende del tipo de mensaje ICMP

Datos: primeros bytes del paquete causante de la incidencia

4.4 Procesado de un paquete IP

Al recibir una trama, el encaminador debe:
1. Validar trama (checksum/crc/etc.)
2. Validar cabecera IP (comprobar checksum)
3. Procesar opciones de la cabecera (si las hay)
4. Buscar dirección destino en tabla de reexpedición
5. Decrementar TTL
6. Realizar fragmentación (si es necesario)
7. Calcular checksum (por cada fragmento)
8. Construir trama de capa inferior (cabecera, crc, etc.)
9. Transmitir a siguiente salto por interfaz correspondiente
10. Generar y enviar paquete ICMP (si es necesario)

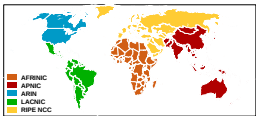
El tiempo de procesado en cada salto **no es despreciable**

5 Direcciones IPv4

- Direcciones de 32 bits:

	8	8	8	8
Dirección:	X	X	X	X
Interpretación decimal:	0-255.	0-255.	0-255.	0-255

- Globalmente únicas, excepto privadas
- Jerárquicas: id. red + id. interfaz
- IANA (Internet Assigned Numbers Authority) reparte bloques de direcciones IPv4 a Registros Regionales de Internet (RIR) bajo demanda → **todos ya repartidos**
  - AFRINIC: África
  - APNIC: Asia-Pacífico
  - ARIN: EE.UU.-Canadá
  - LACNIC: Lat. América-Caribe
  - RIPE NCC: Europa-Asia occid.



- DNS traduce nombres a direcciones IP [Tema 7]

5.1 Jerarquía red-interfaz

- Necesito redes grandes (muchos bits interfaz) y pequeñas
- Especificar cuántos bits de red tiene cada red (**CIDR**)

1	Red ← →	Interfaz	32
10011011 11010010	00000000 00000000		155.210.0.0/16

- Por defecto / históricamente: clases fijas

Clase	Prefijo	1er byte	Nº redes	Nº dir./red	Uso
Clase A	0	0-127	128	16777216	Unicast
Clase B	10	128-191	16348	65536	Unicast
Clase C	110	192-223	2097152	256	Unicast
Clase D	1110	224-239			Multicast
Clase E	1111	240-255			Experimental

A: 7		B: 14		16	C: 21		8
0	Red	10	Red	Interfaz	110	Red	Interfaz
E.g.: 10.3.2.4		E.g.: 128.96.33.81			E.g.: 192.12.69.77		

5.2 Direcciones especiales

Dirección de red: todo 0s en bits de interfaz, e.g.: 128.96.0.0

Difusión/broadcast: todo 1s bits de interfaz, e.g.: 192.12.69.255
- Direcciones unicast posibles en una red: $2^{\text{nº bits interfaz}} - 2$

Bucle (*loopback*): 127.0.0.0/8; el propio nodo (127.0.0.1)

0.0.0.0: no válida; uso dependiente del contexto

Privadas: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
- No únicas globalmente → no válidas en Internet
- Únicas a nivel local → válidas para redes privadas
- Para acceder a Internet necesitan traducción de direcciones (NAT) [p. 36]

Link-local: 169.254.0.0/16; como privadas pero autoasignadas; el equipo debe verificar que no esté en uso
- Tráfico hacia privadas/link-local **no sale de la red local**

5.3 Correspondencia IP-MAC

- La capa de red funciona mediante direcciones IP
- La capa de enlace funciona mediante identificadores MAC

¿Cuándo necesitamos asociar ambas?

- Estación necesita id. MAC de su encaminador por defecto y de las estaciones de su red
- Encaminador necesita id. MAC de siguiente salto
- Encaminador final necesita id. MAC de estación destino

Asociación mediante tablas ARP:

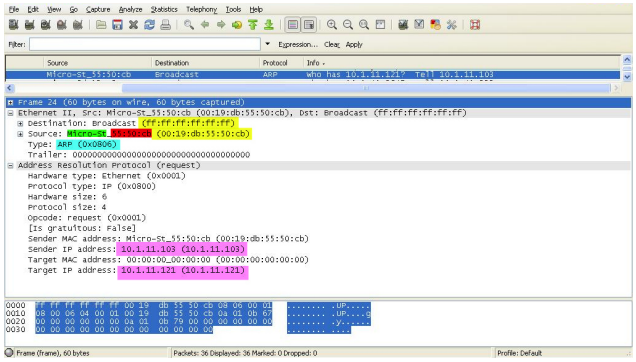
```
lab000:~$ arp
Address                  HWtype  HWaddress           Flags Mask  Iface
hendrix02.cps.uniza      ether   00:14:4f:ec:4d:54   C             br0
155.210.152.254          ether   f0:f7:55:f3:c7:c1   C             br0
camposancos.cps.uni      ether   00:1c:c0:ef:8d:8d   C             br0
```


5.4 Address Resolution Protocol (ARP)

- Mensajes de petición y respuesta sobre capa de enlace
  1. Si dir. IP no está en tabla, petición ARP por difusión
  2. Estación objetivo actualiza su tabla con direcciones origen (IP-MAC) de petición recibida
  3. Estación objetivo responde con su id. MAC
  4. Estación origen actualiza su tabla con IP-MAC de respuesta
- No se añaden entradas por ningún otro procedimiento
- Entradas de tabla son eliminadas tras un tiempo sin usarse

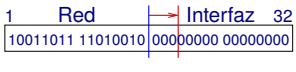
	Petición ARP	Respuesta ARP
ARP	¿Quién tiene 10.1.11.121? Soy: 10.1.11.103 00:19:db:55:50:cb	¡Hola 10.1.11.103! Soy: 10.1.11.121 03:ca:4b:2c:13:8a
Eth	Orig: 00:19:db:55:50:cb Dest: ff:ff:ff:ff:ff:ff	Orig: 03:ca:4b:2c:13:8a Dest: 00:19:db:55:50:cb

5.4.1 Ejemplo ARP



5.5 Subredes

- Problemas en redes grandes
  - Broadcast excesivo, colisiones, retardos, pérdida de tramas
- Solución: añadir nivel de jerarquía red-**subred**-interfaz
  - Se usan **bits de interfaz** para identificar subredes
  - Cada subred necesita dir. red, dir. broadcast y encaminador
  - Fuera de una red no hay consciencia de sus subredes (los encaminadores externos propagan hasta la red)
- Gestionadas mediante **máscaras CIDR**



155.210.0.0/16 fuera
155.210.0.0/19 dentro

- Gestionadas mediante **máscaras de red**
  - Teóricamente se permiten bits de subred no contiguos pero en la práctica ya no se permiten

5.5.1 Máscaras de red

- Información total & Máscara de bits = Información útil

- Dirección IP & Máscara de subred = Dirección de subred

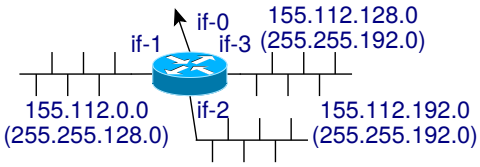
Clase B | Bits red Clase B | Bits interfaz Clase B

IP 155.210.153.238 - 10011011.11010010.10011001.11101110
Mask 255.255.248.0 - 11111111.11111111.11111000.00000000
Sub 155.210.152.0 - 10011011.11010010.10011000.00000000

Bits subred | Bits interfaz

- 155.210.153.0 & 255.255.248.0 ≡ 155.210.153.0/21

5.5.2 Ejemplo subredes



$ /sbin/route:

Destination	Gateway	Genmask	Iface
155.112.0.0	*	255.255.128.0	if-1
155.112.128.0	*	255.255.192.0	if-3
155.112.192.0	*	255.255.192.0	if-2
default	<sigsalto>	0.0.0.0	if-0

$ ip route:

```
155.112.0.0/17 dev if-1          # 155.112.0xxxxxxx.X
155.112.128.0/18 dev if-3        # 155.112.10xxxxxx.X
155.112.192.0/18 dev if-2        # 155.112.11xxxxxx.X
default via <sigsalto> dev if-0
```


5.5.3 Ejercicio subredes

✍ Para la red 155.210.0.0/16, ¿son válidas estas máscaras?

a) 255.252.0.0 (11111111.111111**00**.00000000.00000000)

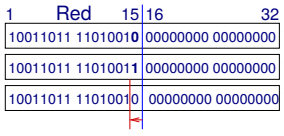
b) /12 (11111111.1111**0000**.00000000.00000000)

c) 255.255.216.0 (11111111.11111111.11**0**11000.00000000)

d) 255.255.252.0 (11111111.11111111.11111100.00000000)

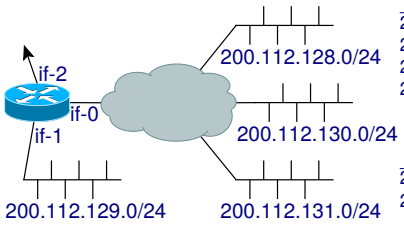
5.6 Superredes

- Los encaminadores «necesitan» una entrada en la tabla por cada red destino y hay más de 2 millones de redes
- Problema: con tantas entradas las tablas no son eficientes
- Objetivo: juntar varias redes contiguas en una única entrada en tabla si tienen el mismo siguiente salto e interfaz
- Gestionado con **máscara CIDR**
  - *Classless Inter-Domain Routing*
  - Ejemplo:



155.210.0.0/16
155.211.0.0/16
155.210.0.0/15 en tabla

5.6.1 Ejemplo CIDR



Destino	Interfaz
200.112.128.0/24	if-0
200.112.129.0/24	if-1
200.112.130.0/24	if-0
200.112.131.0/24	if-0
default	if-2

Destino	Interfaz
200.112.129.0/24	if-1
200.112.128.0/22	if-0
default	if-2

- if-0: 200.112.128.0, .130.0, .131.0 ($100000\frac{0}{1}\frac{0}{1}$)
  - 22 bits de red comunes: 200.112.100000XX.X
  - Prefijo común: 200.112.**128.0/22**
  - Agrupar como entrada única en tablas
- if-1: 200.112.129.0 (10000001) más prioritaria en tabla por estar «incorrectamente» incluida en 200.112.128.0/22

5.7 ¿Cómo usar menos direcciones IP?

- Problema: agotamiento de direcciones IPv4 unicast
- Últimas redes repartidas como bloques /CIDR, **sin clase**
- Servidores necesitan estar localizables en todo momento para recibir peticiones
  - IP **estática** o fija: dirección IP que no cambia con el tiempo
- Clientes no necesitan estar localizables
  - IP **estática**, si hay para todos
  - IP **dinámica**: dirección IP que puede cambiar con el tiempo
- Asignación dinámica desde un servidor (DHCP, PPP)
  - Al apagar el equipo, otro puede usar su IP
  - Al reiniciar el equipo, se puede usar una IP distinta
  - Una dirección IP nunca es usada por varios equipos a la vez
- Compartir direcciones IP unicast (NAT)
  - Una dirección IP es usada por varios equipos a la vez

5.7.1 DHCP

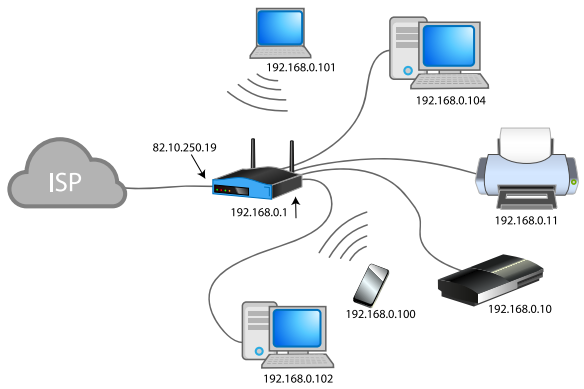
- Dynamic Host Configuration Protocol
- Un servidor DHCP proporciona información para que los equipos de su LAN configuren la red:
  - Dirección IP (por un tiempo limitado y no necesariamente la misma IP siempre)
  - Máscara de subred
  - Encaminador por defecto
  - Servidor de nombres [Tema 7]
  - etc.
- Pasos de configuración:
  1. Al arrancar, el equipo busca un servidor DHCP (IPsrc 0.0.0.0, IPdest 255.255.255.255, UDP:67)
  2. El servidor anuncia su presencia (IPdest 255.255.255.255)
  3. El equipo pide datos al servidor
  4. El servidor proporciona los datos

5.7.2 Network Address Translation

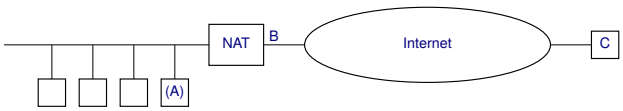
1. Uso de direcciones IP privadas en equipos locales
   - 10.X.X.X, 172.0001XXXX.X.X, 192.168.X.X
   - Válidas localmente pero NO globalmente
2. Enmascarar tras IP válida con encaminador NAT
   - Network Address (and port) Translator
   - El encaminador tiene una dirección IP unicast válida
   - Salida: NAT sustituye IP origen por la propia
   - Entrada: NAT deshace la sustitución; deduce a quién entregar el paquete

- **Solución arquitectónicamente mala**
  - Modifica el funcionamiento básico de la capa de red (el NAT cambia direcciones en paquetes)
  - Implica restricciones de funcionalidad
- Carrier-grade NAT / Large-scale NAT: NAT dentro de NAT
  - 100.64.0.0/10 para CGN

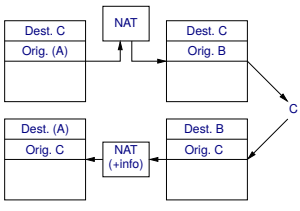
5.7.2 Network Address Translation (II)



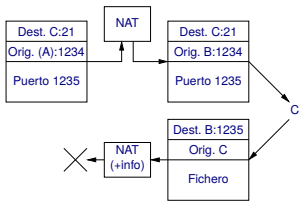
5.7.2 Network Address Translation (III)



Ejemplo «funcional»

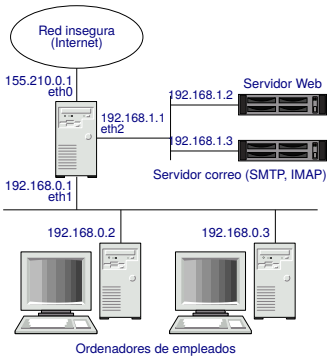


Ejemplo FTP «problemático»



5.7.3 Ejemplo cortafuegos + NAT

Ej. de manipulación de paquetes:



- Redirigir paquetes SMTP e IMAP al servidor de correo (NAT)
- Redirigir paquetes web, con un límite de 1 por segundo y ráfagas de 10 como máximo, al servidor web (NAT+filtro)
- Descartar todo el tráfico con protocolo distinto de TCP (filtro)
- Hacer de NAT a los ordenadores de empleados
- Inhabilitar el uso de la red en la propia máquina (filtro)
- Descartar conexiones desde 192.168.0.3 a puertos 6667 del exterior (filtro)

5.8 Multidestino

- Múltiple unicast es ineficiente
- Dirección IP asociada a un conjunto de equipos
  - Clase D (224-239.X.X.X) en IPv4
  - El emisor envía un paquete al grupo y la red se encarga de que llegue a **todos** los integrantes del grupo
  - E.g. *Discovery Max* en Movistar+: rtp://239.0.0.32:8208
- Necesita soporte de multidestino en encaminadores
  - Estaciones dicen a su encaminador que quieren unirse a un grupo multidestino mediante IGMP (Internet Group Management Protocol) en IPv4
  - Para multidestino, encaminadores pueden tener múltiples «siguiente salto» en tabla de reexpedición
  - Búsqueda/optimización de rutas [p. 51] extendida para multidestino o protocolos específicos (PIM)

6 Túneles

- Encapsulación de capa de red dentro de capa de red

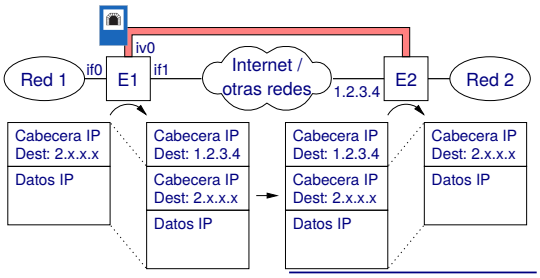


Tabla de reexpedición de E1:

Destino	Interfaz
Red 1	Interfaz 0
Red 2	Interfaz virtual 0
Por defecto	Interfaz 1

6.1 Redes privadas virtuales

VPN (*Virtual Private Network*): túnel+cifrado

- VPN unizar: https://remoto.unizar.es
  - Proporciona cifrado hasta encaminador de entrada a unizar
  - En unizar, paquetes tienen IP origen unizar (155.210.0.0/16)
  - No permite salida a Internet

- VPNs comerciales:
  - Cifrado hasta cierto encaminador (salida a Internet)
  - En Internet, paquetes tienen IP origen del proveedor VPN
  - Pueden permitir la selección del encaminador de salida

7 IP versión 6

- Propuesta inicial en 1991 (IETF IPng: IP Next Generation)
- Nuevo protocolo → requiere actualizar encaminadores
- Intención de transición progresiva, que se ha ido retrasando «forzando» IPv4
- Direcciones de 128 bits jerárquicas (red+interfaz) sin clases
- Configuración automática
  - Configuración con estado (*stateful*): DHCPv6
  - Configuración sin estado (*stateless*): Petición Id. (Sub)Red
- Direcciones «al más cercano» (*anycast*)
  - E.g. al encaminador más cercano
- Al no haber NAT es posible tener seguridad en capa de red
  - Implementación obligatoria de IPsec
- No hay fragmentación (ICMP notifica tamaño excesivo)

7.1 Cabecera IPv6

Cabecera de tamaño fijo (40 bytes):

- No hay checksum ni campos fragmentación ni opciones
- Campos ligeramente modificados: TOS → TrafficClass, Length → PayloadLength, Protocol → NextHeader, TTL → HopLimit
- Nuevo campo *FlowLabel*
- Opciones → Cabeceras de extensión (NextHeader)

7.1 Cabecera IPv6 (II)

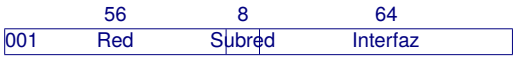
Cabeceras de extensión (ejemplo):

IPv6 Header NextHeader= Security	**Security Header** NextHeader= Fragmentation	**Frag. Header** NextHeader= TCP	TCP Header	DATA

- Enlazadas con campos NextHeader hasta capa superior
- Procesadas por nodo destino (excepto extensión *Hop-by-Hop*)
- Permite añadir funcionalidad sin cambiar protocolo y sin modificar encaminadores
- E.g. salto a salto, fragmentación, autentificación, encapsulado de seguridad de la carga útil

7.2 Direcciones IPv6

- 8 bloques de 4 dígitos hexadecimales separados por ':'
  - 2001:0DB8:0000:0000:0000:0000:A456:0024
  - Bloques contiguos de 0s y 0s a la izquierda se pueden omitir 2001:DB8::A456:24
  - Desambiguación de puerto: http://[2001:DB8::A456:24]:80/
- Espacio de direccionamiento diseñado para **uso disperso**
  - Recomendación: /56 para usuarios finales
    - En cada casa, más direcciones (2^{72}) que en Internet IPv4



  - No es desaprovechar, es decisión de diseño
  - No hay que numerar :1, :2, :3, sino de forma arbitraria
- Identificador de interfaz asignado (DHCP), autoconfigurado a partir de MAC, aleatorio (privacidad) o manual

7.2 Direcciones IPv6 (II)

Algunas direcciones y prefijos (/CIDR):

::1/128: Bucle local

2000::/3 (001): Unicast global (por ahora)

2001:DB8::/32: Específica para ejemplos

FE80::/10 (1111111010): Enlace local

::FFFF:X:X/96: IPv4 mapeada (::FFFF:128.96.33.81)

FF00::/8 (11111111): Multicast, e.g.:

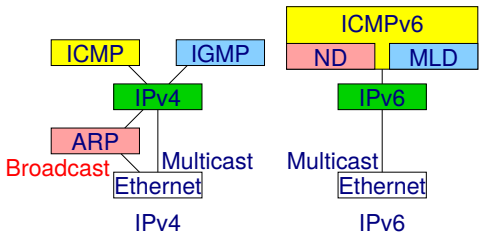
- FF02::1: todos los nodos (difusión)
- FF02::2: todos los encaminadores
- FF02::B: todos los agentes móviles
- FF05::1:3: todos los servidores DHCP

Anycast: Sin prefijo, para poder usarlas para cualquier red

- X:X:X:X:X:X:X:0000: Encaminador más cercano

7.3 Esquema de protocolos

- ARP → (SEcure) Neighbor Discovery (SEND/ND) en ICMPv6
- Multidestino (Multicast Listener Discovery, MLD) en ICMPv6
  - Broadcast es un caso particular de multidestino



7.4 Ejercicio de direcciones

✍ Busca la configuración de los equipos de tu casa. Para cada uno de ellos (excepto el encaminador):

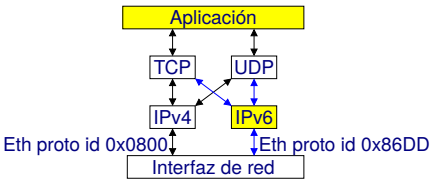
- ¿Cuál es su dirección IP? ¿Tiene IPv4, IPv6 o ambas?
- ¿Es pública (¿de qué clase?) o privada?
- ¿Está en alguna subred?
- ¿Está configurado manualmente o vía DHCP?

Para el encaminador:

- ¿Qué dirección IP tiene cada uno de sus interfaces?
- ¿Tiene IPv4, IPv6 o ambas?
- ¿Cuáles son públicas? ¿De qué clase?
- ¿Es también un servidor DHCP?
- ¿Es también un NAT?

7.5 Transición IPv4 → IPv6

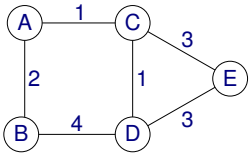
- Imposible cambio instantáneo IPv4 a IPv6
- Despliegue IPv6 incremental, conviviendo ambas versiones
- Operación en modo doble pila de protocolos
  - Nodos tienen protocolo IPv4 e IPv6
- Túneles [p. 41] / MPLS [p. 9]
  - Enviar paquetes IPv6 encapsulados en IPv4 o viceversa
  - Permite comunicar redes IPv6 a través de caminos no-IPv6
  - E.g. Hurricane Electric (http://tunnelbroker.net/)



8 Búsqueda de caminos

Objetivo: dada una red, encontrar el camino con el menor coste entre cada par de nodos, en base a:

- Topología (grafo donde cada red es un nodo)
- Métricas de coste en enlaces
  - Coste constante, e.g. capacidad enlace, nº saltos, etc.
  - Factor dinámico, e.g. nº paquetes en cola, retardo medio últimos minutos, etc.
  - Factores económicos, e.g. precio por transmisión
- Con las tablas de caminos, cada encaminador construye su tabla de reexpedición



8 Búsqueda de caminos (II)

Dos algoritmos:

- Distancia-vector (algoritmo «Bellman-Ford»)
  - Cada nodo propaga, **solo a sus vecinos**, **todo lo que sabe de la red**
  - Problemas de convergencia: la información recibida puede estar obsoleta y ser inválida
  - RIP (LANs), IGRP (WANs pequeñas), EIGRP (WANs), BGP (Internet)
- Estado-enlace (algoritmo «Dijkstra»)
  - Cada nodo difunde, **a todos los nodos**, **solo información de sus enlaces**
  - Convergencia rápida
  - Requiere más memoria y CPU que distancia-vector
  - OSPF (LANs, WANs pequeñas), IS-IS

8.1 Estado de enlace

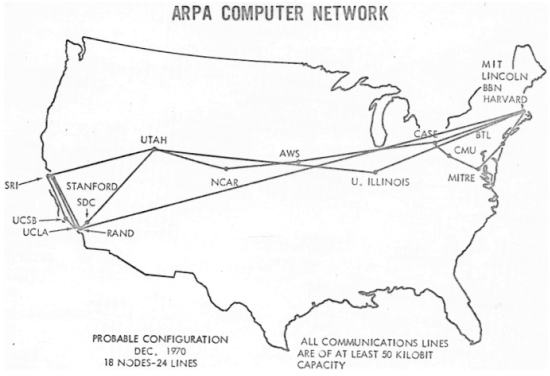
- Cada actualización contiene información sobre los **enlaces directamente conectados** (no toda la tabla de encam.)
- Envia actualización de estado a **todos los nodos** (no solo a los vecinos inmediatos) mediante inundación fiable

Se aplica en cada nodo para calcular la mejor ruta hacia otros:

1. Confirmar camino con distancia 0 al propio nodo
2. Crear lista vacía de posibles caminos y costes
3. Para el nuevo nodo confirmado:
   - 3.1 Añadir a la lista los nodos directamente conectados al nuevo nodo confirmado y el coste para llegar a ellos pasando por él
   - 3.2 Si hay varios caminos para llegar a un nodo, descartar los más costosos
   - 3.3 Confirmar el nodo con menor coste y quitarlo de la lista

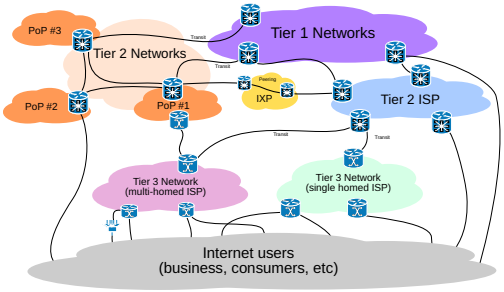
9 Estructura de Internet

- Inicios: centros de investigación interconectados



9 Estructura de Internet (II)

- Grandes empresas y compañías telefónicas (ISPs) fueron interconectando sus redes
- Ahora es una estructura jerárquica operada por proveedores comerciales



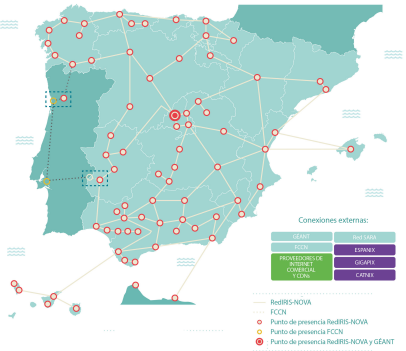
9.1 Sistemas autónomos

Sistema Autónomo (SA/AS): grupo de redes IP que poseen una política de rutas propia e independiente

- Un SA es un dominio administrativo independiente
- Se asigna a cada SA un número de 32 bits (ASN)
- Ejemplos: gran compañía, red troncal/intercontinental
- **Gestión de caminos en dos niveles**: dentro/fuera de un SA
  - Dentro: optimizar caminos entre las redes que contiene
  - Fuera: buscar caminos que comuniquen los distintos SAs
- Ventajas por trabajar con subconjuntos de Internet:
  - **Escalabilidad**: que al crecer todo siga funcionando
  - **Eficiencia**: poco tráfico de configuración
  - **Tolerancia a fallos**: buscar rápido otras rutas en caso de fallo de un enlace o de un encaminador

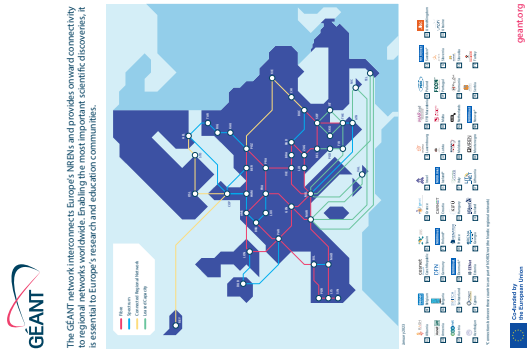
9.1 Sistemas autónomos (II)

- E.g. RedIRIS (SA 766) agrupa las redes de las universidades y centros de investigación en España



9.1 Sistemas autónomos (III)

- E.g. SA troncal: GÈANT (20965) comunica los SAs de investigación europeos



9.2 Dominio interior

Pasarela/dominio interior (interno a cada SA):

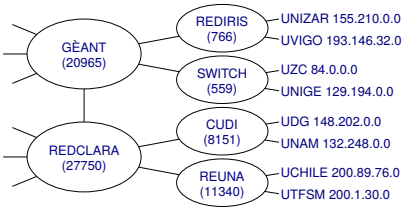
- Topología conocida
  - Estación conoce encaminador por defecto (local de subred)
  - Encam. locales conocen encaminador(es) de su red
  - Encaminadores de cada una de las redes del SA se conocen entre sí y conocen encaminadores de salida del SA (encaminadores frontera)
- Gestión de caminos personalizada
  - Gestionar «pocas» redes permite la búsqueda de **caminos óptimos** entre ellas
  - Todos los encaminadores del SA deben usar el mismo protocolo de búsqueda de caminos
    - RIP: Routing Information Protocol
    - OSPF: Open Shortest Path First
    - Etc.

9.3 Dominio exterior

Pasarela/dominio exterior (comunicación entre SAs):

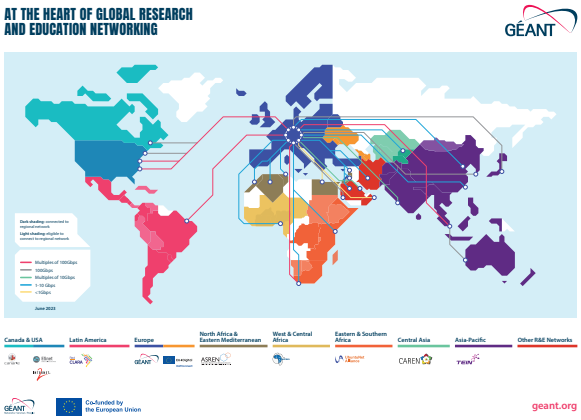
- Todos los SAs deben usar el mismo protocolo:
  - BGP-4 (*Border Gateway Protocol 4*)
- Cada SA tiene encaminadores frontera que anuncian:
  - Redes internas
  - Redes externas alcanzables (sólo en SA de tránsito)
  - Información de caminos
- Elección de rutas basada en políticas explicitas (preferencias basadas en precio, etc.)
- Para poder atravesar ciertos SAs y evitar otros, BGP trabaja con rutas completas, no solo el siguiente salto

9.4 Ejemplo BGP



- Speaker de REDIRIS anuncia alcanzabilidad a sus redes (UNIZAR, UVIGO…)
  - Redes 155.210.0.0 y 193.146.32.0 pueden alcanzarse directamente desde REDIRIS
- Speaker de GÈANT anuncia alcanzabilidad a REDIRIS, SWITCH…
  - Redes 155.210.0.0 y 193.146.32.0 pueden alcanzarse siguiendo el camino (SA 20965, SA 766)

9.4 Ejemplo BGP (II)



Créditos de material reutilizado

Imagen «Encaminador doméstico» (p. 10): CC BY SA Raimond Spekking (fondo eliminado)
Imagen «Cisco CRS-1» (p. 10): CC BY SA Cisco Systems Inc.
Imagen «Regional Internet Registries world map» (p. 22): CC BY SA Dork, Canuckguy et al.
Imagen «Wireshark ARP» (p. 27): CC BY SA Mart0045 (bordes eliminados)
Imagen «Máscara binaria de imagen» (p. 29): PD Ricardo Cancho Niemietz
Imagen «Band Aid Flat Icon Vector» (p. 34): CC BY SA VideoPlasty
Imagen «CPT-NAT-1» (p. 37): CC BY SA Milesjpool
Imagen «Señal de túnel» (p. 41): PD
Imagen «ARPANET 1970 Map» (p. 54): CC BY SA UCLA and BBN
Imagen «Internet Connectivity Distribution and Core» (p. 55): CC BY SA Ludovic.ferre
Imagen «Puntos de presencia RedIRIS» (p. 57): CC BY SA Gobierno de España
Imagen «Gèant topology map, Jan. 2023» (p. 58): Licencia no especificada, con permiso para reutilizar, geant.org, Union Europea
Imagen «Gèant global connectivity map, Jun. 2023» (p. 62): Licencia no especificada, con permiso para reutilizar, geant.org, Union Europea

Redes de Computadores

Tema 5 – Capa de transporte

Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción

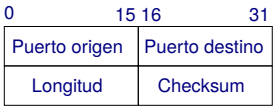
- Función: control de la comunicación extremo a extremo
- Se asume un protocolo de capa de red que puede ser:
  - No fiable (paquetes perdidos/duplicados/desordenados)
  - Paquetes limitados a un tamaño finito (MTU)
  - Paquetes recibidos con retardo arbitrariamente largo
- Posibles servicios extremo a extremo:
  - Fiabilidad
  - Permitir mensajes de tamaño arbitrariamente grande
  - Permitir al receptor controlar el flujo de datos del emisor
  - Reserva de recursos (QoS)
  - Otros

1 Introducción (II)

- UDP (User Datagram Protocol)
  - Protocolo de transporte bidireccional «sin conexión»
  - No garantiza un servicio extremo a extremo fiable
  - Uso más común: retransmisión continua de información actualizada: DNS, SNMP, juegos en red, etc.
  - Utilizado en multicast
- TCP (Transmission Control Protocol)
  - Protocolo de transporte bidireccional «orientado a conexión»
  - Fiable (checksum + retransmisión)
  - Usado cuando se requiere fiabilidad: SSH, HTTP, SMTP, etc.
- RTP (Real Time Protocol)
  - Orientado a aplicaciones de tiempo real: *streaming*, etc.
  - Funciona sobre UDP
- Otros

2 User Datagram Protocol (UDP)

- Unidad de transferencia: datagrama



Checksum: opcional sobre IPv4 y obligatorio sobre IPv6
- pseudo cabecera + cabecera UDP + datos
- pseudo cabecera = IP origen + IP destino + IP protocol (17) + Longitud del segmento UDP

Longitud: tamaño en bytes del datagrama (cabecera + datos)

Puertos origen y destino: números de 16 bits $[0, 65535]$ que asocian comunicaciones de capa de transporte (**sockets**) a procesos en equipos origen y destino

2.1 Puertos software

- Abstracción que identifica extremos de comunicación (asociados a procesos) en un equipo
- Se usan en UDP y TCP
  $\leq$1023: sistema (requieren privilegios de administrador)
  1024–49151: usuario
  $\geq$49152: dinámicos/privados (para clientes **sin** usar *bind*)
- IANA gestiona correspondencia servicio-puerto, e.g. SSH: 22, DNS: 53, HTTP: 80 (`/etc/services`)
- El servidor debe solicitar un puerto específico (*bind*)
- El cliente debe conocer el puerto del servidor y no usar *bind*

3 Transmission Control Protocol (TCP)

- Unidad de transferencia: segmento TCP
- Fases en una trasmisión mediante TCP:
  1. **Establecimiento** de la conexión
  2. Transferencia de datos bidireccional **fiable**
  3. **Cierre** de cada sentido de la conexión
- **Control de errores**: checksum [Tema 3]
- **Control de flujo**: emisor evita desbordar al receptor [p. 13]
- **Control de congestión**: evita sobrecargar la red [Tema 6]
- **Secuenciación de datos extremo-a-extremo** [Tema 3]
  - Segmentos con número de secuencia
  - Confirmación de datos recibidos correctamente
  - Retransmisión de segmento si no se recibe confirmación

3.1 Segmento TCP

Puerto origen y destino: identifican los extremos de la conexión

Nº de secuencia: indica el orden del segmento con respecto al mensaje original en bytes (inicialmente aleatorio)

Sig nº sec. esperado (ACK/Next): indica el nº de secuencia del siguiente byte que se espera recibir

Longitud de la cabecera: medida en palabras de 32 bits

3.1 Segmento TCP (II)

Flag ACK: indica si el campo sig. nº seq. esp. (ACK) es válido

Flags SYN, FIN, RESET: establecer/finalizar/abortar conexión

Flag URG: avisa de datos urgentes (puntero urgente)

Flag PUSH: fuerza el vaciado de *buffers* en origen y destino

Flags NS, CWR, ECE: control de *Explicit Congestion Notification* [Tema 6]

Checksum: pseudocabecera (=UDP) + cabecera TCP + datos

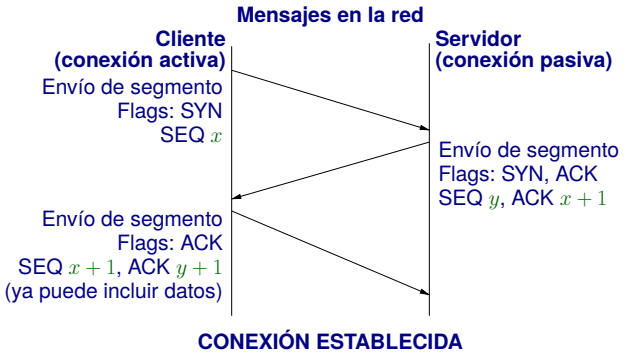
Ventana anunciada: especifica dinámicamente el tamaño de la ventana de recepción [Tema 3]

Opciones: funcionalidades nuevas o poco usadas

Datos: $\leq Maximum\_Segment\_Size = MTU - cabeceras\_IP\_TCP$

3.2 Establecimiento de conexión

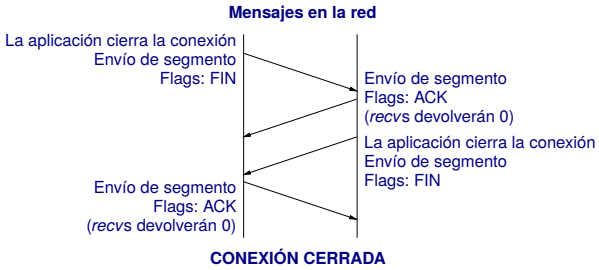
➤ *3-way handshake*



3.3 Finalización de conexión

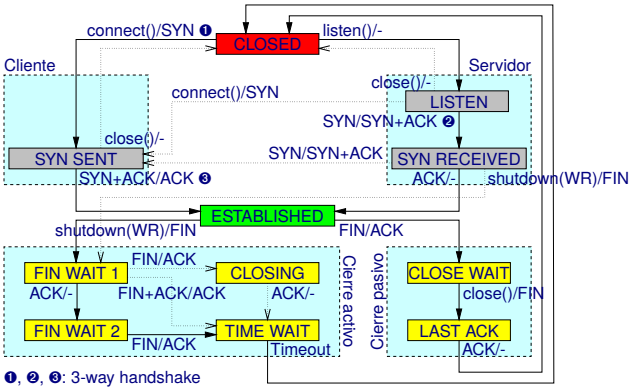
Dependiendo de cuándo se realice `int close(int fd)` o `int shutdown(int sockfd, int how)`, puede conllevar:

➤ Desconexión ordenada o consensuada (flag FIN)



➤ Desconexión desordenada o unilateral (flag RESET)

3.4 Estados de una conexión TCP



3.5 Control de flujo

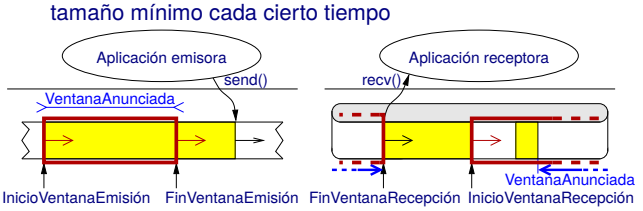
Objetivo: evitar que el emisor desborde al receptor

➤ Ventana deslizante de tamaño variable

➤ Receptor informa del espacio libre de su buffer en el campo **VentanaAnunciada** ($= BufferSize - Ocupado$)

➤ Emisor toma **VentanaAnunciada** como tamaño de ventana de emisión (límite de bytes enviados sin confirmar)

➤ Si recibe **VentanaAnunciada** $= 0$, reintenta envíos de tamaño mínimo cada cierto tiempo



3.6 Eficiencia

Rendimiento: enviar información cuanto antes

✗ Envío de segmentos pequeños

✗ Envío de ACKs sin datos

Eficiencia: minimizar el tráfico

➤ ACK retardado (máx 500 ms):

✓ Permite confirmar varios segmentos con un único ACK

✓ Permite aprovechar envío de datos para enviar ACK

➤ Retrasar envío de datos (algoritmo Nagle):

➤ Ante ACK pendiente, no enviar segmentos pequeños; encolarlos para enviar segmento de tamaño máximo

✓ Evita enviar segmentos pequeños

✓ Permite aprovechar envío de datos para enviar ACK

➤ Configurable desde SO y `setsockopt()`

3.6.1 Rango de números de secuencia

➤ Campo **número de secuencia** de 32 bits en TCP

➤ Se reusan cada $2^{32} = 4$ GiB transmitidos

➤ *Maximum Segment Lifetime* (MSL) de 2 minutos

➤ ¿En cuanto tiempo se reusará un número de secuencia?

➤ 100 Mb/s: $\frac{2^{32} \times 8\ b}{10^8\ b/s} = 5.7$ minutos $>$ MSL $\rightarrow$ Suficiente

➤ 10 Gb/s: $\frac{2^{32} \times 8\ b}{10^{10}\ b/s} = 3.4$ segundos $\ll$ MSL $\rightarrow$ ¡Insuficiente!

➤ Solución mediante opciones TCP:

➤ **Insertar marca de tiempo** de 32 bits en cada segmento

➤ **Extender espacio de números de secuencia** (*Protection Against Wrapping Sequence*, PAWS)

➤ Especifica orden de segmentos combinando campos ⟨marca de tiempo, numero de secuencia⟩

3.6.2 Rango de ventana anunciada

➤ Campo **ventana anunciada** de 16 bits en TCP

➤ Limita **ventana de emisión** ($w \cdot L$) a $2^{16} < 64$ kiB

➤ Velocidades/latencias altas necesitan ventanas grandes: $U < 100\,\% \rightarrow w \cdot T_t < RTT \equiv w \cdot L < RTT \cdot V_t$ [Tema 3]

➤ E.g. para $V_t = 100$ Mb/s y RTT de 96 ms:

➤ $0.096 \times 10^8 = 1.2$ MiB > 64 kiB $\rightarrow$ ¡Insuficiente!

➤ Solución mediante opción TCP:

➤ *Factor* de **escalado de ventana anunciada** (14 bits)

➤ $valorVentanaAnunciada = campoVentanaAnunciada \times Factor$

➤ Máximo: $(2^{16} - 1) \cdot (2^{14} - 1) = 1023$ MiB $\gg 64$ kiB

3.7 Retransmisión adaptativa

- El RTT puede variar mucho
- El tiempo de expiración (*timeout*) debe ajustarse al RTT:
  - Reenviar demasiado tarde infrautiliza la red
  - Reenviar demasiado pronto añade sobrecarga

Objetivo del emisor: para cada envío, establecer su tiempo de expiración en función del RTT estimado

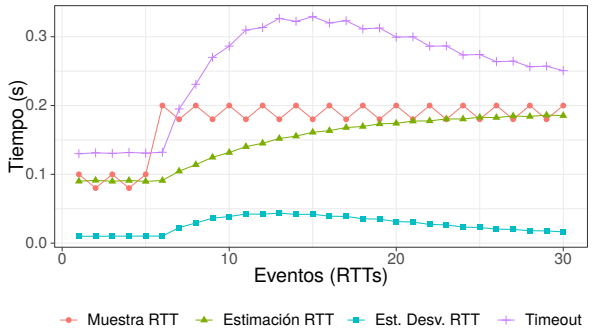
1. Muestrear RTTs: al llegar ACK mide el tiempo que ha tardado desde su correspondiente envío ($MuestraRTT$)
2. Estimar RTT previsto ($EstimRTT$) para cada envío a partir de muestras anteriores
3. Establecer el tiempo de expiración de cada envío en base al RTT estimado

3.7 Retransmisión adaptativa (II)

- Cálculo sencillo del tiempo de expiración ($\delta \approx 0.15$):
  $DifRTT = MuestraRTT - EstimRTT$
  $EstimRTT += \delta \cdot DifRTT$
  Timeout $= 2 \times EstimRTT$
- Cálculo mejorado (Jacobson/Karels) añadiendo desviación al anterior ($\delta \in [0,1]$, $\mu = 1$, $\phi = 4$):
  $DesvRTT = |DifRTT| - EstimDesvRTT$
  $EstimDesvRTT += \delta \cdot DesvRTT$
  Timeout $= \mu \cdot EstimRTT + \phi \cdot EstimDesvRTT$
- No se toma $MuestraRTT$ de mensaje reenviado

3.7 Retransmisión adaptativa (III)

Ejemplo: paso de RTTs de 80-100 ms a RTTs de 180-200 ms ignorando los efectos de segmentos perdidos



Redes de Computadores

Tema 6 – QoS y Control de Congestión

Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción

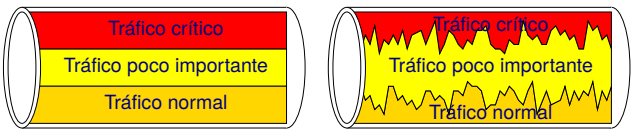
- La red tiene recursos limitados: ancho de banda de enlaces, buffers en dispositivos de interconexión
- Si la capacidad de la red no es suficiente, se pierden paquetes y se dice que está **congestionada**

Objetivos:

1. Repartir los recursos para que todos perciban la mayor **calidad de servicio** posible
2. Reducir la pérdida de paquetes mediante mecanismos de **control de congestión**

2 Calidad de Servicio (QoS)

- **Calidad de servicio (QoS)**: rendimiento promedio percibido por los usuarios, o más específicamente en redes, los mecanismos para proporcionar dicho rendimiento
  - Clasificar paquetes y tratarlos en contexto



Con calidad de servicio — Sin calidad de servicio

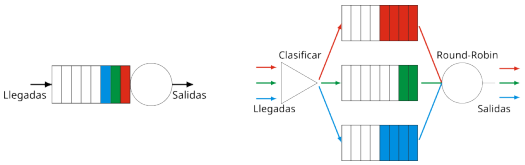
2 Calidad de Servicio (QoS) (II)

La calidad percibida depende de nuestro tráfico **y del tráfico del resto del mundo**. Opciones:

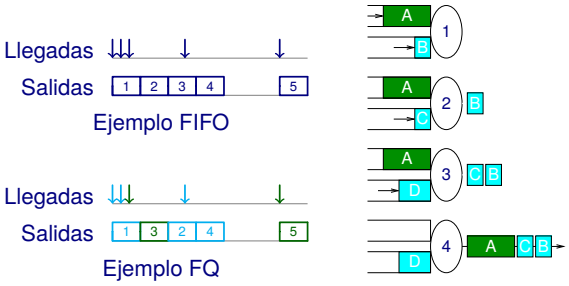
- No dar ningún tipo de calidad de servicio
  - Los paquetes son encolados y procesados por estricto orden de llegada al encaminador (FIFO, *First-In-First-Out*)
  - ✓ Sencillo y eficaz con poco tráfico
  - ✗ Calidad dependiente del resto del tráfico
- Ofrecer QoS «perfecta»
  - Negociar entre red y aplicación el tráfico que se va a enviar
  - Reservar recursos de red para tratar ese **flujo** de tráfico
  - Rechazar nuevos flujos si no hay recursos suficientes
  - ✓ Calidad independiente del resto del tráfico
  - ✗ Complejo y lento
- Cualquier opción intermedia entre las dos anteriores

2.1 Disciplina de colas

- **FIFO**: una única cola en el encaminador
  - No proporciona ningún tipo de calidad de servicio
- **Colas equitativas** (*Fair Queuing, FQ*)
  - Varias colas FIFO en el encaminador
  - Las colas se atienden de forma equitativa (*Round-Robin*)
  - Al llegar un paquete, se clasifica y se mete en la cola correspondiente
    - ¿Qué política se sigue para clasificar? [p. 8][p. 9]
  - Colas equitativas **ponderadas** (WFQ): una cola se atiende con k veces más frecuencia que otra



2.1 Disciplina de colas (II)



Ej. FQ con tamaños distintos

- Aplicación FQ/WFQ:
  - Se calcula el tiempo en el que finalizaría la transmisión de cada paquete como si solo existiera su cola
  - Transmisiones ordenadas por tiempo de finalización

2.2 Servicios diferenciados (*DiffServ*)

- Clasificación de paquetes (y su correspondiente asignación a colas de encaminadores) por el **tipo de tráfico** que contengan, es decir, **diferenciar servicios**
- 6 bits de mayor peso de campos «ToS» (IPv4) y «TrafficClass» (IPv6):
  - Paquetes de control de red
  - Paquetes críticos
  - Paquetes prioritarios
  - Paquetes «best effort» (sin garantías)
  - etc.
- El tipo puede ser establecido por el origen, por el encaminador de entrada a un SA, etc.
- La gestión por parte del encaminador es sencilla
- No permite dar un servicio particularizado

2.3 Servicios integrados (*IntServ*)

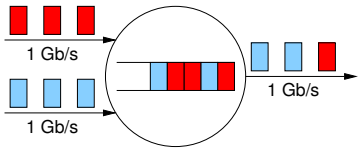
- Clasificación de paquetes (y su correspondiente asignación a colas de encaminadores) por **flujo** al que pertenezcan
  - E.g. paquetes de transmisión de vídeo entre x y y
  - IPv4 debe analizar IPs y puertos
  - IPv6 incorpora «FlowLabel» para facilitar la clasificación
- Combinar colas WFQ con circuitos virtuales o protocolos de **reserva de recursos**
  - La aplicación especifica a la red sus requisitos de tráfico y la red o los garantiza o deniega el servicio
  - Gestión compleja en el encaminador: necesita mantener mucha información (una cola/circuito/reserva por flujo)

2.3.1 Protocolo RSVP

- Resource ReSerVation Protocol (RSVP)
- Diseñado para funcionar sobre IP
- Utiliza refresco periódico como alternativa a los circuitos virtuales
- Permite cambios dinámicos de configuración
- Diseñado para ofrecer multicast
  - Fusiona requerimientos de receptores
  - Puede especificar varios interlocutores
- Basado en mensajes periódicos PATH/RESV
  - PATH (Origen → Destino): recoge datos sobre recursos de encaminadores
  - RESV (Destino → Origen): reserva recursos

3 Control de congestión

- **Congestión**: pérdida frecuente de paquetes
  - Cuando un dispositivo recibe paquetes a más velocidad de la que los puede reexpedir se va llenando su cola
  - Si llegan paquetes estando la cola llena, se descartan
  - Mecanismos de **control de congestión** para tratarla



3 Control de congestión (II)

- La tasa a la que llega el tráfico depende de todas las capas
  - Tiempos de espera aleatorios en redes locales
  - Tiempo en colas de encaminadores en capa de red
  - Velocidad efectiva de tráfico dependiente de ventana, retransmisiones, etc. en capa de transporte
- ¿Cómo actúa cada capa para tratar la congestión?
  - Independientemente → peligro de entorpecerse
  - Coordinadas → se pierde la abstracción al asumir que por encima/debajo se actúa de cierta forma
- Mecanismos de control de congestión desde…
  - TCP: Tahoe (**comienzo lento, prevención de congestión, retransmisión rápida**), Reno (**recuperación rápida**), **SACK**, **Vegas**, BIC/CUBIC, Compound, etc.
  - Capa de red: **ECN**, **RED**, etc.

3.1 Control de congestión en TCP

- TCP gestiona la **ventana de emisión**: enviar demasiado…
  - …puede saturar al receptor (**control de flujo**)
    - Campo **v. anunciada** ($vanun$) en cabecera TCP [Tema 5]
  - …puede congestionar la red (**control de congestión**)
    - Nueva variable v. congestión ($vcong$) en emisor
  - Ventana de emisión $= \min(vanun, vcong)$
- En transmisión por cable hay pocos errores [Tema 3]
- El emisor TCP no conoce la capacidad de la red
  - Asumir que **paquetes perdidos** → **congestión**
  - Si no se pierden paquetes, aumentar un poco $vcong$
  - Si se pierden paquetes, reducir mucho $vcong$
  - Incremento aditivo, decremento multiplicativo

3.1.1 Algoritmos básicos

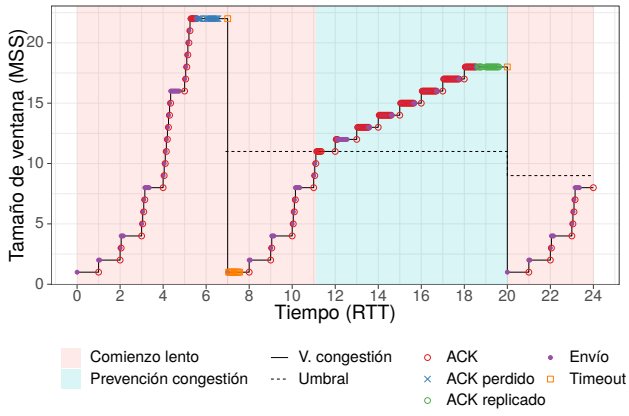
- Inicialmente $umbral \leftarrow \infty$ // umbral de comienzo lento
- Algoritmo de **comienzo lento**:

```
vcong ← 10 segmentos          // ventana de congestión inicial
para cada ACK recibido:       // hay vcong ACKs en cada RTT
    vcong ← vcong + 1 // incr. exponencial; vcong × 2 cada RTT
    si vcong = umbral:
        iniciar prevención de congestión
    si vence timeout:
        umbral ← vcong/2
        pasar a comienzo lento
```

- Algoritmo de **prevención de congestión**:

```
para cada ACK recibido:       // hay vcong ACKs en cada RTT
    vcong ← vcong + 1/⌊vcong⌋  // incr. lineal; vcong + 1 cada RTT
    si vence timeout:
        umbral ← vcong/2
        pasar a comienzo lento
```

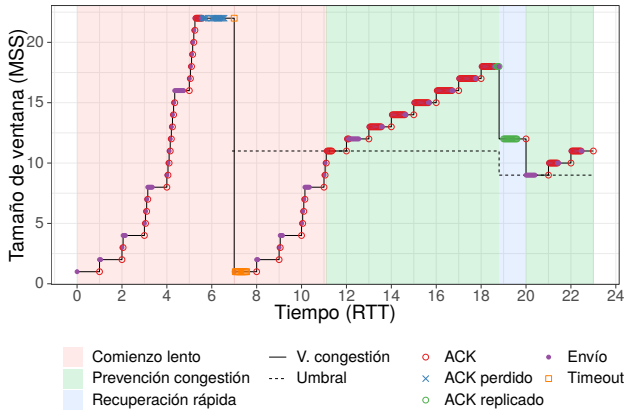

3.1.1 Algoritmos básicos (II)



3.1.2 Mejoras

- **Retransmisión rápida** (en TCP Tahoe)
  - Reenviar segmento tras 3 ACK repetidos
  - Evita esperar a que venza el timeout
- **Recuperación rápida** (en TCP Reno)
  - Si el ACK del segmento retransmitido se recibe antes de su timeout, iniciar prevención de congestión con nuevo umbral
  - Evita volver a comienzo lento
- **ACK selectivo** (SACK, en opciones TCP)
  - Notifica bloques no contiguos recibidos correctamente
    - ACK 5100 vs. SACK ⟨5000–5100, 5200–5300, 5400–5500⟩
  - Un segmento perdido ya no oculta pérdidas posteriores
- Otras…

3.1.2 Mejoras (II)



3.2 Prevención desde origen (TCP Vegas)

- Crecimiento de colas en encaminadores → proximidad de congestión
- Prevención basada en retardos, no en pérdidas
- Estación origen observa indicios de crecimiento de colas:
  - Si crece **ventana de congestión** (y tasa de envíos)…
  - y se mantiene **tasa de ACKs** (y de recepciones)…
  - entonces algunos paquetes están **quedando en colas**:
    - Reducir ventana de congestión
  - Si no, aumentar ventana de congestión

3.3 Detección Explícita de Congestión

- Detección en capa de red; control en capa de transporte
- Encaminador monitoriza la ocupación media de cola
  - Marca **bit de congestión** (en la cabecera del paquete) si ocupación media de cola $>$ umbral
- Destino reenvía a origen el valor del bit
- Origen ajusta su tasa de envío en base a los bits marcados
  - $< 50\,\%$ de última ventana → incr. ventana
  - $\geq 50\,\%$ de última ventana → decr. ventana
- En TCP/IP: *Explicit Congestion Notification* (ECN)
  - 2 bits en ToS (IPv4) y TraficClass (IPv6) para notificar ECN activo y marcar congestión hacia destino
  - 3 bits en flags TCP para notificar congestión a origen y avisar a receptor sobre reducción de ventana de congestión

3.4 Detección temprana aleatoria (RED)

- *Random Early Detection* (RED)
- Detección y control de congestión en capa de red
- Notifica descartando paquetes (coopera con TCP)
- Encaminador monitoriza la ocupación media de la cola a la llegada de cada paquete
  - Si $> UmbralMax$, descarta paquete
  - Si $> UmbralMin$, descarta paquete con una probabilidad P
  - Con cada paquete no descartado, incrementa P
- La probabilidad de descarte en un flujo es proporcional a su cantidad de tráfico
- Posibilidad de «castigar» a quienes no regulan congestión

Créditos de material reutilizado

Imagen «Parallel network of queues» (p. 6): CC BY SA Gonzalo Medina (editadas)

Redes de Computadores

Tema 7 – Capas superiores

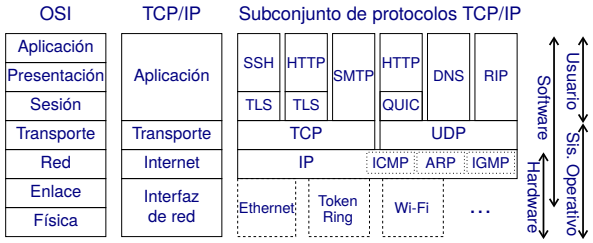
Juan Segarra y Jesús Alastruey

Dpt. de informática e ingeniería de sistemas
Universidad de Zaragoza

Índice

1 Introducción

- **Capa de sesión**: control/coordinación de comunicaciones
- **Capa de presentación**: codificación/interpretación de información
- **Capa de aplicación**: protocolos específicos para aplicaciones concretas



2 Capa de sesión OSI

Sesión: espacio de tiempo ocupado por una actividad

- Capa encargada del establecimiento, mantenimiento y terminación de comunicaciones dentro de una sesión
- Servicios: autentificación, permisos, restablecer sesión, etc.
- En TCP/IP la aplicación gestiona la sesión
  - HTML cookies mantienen estado de la sesión (e.g. carrito de compra) y lo recuperan en sesiones posteriores
  - X Window System coordina flujos de pantalla/teclado/ratón
  - …
- Especialmente importante en voz sobre IP (VoIP)
  - Session Description Protocol (SDP), Session Initiation Protocol (SIP), Control de llamada (H.323), etc.

3 Capa de presentación OSI

- Problema: el formato de la información depende de:
  - Hardware: *big/little-endian*, tamaño de registros (tamaño de enteros por defecto)…
  - Sistema operativo: repertorio y codificación de caracteres (ASCII 7/8 bits, ISO, UTF), salto de línea, protocolos disponibles…
  - Compilador: uso de relleno (*padding*) para alinear campos, elementos vectoriales…
  - Lenguaje de programación: tipos de datos (e.g. booleano) y funciones predefinidas (e.g. serialización)
  - Aplicación: datos a enviar
- Para intercambiar datos se necesita un formato común
  - E.g. formatos XDR, ASN.1, MIME, HTML, XML, JSON

3 Capa de presentación OSI (II)

- Codificación/compresión de datos
  - Minimiza recursos de almacenamiento/transmisión
    - Tasa de compresión muy buena en texto (e.g. XML)
  - Métodos optimizados para tipos y/o usos de datos
    - Vídeo: MPEG 2 (calidad), MPEG 4 (eficiencia)
  - E.g. RTP puede recodificar datos «en tránsito» para adaptarlos al receptor
- Seguridad extremo-a-extremo
  - Privacidad (cifrado de datos)
  - Integridad (garantizar la no alteración del mensaje)
  - E.g. protocolo TLS (*Transport Layer Security*)

3.1 Formato XML

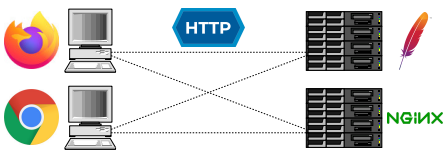
Extensible Markup Language:

- Toda la información va marcada semánticamente
- Entendible por el ser humano y la máquina
- Sólo usa texto (e.g. entero $83 \rightarrow$ caracteres 8 y 3)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>two of our famous Belgian Waffles with plenty of real
        maple syrup</description>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>light Belgian waffles covered with strawberries and
        whipped cream</description>
  </food>
</breakfast_menu>
```


4 Capa de aplicación OSI

- Protocolos específicos para cada servicio
- Abstrae a las aplicaciones de sus comunicaciones
  - Protocolo en capa de aplicación para servicios web: HTTP
  - Aplicaciones web: firefox, chrome, apache, etc.
- Modelo TCP/IP aglutina aplicación, presentación y sesión, e.g. HTTP + HTML + TLS + cookies



4.1 Protocolo HTTP

- HyperText Transport Protocol
- Dos tipos de mensaje: petición, respuesta
- Objetos web (páginas, imágenes, etc.) referenciables mediante URL (*Uniform Resource Locator*):
  - esquema://[[user[:pass]]@]dirección[:puerto]/ruta/objeto
  - http://www.unizar.es/academico/unizar.html
- Versiones
  - HTTP/1.0: por defecto **un objeto por conexión**
  - HTTP/1.1: por defecto **conexión persistente**
  - HTTP/2: **comprime cabeceras** y puede enviar objetos aún no solicitados o en **orden distinto** al solicitado

4.1 Protocolo HTTP (II)

HTTP/3: HTTP sobre **QUIC**

- *Quick UDP Internet Connections*
- Aglutina capas de…
  - Transporte: sobre UDP, con conexión, secuenciación y control de congestión
  - Sesión: sesión identificada por ID
  - Presentación: siempre cifrado
- Evita RTTs de establecimiento TCP+TLS
  - Servidor envía configuraciones, claves, etc. en 1er mensaje
  - Sesiones siguientes lo reutilizan (0 RTTs)
- Mejor secuenciación que TCP: diferencia transmisiones de retransmisiones (recordar muestreo RTTs [Tema 5])

✍ Capturar tráfico hacia Google con wireshark.

4.1.1 Petición HTTP

1. Línea de petición (**negrita**) + salto de línea (<CR><LF>)
2. Cabeceras de petición (*cur.*) + salto de línea (<CR><LF>)
3. Salto de línea (<CR><LF>)
4. Cuerpo (vacío si no se transmiten datos)

GET /index.html HTTP/1.1
Host: www.unizar.es
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive

4.1.1 Petición HTTP (II)

- Línea de petición: tipo objeto versión
  - E.g. `GET /index.html HTTP/1.1`
- Tipos de petición (*method*):

GET	Solicita un elemento (HTTP/1.0)
POST	Envía datos a un *script* (HTTP/1.0)
HEAD	Solicita sólo la cabecera (HTTP/1.0)
PUT	Reemplaza el elemento (HTTP/1.1)
DELETE	Borra el elemento indicado (HTTP/1.1)
otros:	TRACE, OPTIONS, CONNECT, PATCH

- GET y HEAD nunca modifican contenido
- Dependiendo del método, se pueden enviar datos en el URL (`miurl?var=25&var2=30`) o en el cuerpo de la petición

4.1.1 Petición HTTP (III)

Cabeceras más importantes para la petición:

Host	Nombre del equipo al que se solicitan datos
User-Agent	Identificador del software cliente
Accept	Tipos de datos aceptados
Accept-Language	Idiomas aceptados
Accept-Encoding	Métodos de compresión aceptados
Accept-Charset	Codificaciones de texto aceptadas
Connection	Comportamiento de conexión deseado
Keep-Alive	Tiempo a mantener la conexión abierta
Referrer	URL que nos ha llevado a esta petición
If-Modified-Since	Petición condicionada
Cookie	Identificador proporcionado por el servidor

4.1.2 Respuesta HTTP

1. Línea de respuesta (**negrita**) + salto de línea (<CR><LF>)
2. Cabeceras de respuesta (*cur.*) + salto de línea (<CR><LF>)
3. Salto de línea (<CR><LF>)
4. Cuerpo (texto normal)

HTTP/1.1 200 OK
Date: Tue, 04 Dec 2012 09:48:53 GMT
Server: Apache
Set-Cookie: JSESSIONID=45C0E390670416F258729C7464522923; Path=/
ETag: W/"13322-1354614320000"
Last-Modified: Tue, 04 Dec 2012 09:45:20 GMT
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1
Content-Language: es

```
1ff8

<!DOCTYPE HTML PUBLIC //W3C//DTD HTML 4.01 Transitional//EN"…
```


4.1.2 Respuesta HTTP (II)

- Línea de respuesta comprensible por máquinas y humanos: versión código texto (`HTTP/1.1 200 OK`)
  - 2xx Success
    - 200 OK
  - 3xx Redirection
    - 301 Moved permamently
    - 304 Not modified (ante petición condicional)
  - 4xx Client Errors
    - 400 Bad request
    - 401 Authorization required
    - 404 Not found
  - 5xx Server errors
    - 500 Internal error
    - 502 Service overloaded
    - 505 Version not supported

4.1.2 Respuesta HTTP (III)

Cabeceras más importantes para la respuesta:

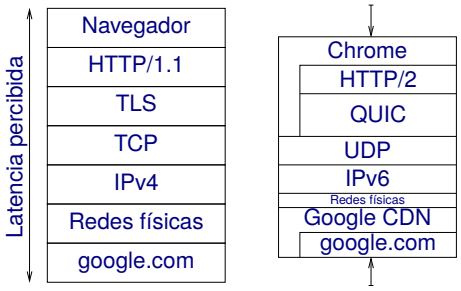
Server	Software del servidor
Date	Fecha actual
Last-Modified	Fecha de modificación
Expires	Fecha de expiración
Transfer-Encoding	Codificación de la transferencia
Content-Type	Tipo de datos enviados
Content-Length	Longitud de datos enviados
Content-Encoding	Método de compresión usado
ETag	Identificador de recurso
Set-cookie	Envía un identificador al usuario
Cache-Control	Especificaciones de cacheo de contenido

4.1.3 Distribución de contenidos

- Antes, servidores *proxy* (caches en el ISP)
  - Poco sentido con webs dinámicas, donde cada página está personalizada (*cookies*, navegador, idioma, etc.)
  - E.g. Proxy unizar deshabilitado en 2006
- Ahora, **redes de distribución de contenidos** (CDNs)
  - Red de servidores «cercanos» mantienen réplicas
  - Latencia: el servidor está cerca del cliente
  - Redundancia: si hay fallos, otro puede responder
  - *Geotargeting*: respuesta en contexto geográfico (e.g. idioma, búsquedas)
- Empresas especializadas en distribución de contenidos
  - E.g. *Google*, *Akamai Technologies*, *Amazon CloudFront*, etc.
  - Los servidores centrales distribuyen su web a las réplicas
  - Redirección a réplicas a través de DNS, etc.

4.1.4 Enfoque Google

- Mejorar tiempo de respuesta de servidores mediante CDNs
- Mejorar navegador y protocolos web
- Mejorar transporte-sesión-presentación mediante QUIC



4.2 Protocolo SMTP

- Simple Mail Transfer Protocol
- Servicio de *envío* de correo electrónico
- Mensages de texto de petición/respuesta
  - En principio sólo admite texto ASCII (7 bits)
  - Formato MIME admite codificaciones con más bits
  - Adjuntos codificados como ASCII (base64, uuencode)
- Formato del mensaje separado del protocolo
  - Formatos RFC2822, MIME

✍ Experimentar con SMTP desde lab000: `nc localhost 25`

4.2 Protocolo SMTP (II)

```
S: 220 www.example.com ESMTP Postfix
C: HELO mydomain.com
S: 250 Hello mydomain.com
C: MAIL FROM:<sender@mydomain.com>
S: 250 Ok
C: RCPT TO:<friend@example.com>
S: 250 Ok
C: DATA
S: 354 End data with <CR><LF>.<CR><LF>
C: Subject: test message
C: From: sender@mydomain.com
C: To: friend@example.com
C:
C: Hello,
C: This is a test.
C: Goodbye.
C: .
S: 250 Ok: queued as 12345
C: QUIT
S: 221 Bye
```


4.3 Protocolo DNS

Domain Name System / Sistema de Nombres de Dominios

Dominio de Internet: nombre jerárquico asociado a un grupo de dispositivos o equipos conectados a Internet

- DNS proporciona servicios de nombres de dominio
  - Traducción entre nombre y dirección IP:

```
$ host hendrix01.cps.unizar.es
hendrix01.cps.unizar.es has address 155.210.152.183
$ dig -x 155.210.152.183 +short
hendrix01.cps.unizar.es.
```

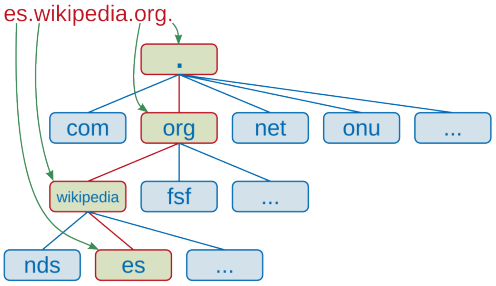
  - Servidor de correo asociado al dominio:

```
$ host -t mx unizar.es
unizar.es mail is handled by 10 mx01.puc.rediris.es.
unizar.es mail is handled by 10 mx02.puc.rediris.es.
```

- Servicio importante: mínimo 2 servidores por dominio

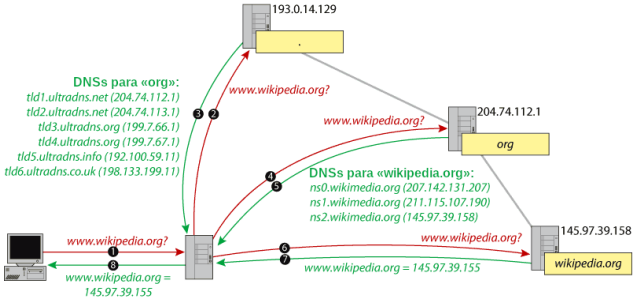
4.3 Protocolo DNS (II)

- Organización jerárquica en dominios
- Nivel superior (*top level domain*, TLD) gestionado por NIC
- *Full Qualified Domain Name* (FQDN) o nombre absoluto: toda la jerarquía separada por puntos y acabada en punto



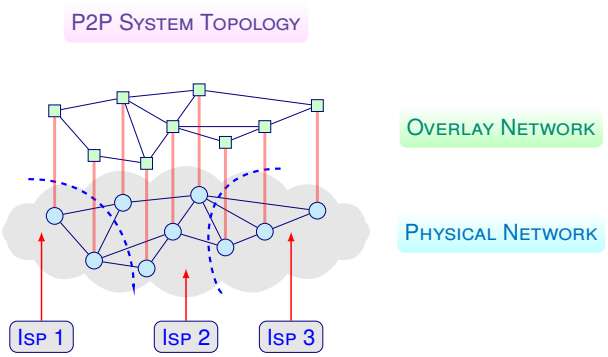
4.3 Protocolo DNS (III)

- En general, estrategia de resolución iterativa por parte del DNS local, e.g. resolución de «www.wikipedia.org»:



5 Redes superpuestas

- Red virtual construida sobre una red física



5 Redes superpuestas (II)

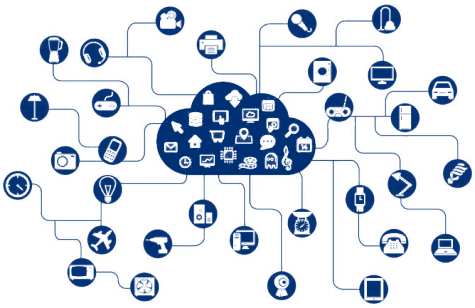
- Cada enlace virtual punto-a-punto es un túnel o un enlace extremo-a-extremo
- Cada nodo implementa un encaminador virtual usando esos enlaces virtuales
- Las aplicaciones pueden usar los **nuevos servicios de red** proporcionados
  - Versiones experimentales de IP, e.g. IPv6
  - Redes superpuestas anónimas, e.g. Tor, I2P
  - Redes de distribución de contenidos [p. 17]
- Capas superpuestas y enlaces virtuales generan sobrecarga

5.1 Redes P2P

- Redes igual-a-igual (*Peer-to-Peer, P2P*)
- Aplicación implementa encaminador y enlaces virtuales
- Aplicación integra cliente (e.g. buscador de ficheros) y servidor (e.g. servidor de mis ficheros a otros clientes)
- Descentralización y autoorganización → muy escalable
- Redes superpuestas no estructuradas
  - Basadas en búsquedas mediante inundación, e.g. Gnutella
- Redes superpuestas estructuradas
  - Tablas *hash* distribuidas identifican objetos y nodos, e.g. eMule, BitTorrent
  - (*magnet-link*) Hash de un objeto compartido en una red superpuesta BitTorrent

6 Universal Plug and Play (UPnP)

- Conjunto de protocolos para que los dispositivos en una red descubran y puedan usar otros dispositivos presentes en la red de forma fácil (Alexa, google home, etc.)



6 Universal Plug and Play (UPnP) (II)

1. Asignar dirección de red
   a)
   b)
2. Descubrir (*Simple Service Discovery Protocol*, SSDP)
   - Mensajes HTTP (tipo de servicio y URL con detalles)
   - ¿Unicast o multicast?
   - ¿TCP o UDP?
3. Describir comandos proporcionados (URL anterior)
   - ¿Formato de descripción?
4. Interactuar con otros
   - ¿Formato comandos?
   - ¿Formato eventos?

Créditos de material reutilizado

Imagen «Firefox logo, 2019» (p. 8): Mozilla Public License Version 2, Mozilla Corporation
Imagen «Google Chrome icon (September 2014)» (p. 8): PD Google
Imagen «Apache Feather Logo» (p. 8): Apache License, Version 2.0, The Apache Software Foundation
Imagen «Nginx logo» (p. 8): PD Nginx
Imagen «HTTP logo» (p. 8): PD IETF HTTP Working Group (HTTPbis)
Imagen «DNS schema» (p. 22): CC BY SA TilmannR
Imagen «Dns-wikipedia» (p. 23): CC BY SA Pavel.satrapa (traducida)
Imagen «Example: P2P system topology» (p. 24): CC BY Claudio Fiandrino
Imagen «Horse shoe Magnet» (p. 26): PD Sav vas, Francesco Rollandin
Imagen «IoT» (p. 27): CC BY SA Electronics-lab.com (colores modificados)