

uc3m

Universidad  
**Carlos III**  
de Madrid

# Introducción a la librería de desarrollo librawnet

Redes y Servicios de Comunicaciones Avanzadas

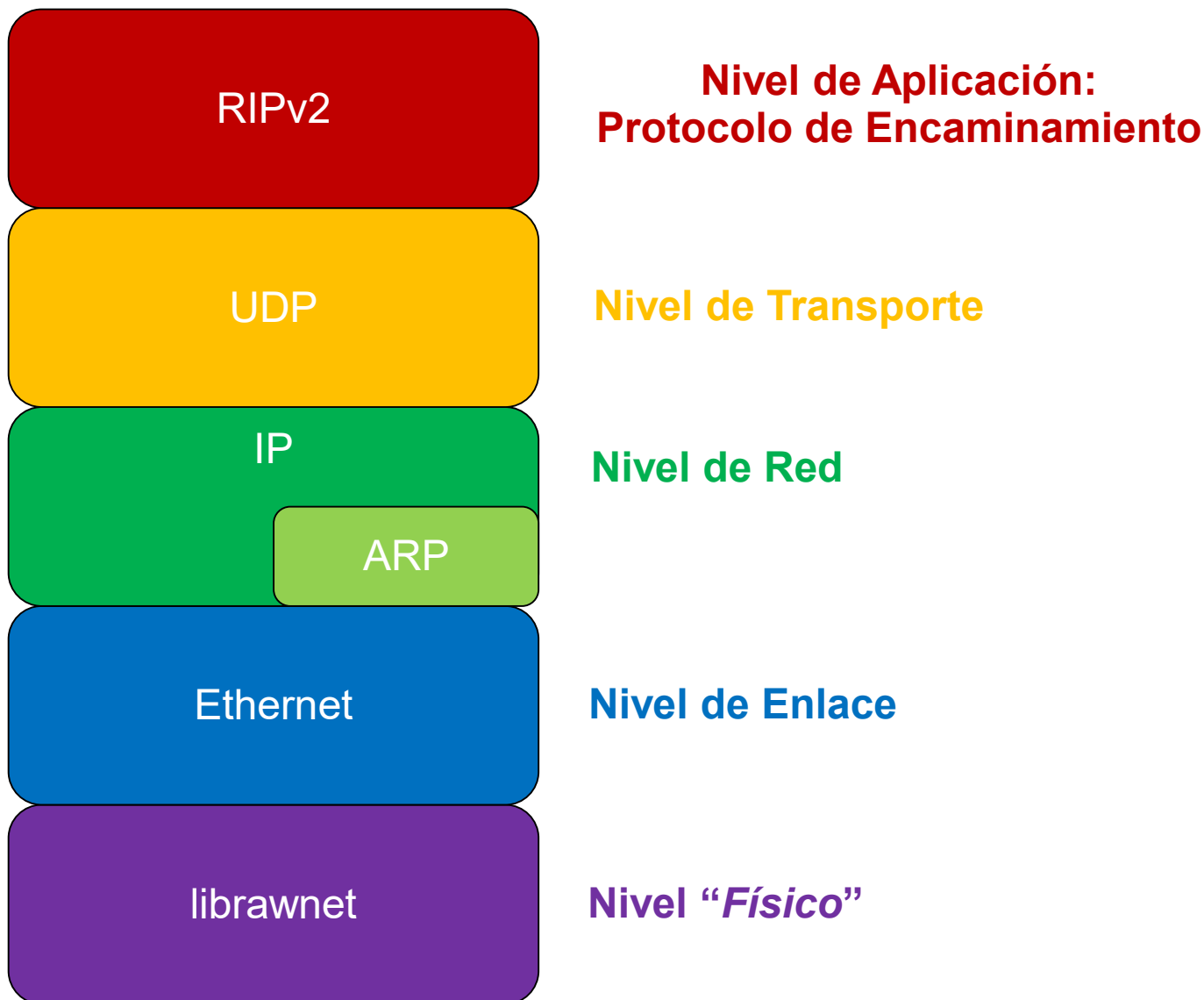
Departamento de Ingeniería Telemática

Manuel Urueña

[muruenya@it.uc3m.es](mailto:muruenya@it.uc3m.es)

Carlos J. Bernardos

[cjbc@it.uc3m.es](mailto:cjbc@it.uc3m.es)



- ❑ **Librería para la programación de protocolos de red “*en crudo*”:**
  - ❖ Permite acceder directamente a los interfaces de red del equipo.
- ❑ **Los paquetes enviados/recibidos no han sido procesados por la pila de protocolos del equipo:**
  - ❖ Es necesario añadir/procesar las **cabeceras** de todos los protocolos de la pila que se desea desarrollar.
  - ❖ Realmente el *kernel* y **librawnet** reciben una **copia** de todos los paquetes recibidos por la tarjeta de red.
- ❑ **librawnet proporciona dos módulos:**
  1. Temporizadores con precisión de milisegundos: **timerms.h** (**man timerms**)
  2. Gestión de los interfaces de red, envío/recepción de paquetes “crudos”: **rawnet.h** (**man rawnet**)

## ❑ Obtener tiempo actual:

```
long long int timerms_time();
```

## ❑ (Re)iniciar temporizador:

```
long long int timerms_reset ( timerms_t * timer,  
                              long int timeout );  
  
/* 'timeout' >= 0: 'timer' expira en 'timeout' ms  
   'timeout' < 0: Temporizador "infinito"          */
```

## ❑ Consultar estado temporizador:

```
long int timerms_elapsed ( timerms_t * timer );  
long int timerms_left ( timerms_t * timer );
```

## ❑ Temporizador que expira a los 500 ms:

```
timerms_t timer;
long int timeout = 500;
long long int now = timerms_reset(&timer, timeout);
long elapsed_time = timerms_elapsed(&timer); /* elapsed_time == 0 */
long time_left = timerms_left(&timer);      /* time_left == 500 */
...[+100 ms]
elapsed_time = timerms_elapsed(&timer);    /* elapsed_time == 100 */
time_left = timerms_left(&timer);          /* time_left == 400 */
...[+400 ms]
elapsed_time = timerms_elapsed(&timer);    /* elapsed_time == 500 */
time_left = timerms_left(&timer);          /* time_left == 0 */
...[+500 ms]
elapsed_time = timerms_elapsed(&timer);    /* elapsed_time == 1000 */
time_left = timerms_left(&timer);          /* time_left == 0 */
```

## ❑ Temporizador “*infinito*”:

```
timerms_t timer;  
long int timeout = -1;  
long long int now = timerms_reset(&timer, timeout);  
long elapsed_time = timerms_elapsed(&timer); /* elapsed_time == 0 */  
long int time_left = timerms_left(&timer); /* time_left < -1 */  
...[+1000 ms]  
elapsed_time = timerms_elapsed(&timer); /* elapsed_time == 1000 */  
time_left = timerms_left(&timer); /* time_left < -1 */
```

## ❑ Operaciones con interfaces “*crudos*”:

```
rawiface_t * rawiface_open    ( char* ifname );
char* rawiface_getname      ( rawiface_t * iface );
int rawiface_getaddr       ( rawiface_t * iface, uchar addr[] );
int rawiface_getmtu       ( rawiface_t * iface );
int rawiface_close        ( rawiface_t * iface );
```

## ❑ Enviar/Recibir paquetes “*crudos*”:

```
int rawnet_send ( rawiface_t * iface, uchar * pkt, int pkt_len );
int rawnet_recv ( rawiface_t * iface, uchar buffer[], int buf_len,
                long int timeout );
int rawnet_poll ( rawiface_t * ifaces[], int ifnum,
                long int timeout );
```

## ❑ Gestión de errores (no *multi-thread*):

```
char* rawnet_strerror();
```

# Ejemplo: Capa Ethernet

`eth_client.c`

Cliente Ethernet

Ethernet

librawnet

`eth_server.c`

Servidor Ethernet

Ethernet

librawnet

`eth.h`  
`eth.c`

`rawnet.h`  
`timerms.h`  
`librawnet.so`





## ❑ Direcciones MAC:

```
#define MAC_ADDR_SIZE 6      /* 6 bytes == 48 bits */
typedef unsigned char mac_addr_t [MAC_ADDR_SIZE];
mac_addr_t MAC_BCAST_ADDR = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
#define MAC_STR_LENGTH 18   /* strlen("12:45:78:01:34:67" + '\0') */
void mac_addr_str ( mac_addr_t addr, char str[] );
int mac_str_addr ( char* str, mac_addr_t addr );
```

## ❑ Operaciones con interfaces Ethernet:

```
eth_iface_t * eth_open      ( char* ifname );
char * eth_getname ( eth_iface_t * iface );
void eth_getaddr ( eth_iface_t * iface, mac_addr_t addr );
int eth_close ( eth_iface_t * iface );
```

## ❑ Enviar/Recibir tramas Ethernet:

```
int eth_send ( eth_iface_t * iface, mac_addr_t dst, uint16_t type,
              unsigned char * payload, int payload_len );
int eth_recv ( eth_iface_t * iface, mac_addr_t src, uint16_t type,
              unsigned char buffer[], int buf_len, long int timeout );
int eth_poll ( eth_iface_t * ifaces[], int ifnum, long int timeout );
```

```
#include "eth.h"
#include <rawnet.h>
#include <timerms.h>

typedef struct eth_iface {
    rawiface_t * raw_iface; /* Manejador del interfaz "crudo" */
    mac_addr_t mac_address; /* Dirección MAC del interfaz. */
} eth_iface_t;

eth_iface_t * eth_open ( char* ifname ) {
    struct eth_iface * eth_iface;
    eth_iface = malloc(sizeof(struct eth_iface));

    rawiface_t * raw_iface = rawiface_open(ifname);
    if (raw_iface == NULL) { return NULL; }
    eth_iface->raw_iface = raw_iface;

    rawiface_getaddr(raw_iface, eth_iface->mac_address);
    return eth_iface;
}
```

# eth\_send() (I)

```
struct eth_frame {
    mac_addr_t dest_addr;          /* Dirección MAC destino */
    mac_addr_t src_addr;          /* Dirección MAC origen */
    uint16_t type;                /* Campo 'Tipo' */
    unsigned char payload[ETH_MTU]; /* Campo 'Payload': 1500 bytes */
};

#define ETH_HEADER_SIZE 14        /* 6 + 6 + 2 (sin FCS)
    */

int eth_send ( eth_iface_t * iface, mac_addr_t dst, uint16_t type,
              unsigned char * payload, int payload_len ) {
    /* Crear la Trama Ethernet y rellenar todos los campos */
    struct eth_frame eth_frame;
    memcpy(eth_frame.dest_addr, dst, MAC_ADDR_SIZE);
    memcpy(eth_frame.src_addr, iface->mac_address, MAC_ADDR_SIZE);
    eth_frame.type = htons(type);
    memcpy(eth_frame.payload, payload, payload_len);
    int eth_frame_len = ETH_HEADER_SIZE + payload_len;
    /* ... */
}
```

```
                                /* ... */
/* Imprimir trama Ethernet */
char* iface_name = eth_getname(iface);
mac_addr_str(dst, mac_str);
printf("eth_send(type=0x%04x, payload[%d]) > %s/%s\n",
       type, payload_len, iface_name, mac_str);
print_pkt((unsigned char *) &eth_frame, eth_frame_len, ETH_HEADER_SIZE);

/* Enviar la trama Ethernet creada con rawnet_send() */
bytes_sent = rawnet_send(iface->raw_iface, (unsigned char *)
                        &eth_frame, eth_frame_len);
if (bytes_sent == -1) {
    fprintf(stderr, "eth_send(): ERROR en rawnet_send(): %s\n",
           rawnet_strerror());
    return -1;
}
return (bytes_sent - ETH_HEADER_SIZE);
}
```

## eth\_recv() (I)

```
int eth_recv ( eth_iface_t * iface, mac_addr_t src, uint16_t type,
              unsigned char buffer[], int buf_len, long int timeout ) {
    timerms_reset(&timer, timeout);
do {
    /* Recibir trama del interfaz Ethernet */
    long int time_left = timerms_left(&timer);
    unsigned char eth_buffer[ETH_FRAME_MAX_LENGTH]; /* 14 + 1500 */
    int frame_len = rawnet_recv (iface->raw_iface, eth_buffer,
                                ETH_FRAME_MAX_LENGTH, time_left);
    if ((frame_len == -1) || (frame_len == 0)) { return frame_len; }
    else if (frame_len < ETH_HEADER_SIZE) { continue; }
    /* Comprobar si es la trama que nos han pedido */
    struct eth_frame * eth_frame_ptr = (struct eth_frame *) eth_buffer;
    int is_my_mac = (memcmp(eth_frame_ptr->dest_addr, iface->mac_address,
                            MAC_ADDR_SIZE) == 0);

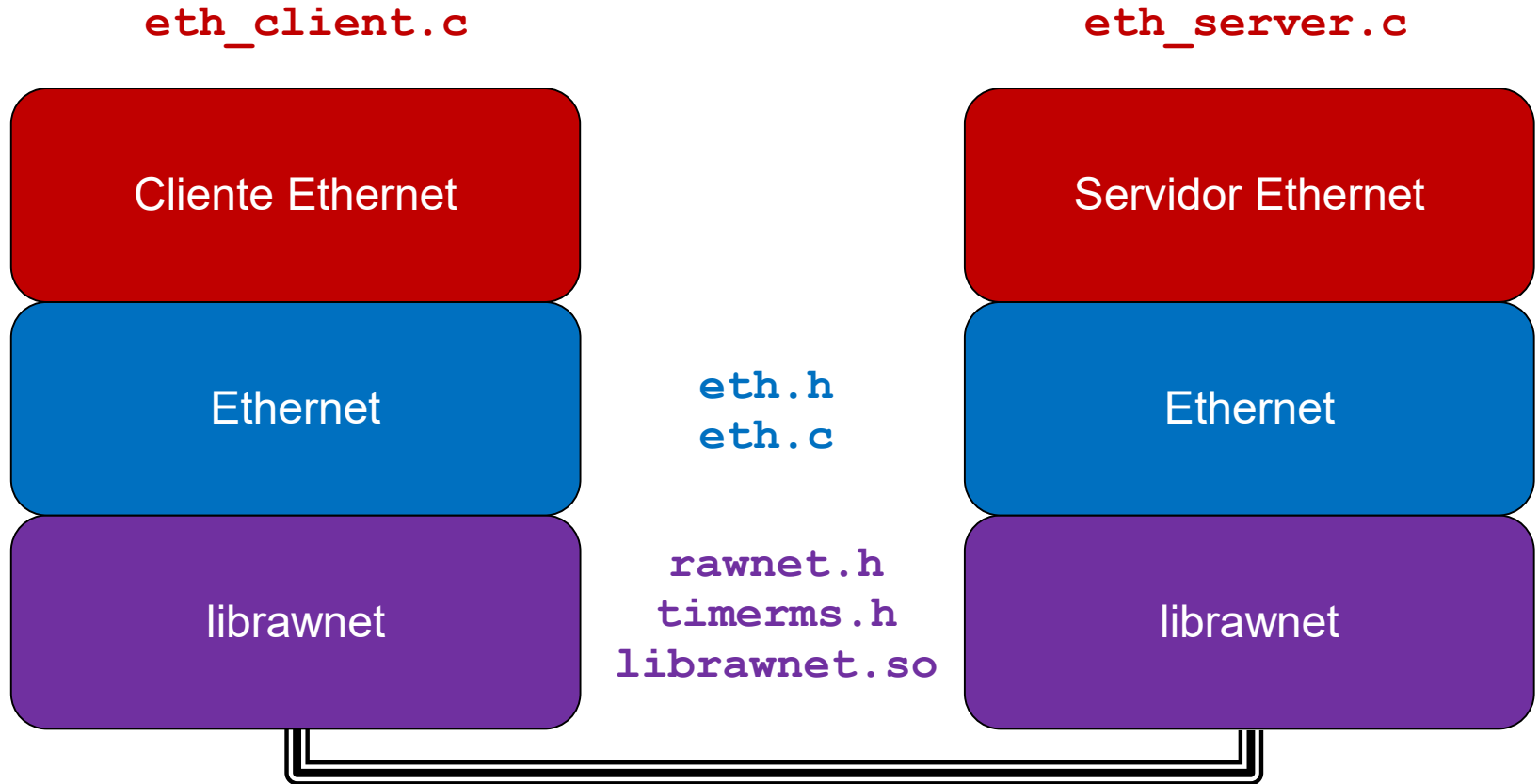
    int is_target_type = (ntohs(eth_frame_ptr->type) == type);
} while ( ! (is_my_mac && is_target_type) );
    /* ... */
}
```

```
        /* ... */

/* Trama recibida con el 'Type' solicitado:
   Copiar datos y dirección MAC origen */
memcpy(src, eth_frame_ptr->src_addr, MAC_ADDR_SIZE);
payload_len = frame_len - ETH_HEADER_SIZE;
if (buf_len > payload_len) {
    buf_len = payload_len;
}
memcpy(buffer, eth_frame_ptr->payload, buf_len);

return payload_len;
}
```

# Ejemplo: Cliente / Servidor Ethernet



```
int main ( int argc, char * argv[] ) {
    /* Procesar argumentos */
    if (argc != 5) {
        printf("Uso: %s <iface> <tipo> <mac> <long>\n", argv[0]); exit(-1);
    }
    /* Abrir la interfaz Ethernet */
    eth_iface_t * eth_iface = eth_open(iface_name);
    if (eth_iface == NULL) { exit(-1); }
    /* Enviar trama Ethernet al Servidor */
    int err = eth_send(eth_iface, server_addr, eth_type, payload, payload_len);
    if (err == -1) { exit(-1); }
    /* Recibir trama Ethernet del Servidor y procesar errores */
    len = eth_rcv(eth_iface, src_addr, eth_type, buffer, ETH_MTU, 2000);
    if (len <= 0) { fprintf(stderr, "%s: ERROR en eth_rcv()\n", argv[0]);
    } else { printf("Recibidos %d bytes del Servidor Ethernet\n", len); }
    /* Cerrar interfaz Ethernet */
    eth_close(eth_iface);
    return 0;
}
```



```
int main ( int argc, char * argv[] ) {
    /* Procesar parámetros */
    if (argc != 3) { printf("Uso: %s <iface> <tipo>\n", argv[0]); exit(-1); }
    /* Abrir la interfaz Ethernet */
    eth_iface_t * eth_iface = eth_open(iface_name);
    if (eth_iface == NULL) { exit(-1); }
    while(1) {
        /* Recibir trama Ethernet del Cliente */
        printf("Escuchando tramas Ethernet (tipo=0x%04x) ...\n", eth_type);
        unsigned char buffer[ETH_MTU];
        int data_len = eth_rcv(eth_iface, src_addr, eth_type, buffer, ETH_MTU, -1);
        if (data_len == -1) { break; }
        /* Enviar la misma trama Ethernet de vuelta al Cliente */
        int len = eth_send(eth_iface, src_addr, eth_type, buffer, data_len);
        if (len == -1) { break; }
        printf("Enviado %d bytes al Cliente Ethernet:\n", payload_len);
    }
    /* Cerrar interfaz Ethernet */
    eth_close(eth_iface);
    return 0;
}
```

❑ Acceder a los interfaces y enviar/recibir paquetes en “crudo” requiere permisos especiales:

❖ **Compilar con:** `rawnetcc /tmp/<prog> <f1.c> <f2.c> ...`

❑ **Ejemplo:** `rawnetcc /tmp/eth_client eth.c eth_client.c`

1. **Compila el ejecutable:**

```
/usr/bin/gcc -o /tmp/eth_client eth.c eth_client.c -lrawnet
```

2. **Otorga permisos especiales:**

```
/usr/sbin/setcap cap_net_raw,cap_net_admin=eip /tmp/eth_client
```

ó

```
/usr/bin/chown root.root /tmp/eth_client
```

```
/usr/bin/chmod 4755 /tmp/eth_client
```

❑ Depurar tráfico enviado con Wireshark

❖ `wireshark &`

❑ **Ejecutar cliente y servidor:** `/tmp/eth_server eth0 0x1234`

```
/tmp/eth_client eth0 0x1234 <MAC>
```

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

int main(int argc, char* argv[]) {
    int nums = atoi(argv[1]);
    /* Inicializar semilla para rand() */
    unsigned int seed = time(NULL);
    srand(seed);
    int i;
    for (i=0; i<nums; i++) {
        /* Generar número aleatorio entre 0 y RAND_MAX */
        int dice = rand();
        /* Número entero aleatorio entre 1 y 10 */
        dice = 1 + (int) (10.0 * dice / (RAND_MAX + 1.0));
        printf("%i\n", dice);
    }
}
```

Nivel de Aplicación:  
Protocolo de  
Encaminamiento

RIPv2

`ripv2_server.c`

Nivel de Transporte

UDP

`udp.h`      `udp_client.c`  
`udp.c`      `udp_server.c`

Nivel de Red

IP

`ip.h`      `ip_client.c`  
`ip.c`      `ip_server.c`

ARP

`arp.h`      `arp_client.c`  
`arp.c`

Nivel de Enlace

Ethernet

`eth.h`      `eth_client.c`  
`eth.c`      `eth_server.c`

Nivel Físico

librawnet

`rawnet.h`  
`timerms.h`  
`librawnet.so`

```
#include "eth.h"
#include "ipv4.h"
// void ipv4_addr_str(ipv4_addr_t addr, char str[]);
// int ipv4_str_addr(char* str, ipv4_addr_t addr);

int arp_resolve ( eth_iface_t * iface,
                  ipv4_addr_t dest,
                  mac_addr_t mac );
```