REDES DE SENSORES INALÁMBRICOS

CONTIKI OS – PARTE II

Javier Schandy
Inst. de Ingeniería Eléctrica, Facultad de Ingeniería
Universidad de la Republica (Uruguay)

AGENDA

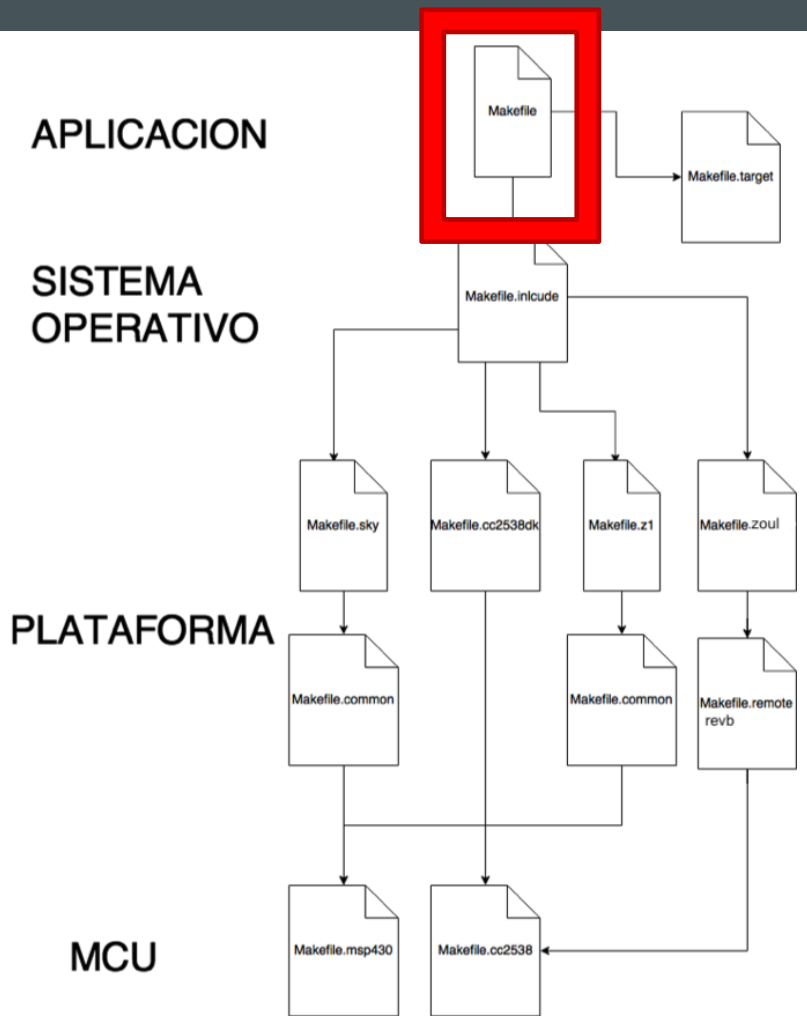
- Proceso de compilación de Contiki
- Stack de red de Contiki
- Recorrido de un paquete

PROCESO DE COMPILACIÓN

COMPILACIÓN

- `make`
  - Permite automatizar la compilación de proyectos
  - Determina:
    - Qué archivos del proyecto se deben compilar
    - Dependencias entre archivos
    - Qué comandos se deben usar para dicha compilación
  - Muy útil para proyectos de muchos archivos
  - Las reglas se encuentran en un archivo llamado `Makefile`

COMPILACIÓN



MAKEFILE APLICACIÓN

- En el directorio donde estemos trabajando en el proyecto siempre debe existir un archivo Makefile
- Debe incluir alguna de las siguientes sentencias:

1. Path al directorio donde se encuentra el sistema operativo.
   - `CONTIKI = ../.. (relativo)`
   - `CONTIKI = /home/user/contiki/ (absoluto)`
2. El Target para el cual estamos compilando.
   - ```
     ifndef TARGET
         TARGET=zoul
     endif
     ```
3. Los distintos proyectos que queremos compilar
   - `all: prueba`
   - `all: prueba prueba2`
     ```

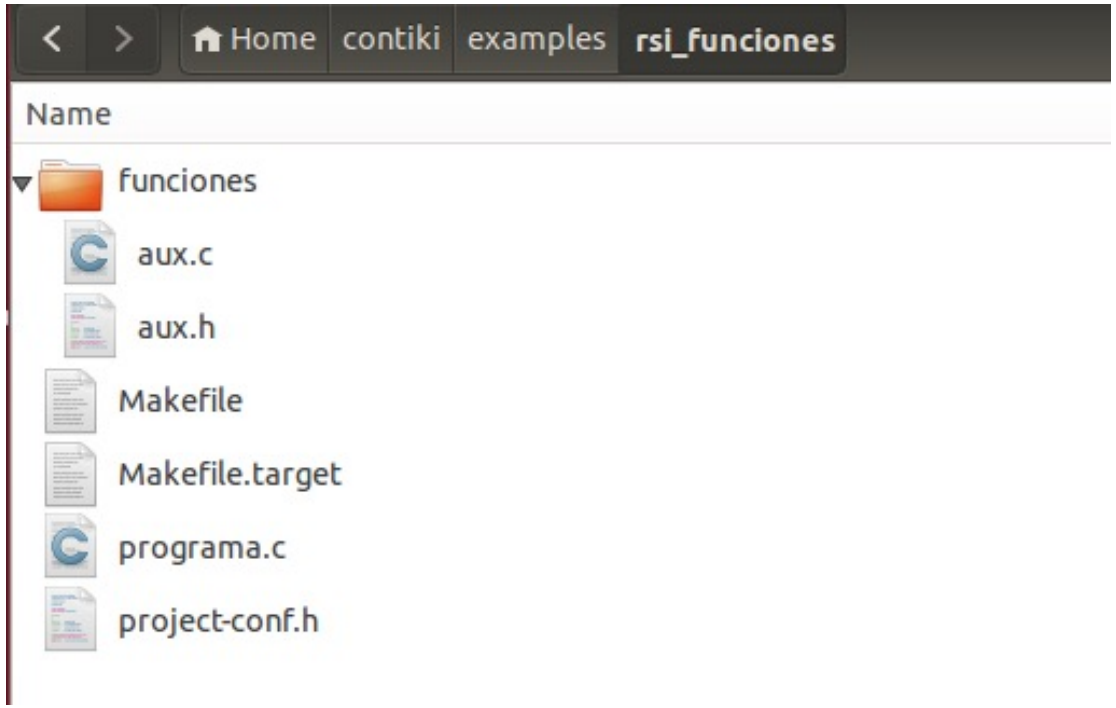
MAKEFILE APLICACIÓN

4. Se pueden hacer definiciones de distintas banderas del sistema operativo.
   - `CONTIKI_WITH_IPV6 = 1`
   - `CONTIKI_WITH_IPV4 = 1`
5. Hay distintas aplicaciones que se pueden incluir al proyecto.
   - `APPS=servreg-hack`
6. Ruta al archivo Makefile con las directivas generales del sistema operativo.
   - `include $(CONTIKI)/Makefile.include`

- Los puntos 1 y 6 **son obligatorios** y si no se encuentran el programa no va a compilar.
- Puntos 2, 3, 4 y 5 son equivalentes a:
  - `make TARGET=zoul prueba prueba2 CONTIKI_WITH_IPV4=1 APPS=servreg-hack`

TUTORIAL: FUNCIONES EN ARCHIVOS EXTERNOS

Supongamos que en el directorio del proyecto tenemos una carpeta `funciones` con los archivos `aux.c` y `aux.h`.



TUTORIAL: FUNCIONES EN ARCHIVOS EXTERNOS

aux.h

```
#include <stdint.h>

uint32_t suma(uint32_t a, uint32_t b);

uint32_t prod(uint32_t a, uint32_t b);

uint32_t duplicar(uint32_t a);
```

Agregar al Makefile las siguientes líneas:

```
CONTIKIDIRS += funciones

CONTIKI_SOURCEFILES += aux.c
```

aux.c

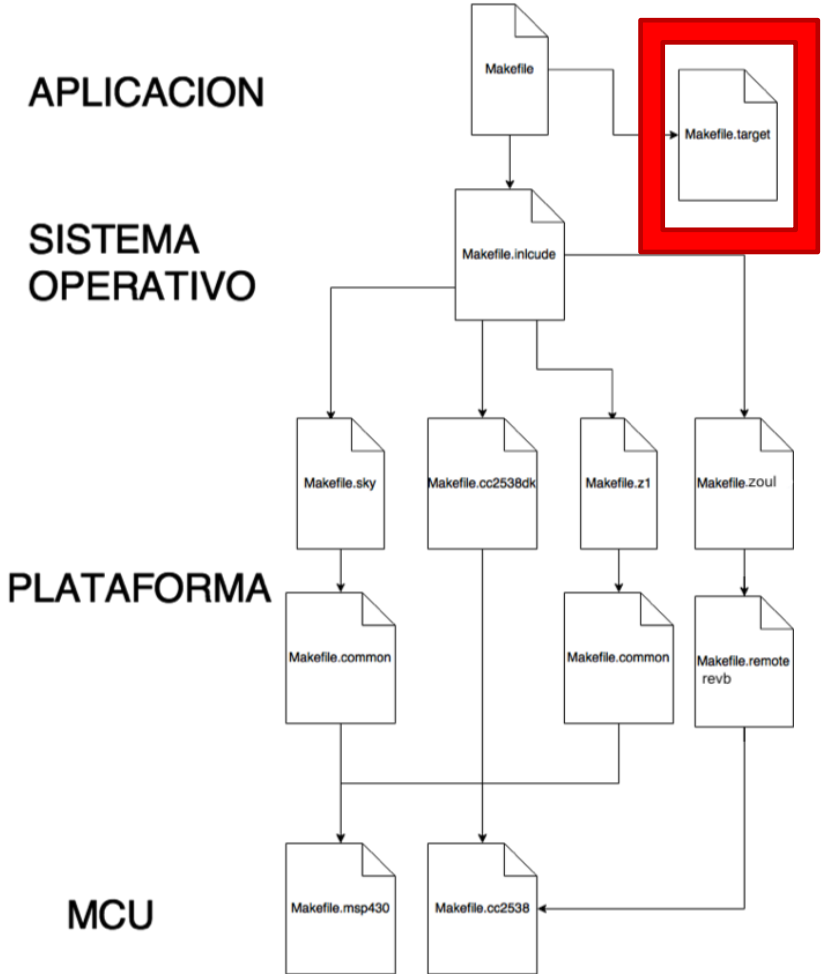
```
#include aux.h

uint32_t suma(uint32_t a, uint32_t b){
    return a+b;
}

uint32_t prod(uint32_t a, uint32_t b){
    return a*b;
}

uint32_t duplicar(uint32_t a){
    return 2*a;
}
```

COMPILACIÓN



MAKEFILE.TARGET

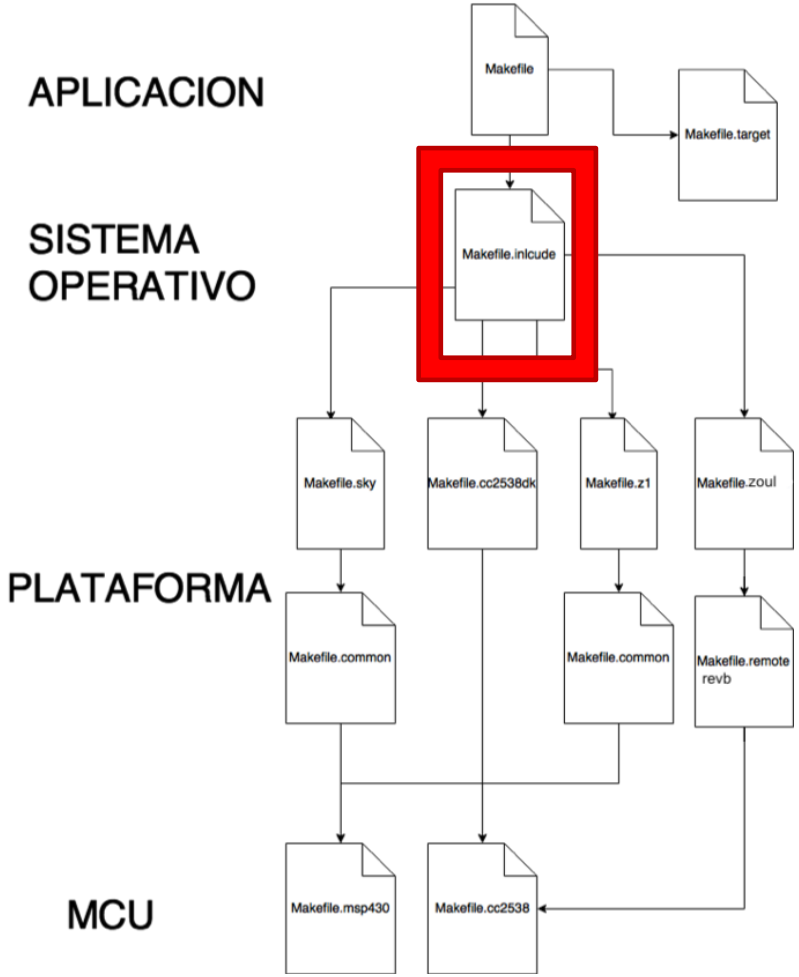
- Una forma alternativa para definir la plataforma es crear un archivo `Makefile.target` en el directorio de trabajo del ejemplo cuyo contenido sea la siguiente directiva:

```
TARGET = zoul
```

- Este archivo se puede crear automáticamente ejecutando el comando:

```
make TARGET=zoul savetarget
```

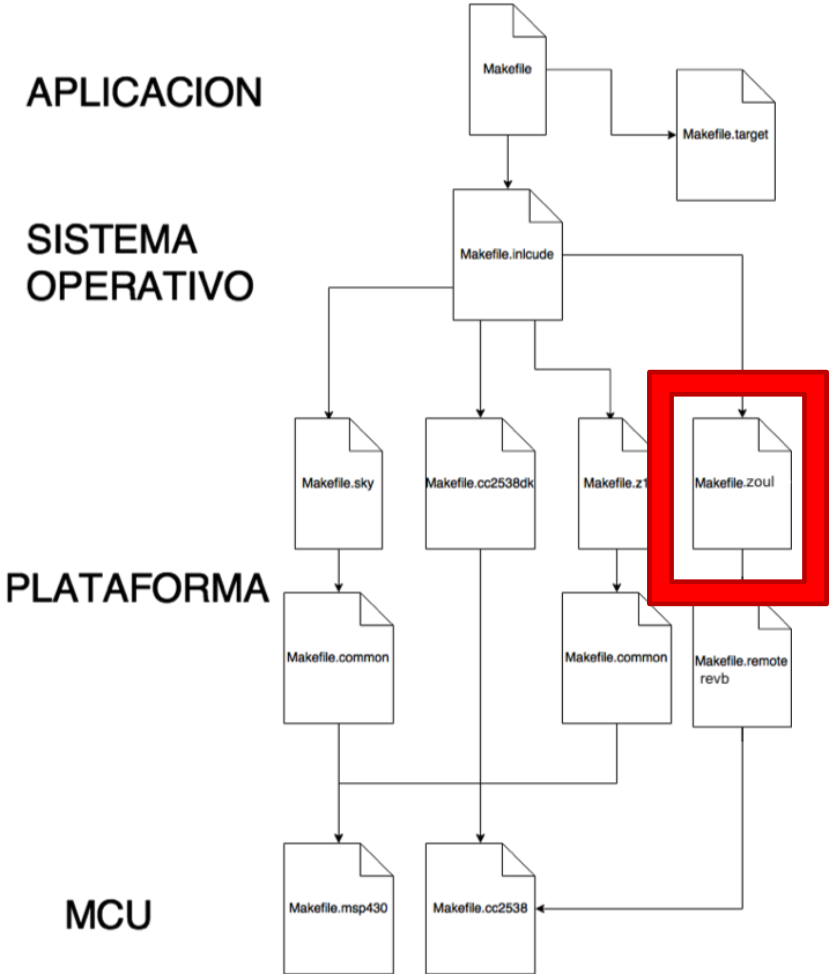
COMPILACIÓN



MAKEFILE.INCLUDE

- Contiene las directivas generales de la compilación.
- Muchas de ellas son chequeos de integridad para asegurarse que esté todo bien definido

COMPILACIÓN

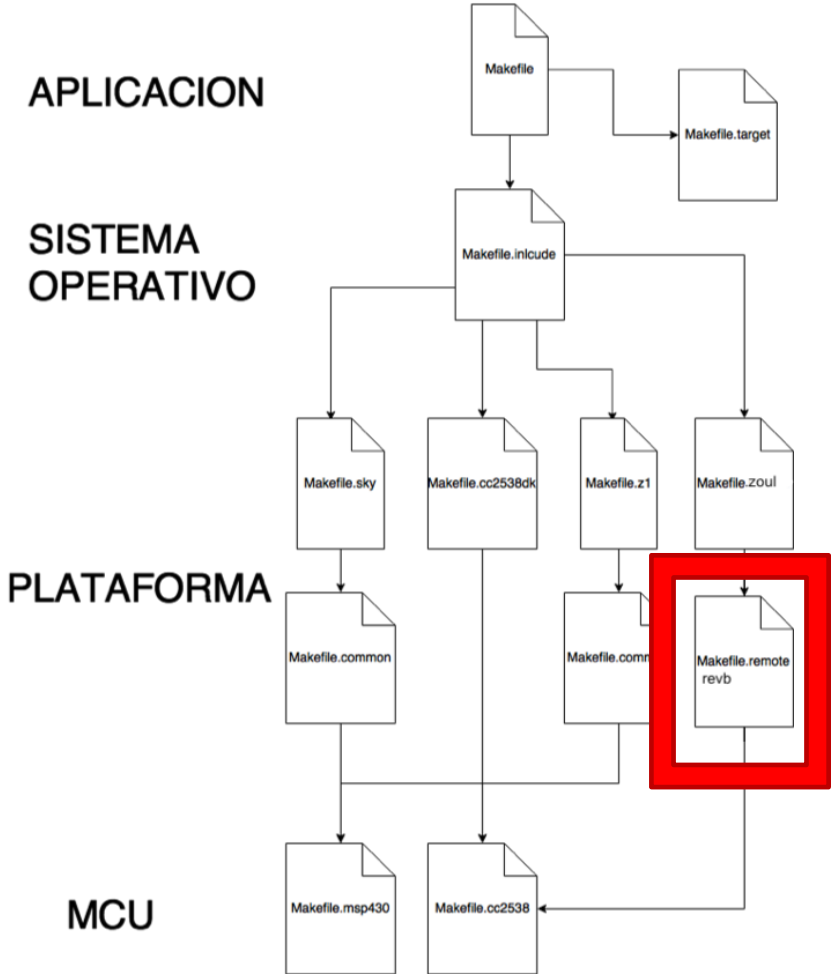


MAKEFILE.ZOUL

- Se encuentra en `$CONTIKI/platform/zoul`
- Define las directivas particulares de la plataforma, pero particularmente del nodo:

1. Los archivos particulares de la plataforma que hay que agregar al proyecto: sensores, botones, leds, etc.
   - `CONTIKI_TARGET_SOURCEFILES += contiki-main.c leds.c cc1200-zoul-arch.c adc-zoul.c button-sensor.c zoul-sensors.c`
2. Los módulos que usa la plataforma por defecto:
   - `MODULES += core/net core/net/mac core/net/ip core/net/mac/contikimac core/net/llsec core/net/llsec/noncoresec dev/cc1200dev/ds2411`
3. Incluye Makefile de la placa particular:
   - `include $(CONTIKI)/platform/zoul/remote-revb/Makefile.remote-revb`

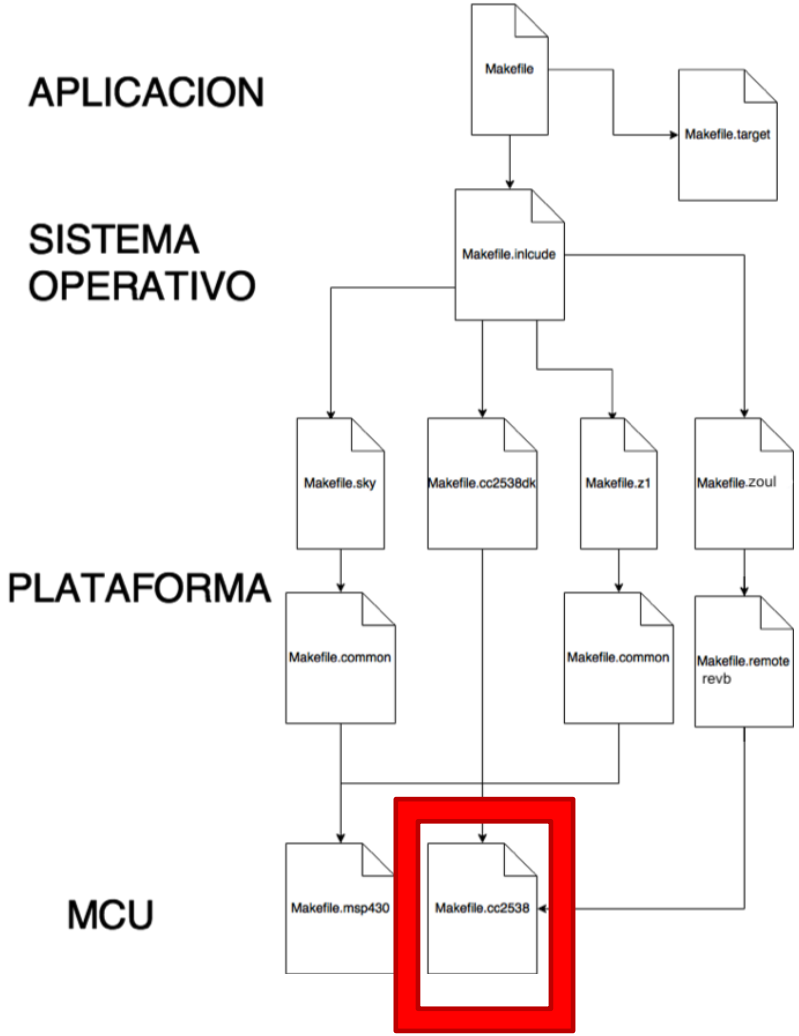
COMPILACIÓN



MAKEFILE.REMOTE-REVB

- Se encuentra en `$CONTIKI/platform/zoul/remote-revb`
- Define las directivas particulares de la placa remote-revb
  1. Los archivos fuente que se deben agregar al proyecto (leds, asignación de pines, manejo de alimentación, etc.)
     - `BOARD_SOURCEFILES += board.c antenna-sw.c mmc-arch.c rtcc.c leds-res-arch.c power-mgmt.c`
  2. Módulos
     - `MODULES += lib/fs/fat lib/fs/fat/option platform/zoul/fs/fat dev/disk/mmc`

COMPILACIÓN



MAKEFILE.CC2538

- Se encuentra en `$CONTIKI/cpu/cc2538`
- Define las directivas particulares del SoC.

STACK DE RED DE CONTIKI

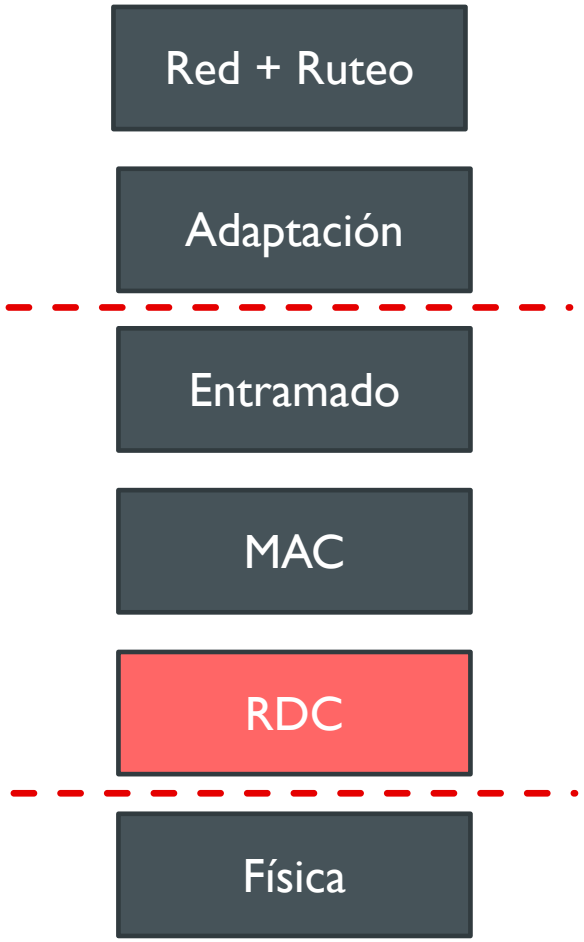
STACK DE RED

- Archivos en `core/net`

STACK DE RED: CAPA FÍSICA

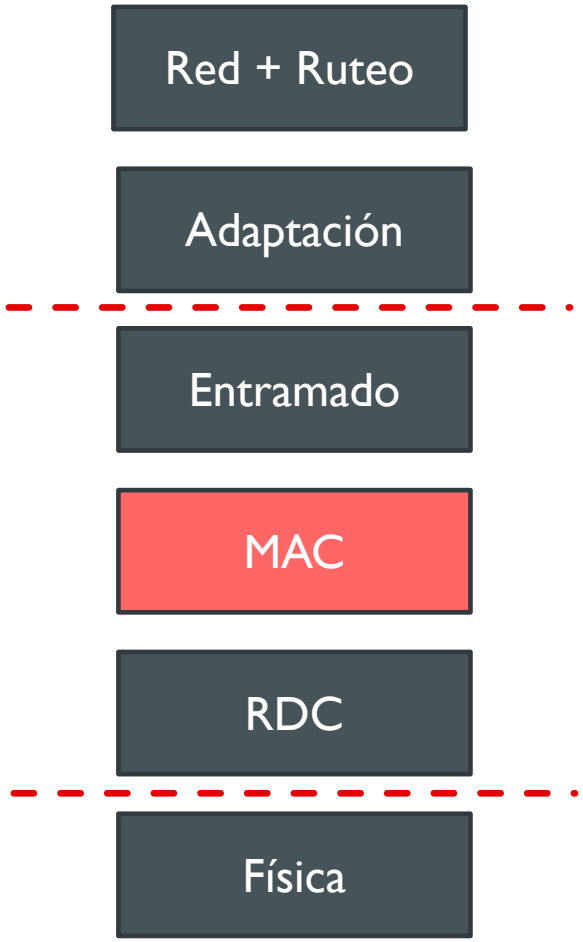
- Definida en el estándar IEEE 802.15.4.
- Direct Sequence Spread Spectrum.
- Modulación O-QPSK.
- Frecuencia 2,4 GHz / 900MHz / 868MHz.

STACK DE RED: RDC



- La radio tiene un gran consumo de corriente en TX y RX (~ 20-30mA a 3V).
- RDC maneja los tiempos en que la radio está encendida.
- Contiki implementa:
  - NullRDC
  - ContikiMAC
  - XMAC
  - LPP

STACK DE RED: MAC

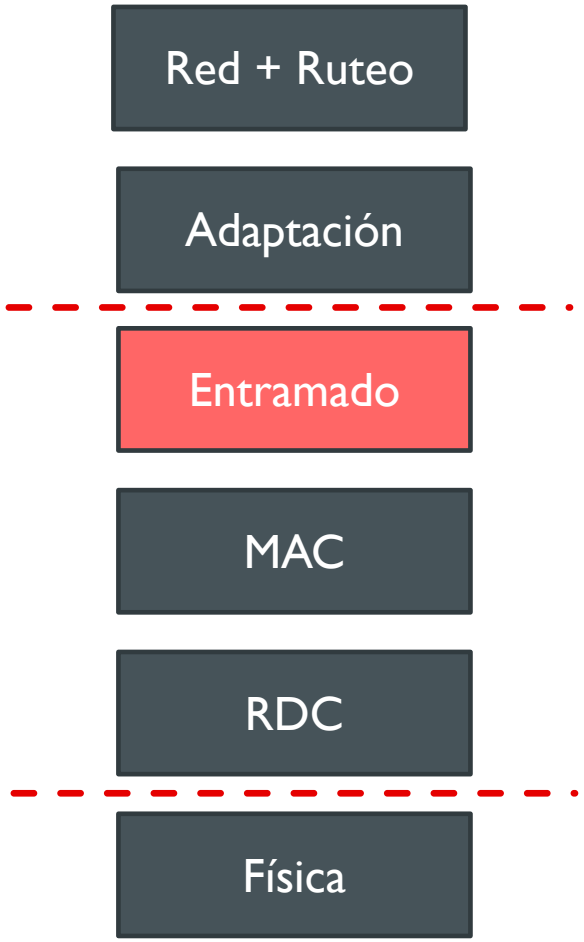


- Definido en el estándar IEEE 802.15.4.
- CSMA/CA no persistente
  - Arbitra el acceso al medio en canales compartidos
  - Cada nodo anuncia su intención de transmitir antes de hacerlo para evitar colisiones

STACK DE RED: CONFIGURACIÓN

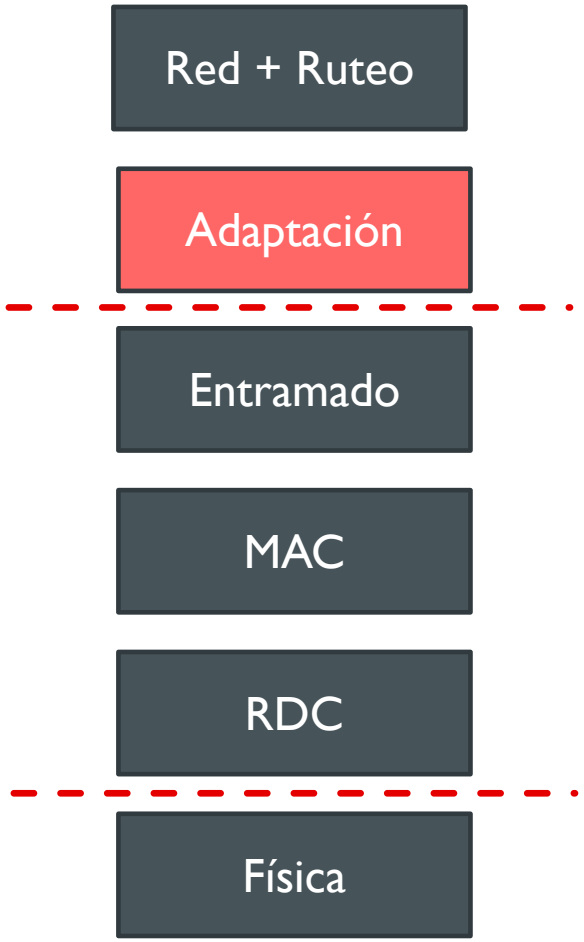
- Para ver los drivers por defecto de cada capa del NETSTACK se puede inspeccionar el archivo **`platform/zoul/contiki-conf.h`**
- Para modificarlos:
  - Agregar las siguientes líneas al project-conf.h en el directorio de compilación:
    - **`#define NETSTACK_CONF_RDC contikimac_driver`**
    - **`#define NETSTACK_CONF_MAC csma_driver`**
    - **`#define NETSTACK_CONF_RDC_CHANNEL_CHECK_RATE 8`**
    - **`#define NETSTACK_CONF_RADIO cc2538_rf_driver`**
    - **`#define NETSTACK_CONF_FRAMER framer_802154`**

STACK DE RED: ENTRAMADO



- El estándar define cuatro tipos de tramas:
  - Beacon
    - Balizas para señalizar o establecer la conguración de red
  - Datos
    - Se utilizan para transmitir la información relevante o carga útil.
  - Reconocimiento de paquetes (ACK)
    - Confirmar las recepciones exitosas
  - Comandos MAC
    - Control entre entidades pares de la capa de acceso al medio.

STACK DE RED: ADAPTACIÓN

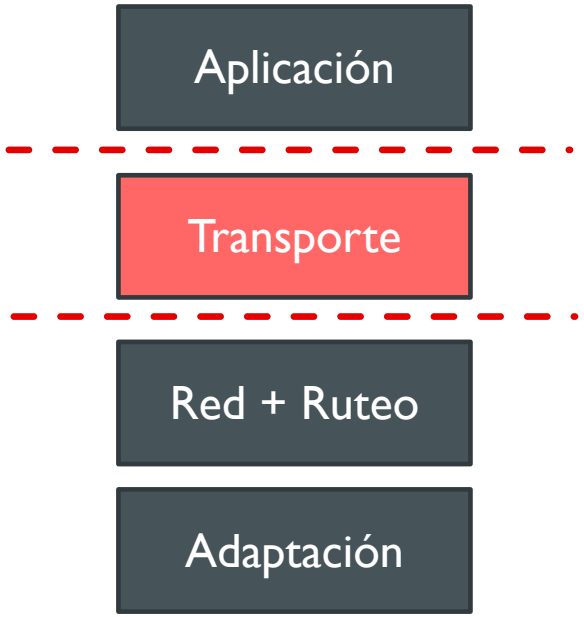


- IPv**6** over **Lo**w power **W**ireless **P**ersonal **A**rea **N**etworks:
  - Fragmentación y reensamblado de paquetes
  - Compresión de encabezados

STACK DE RED: RED Y RUTEO

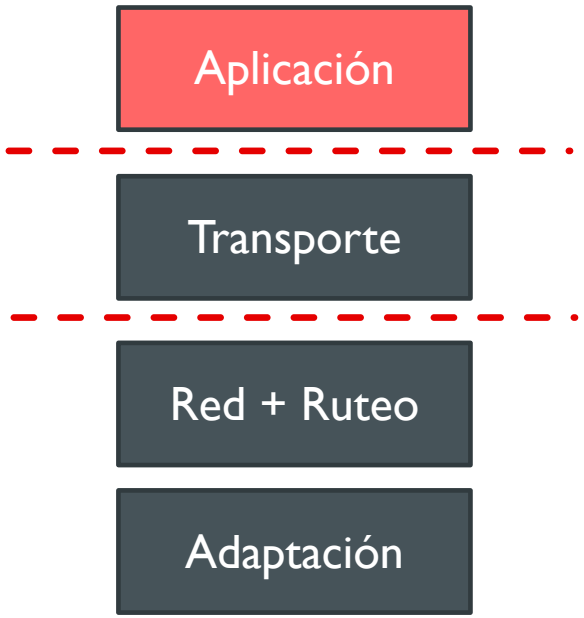
- uIP
  - Implementación del stack TCP/IP
  - Optimizado para sistemas con bajos recursos.
  - Soporte IPv4 e IPv6.
- Rime
  - Proporciona una serie de primitivas livianas y sencillas para establecer una comunicación.
  - El desarrollador las puede combinar de distintas formas según su objetivo.
- **R**outing **P**rotocol for **L**ow-Power and Lossy Networks:
  - Protocolo de ruteo proactivo basado en un vector de distancia calculado a partir de distintas métricas:
    - ETX
    - Energía
    - Cantidad de saltos

STACK DE RED: CAPA DE TRANSPORTE



- **U**ser **D**atagram **P**rotocol:
  - Protocolo de capa de transporte no orientado a conexión y no confinable
  - Liviano (menos transacciones)
  - Control de orden y duplicados se debe hacer a nivel de aplicación
- **T**ransmission **C**ontrol **P**rotocol

STACK DE RED: CAPA DE APLICACIÓN



- **Co**nstrained **A**pplication **P**rotocol:
  - Protocolo de aplicación diseñado para dispositivos de hardware reducido
  - Modalidad cliente-servidor
  - Muy reducido overhead de encabezados
  - Pocas transacciones (comparado con HTTP)
- MQTT
- LwM2M

TUTORIAL: EJEMPLO BROADCAST

```
#define UDP_PORT 1234

#define SEND_INTERVAL (20 * CLOCK_SECOND)
#define SEND_TIME     (random_rand() % (SEND_INTERVAL))

static struct simple_udp_connection broadcast_connection;

PROCESS(broadcast_example_process, "UDP broadcast example process");
AUTOSTART_PROCESSES(&broadcast_example_process);
static void
receiver(struct simple_udp_connection *c,
         const uip_ipaddr_t *sender_addr,
         uint16_t sender_port,
         const uip_ipaddr_t *receiver_addr,
         uint16_t receiver_port,
         const uint8_t *data,
         uint16_t datalen)
{
  printf("Data received on port %d from port %d with length %d\n",
         receiver_port, sender_port, datalen);
}
```

TUTORIAL: EJEMPLO BROADCAST

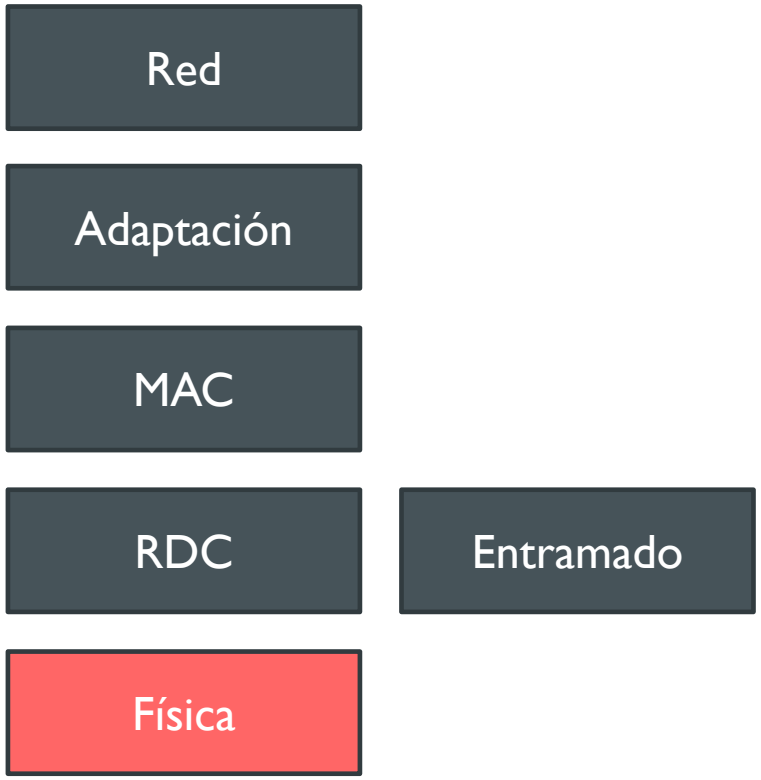
```
PROCESS_THREAD(broadcast_example_process, ev, data){
  static struct etimer periodic_timer;
  static struct etimer send_timer;
  uip_ipaddr_t addr;

  PROCESS_BEGIN();
  simple_udp_register(&broadcast_connection, UDP_PORT,
                      NULL, UDP_PORT,
                      receiver);
  etimer_set(&periodic_timer, SEND_INTERVAL);
  while(1) {
    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));
    etimer_reset(&periodic_timer);
    etimer_set(&send_timer, SEND_TIME);

    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&send_timer));
    printf("Sending broadcast\n");
    uip_create_linklocal_allnodes_mcast(&addr);
    simple_udp_sendto(&broadcast_connection, "Test", 4, &addr);
  }
  PROCESS_END();
}
```

RECORRIDO DE UN PAQUETE

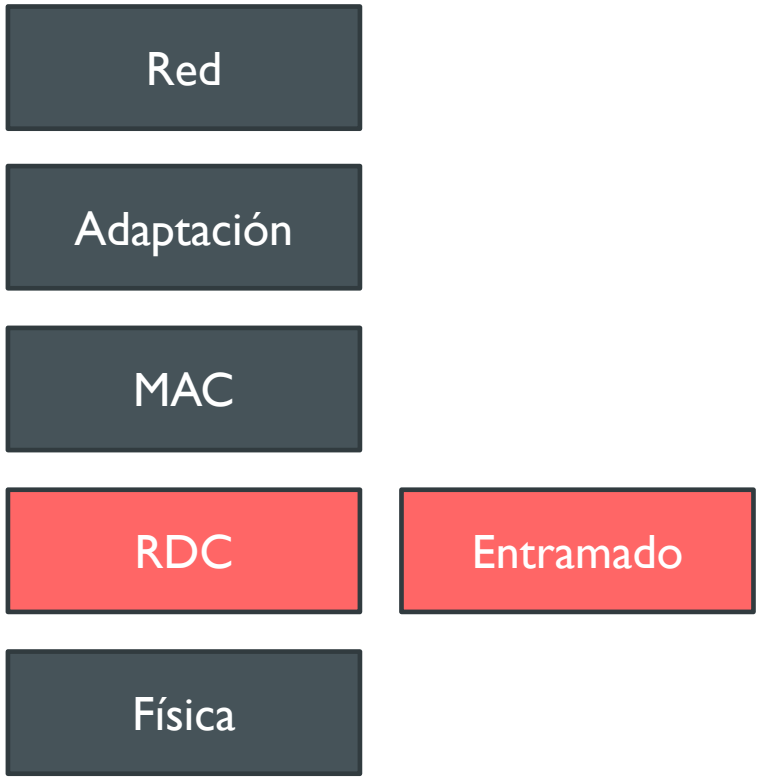
RECORRIDO DE UN PAQUETE



1) Llega un paquete a la radio

- Se genera una interrupción.
  - Se guarda el contenido del paquete en un buffer.
  - Se hace un **`process_poll()`** al proceso de la radio.
- Se corre el proceso de la radio
  - Se copian los datos del paquete al **`packetbuf`**.
  - Se lee el RSSI.
  - Se llama **`NETSTACK_RDC.input()`**

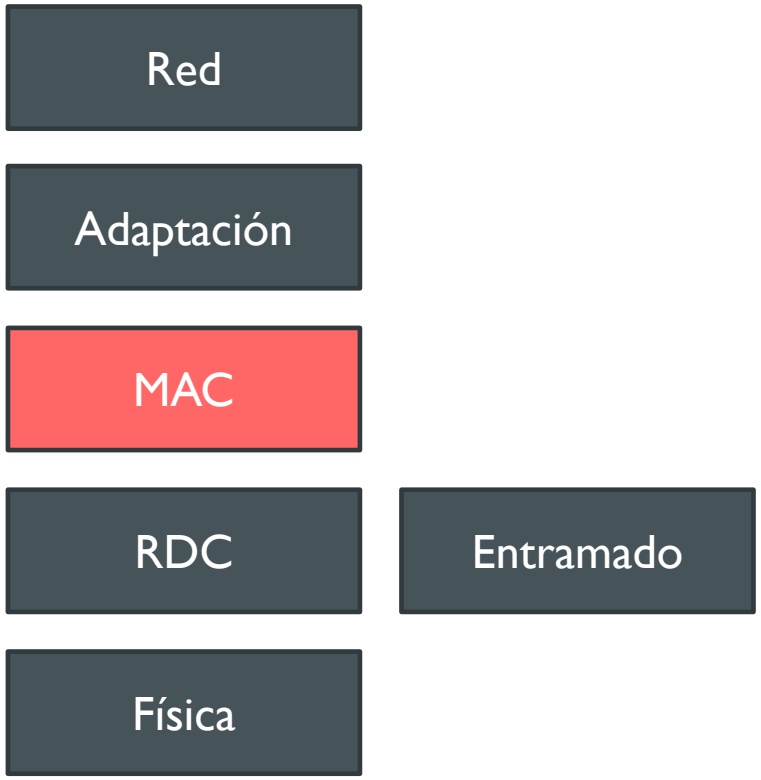
RECORRIDO DE UN PAQUETE



2) Manejo del RDC

- Se llama al Entramado para que analice los encabezados del paquete.
- Se revisa el bit de **`pending`**
  - Si esta en 1, se debe mantener la radio encendida para esperar el resto de la racha.
  - Si esta en 0, se lleva la radio a LPM.
- Se llama **`NETSTACK_MAC.input()`**

RECORRIDO DE UN PAQUETE



3) MAC

- Se llama **`NETSTACK_NETWORK.input()`**

RECORRIDO DE UN PAQUETE

4) Adaptación

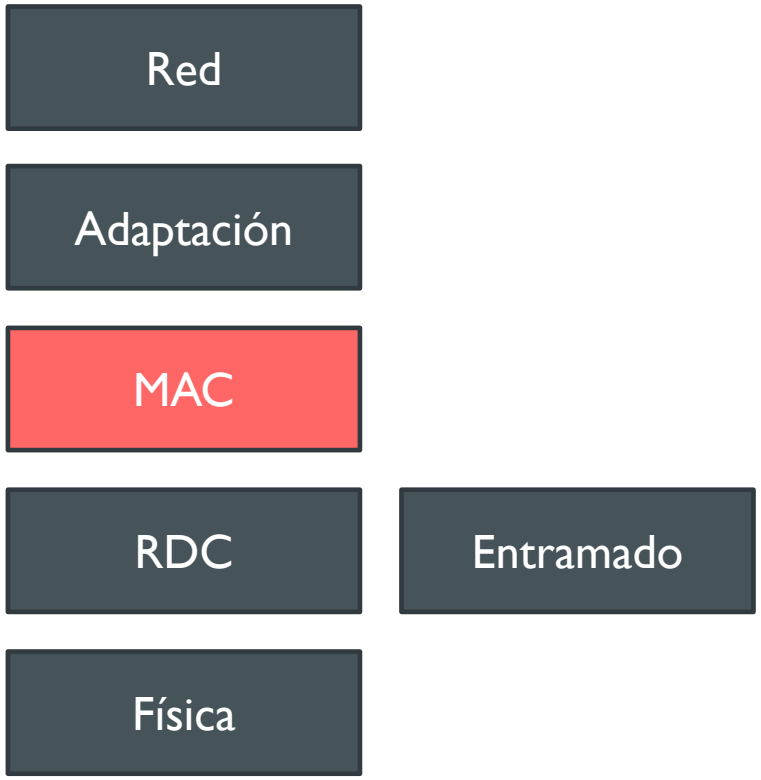
- Se descomprimen los encabezados.
  - Si es parte de un paquete IPv6 mas grande:
    - Copia los datos en un buffer de «fragmentation reassembly»
    - Espera el resto de los paquetes.
  - Si no, copia el paquete en el uip_buf y llama al stack IPv6 (`tcpip_input()`)

RECORRIDO DE UN PAQUETE (←)

1) Adaptación

- Se borra el **`packetbuf`**.
- Se comprime el encabezado del **`uip_buf`** y se copia todo al **`packetbuf`**.
- Si no entra en un **`packetbuf`**, se separa en los **`queuebuf`** que sea necesario.
- Se llama **`NETSTACK_MAC.send()`**.

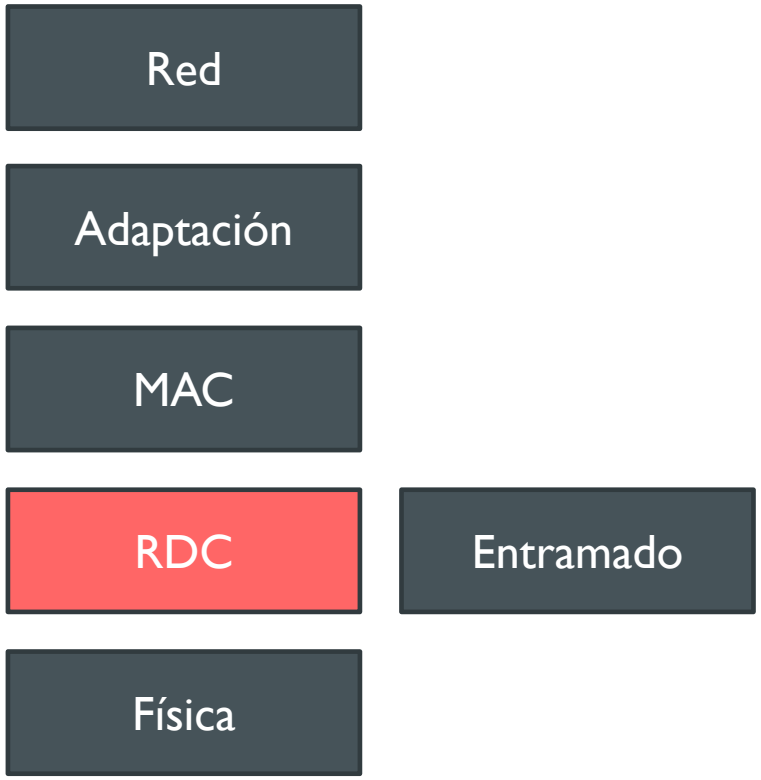
RECORRIDO DE UN PAQUETE (←)



2) MAC

- Se busca el receptor en la lista de vecinos.
- Se agrega el/los paquete/s a la cola de paquetes pendientes de el receptor.
- Se llama **`NETSTACK_RDC.send_list()`**

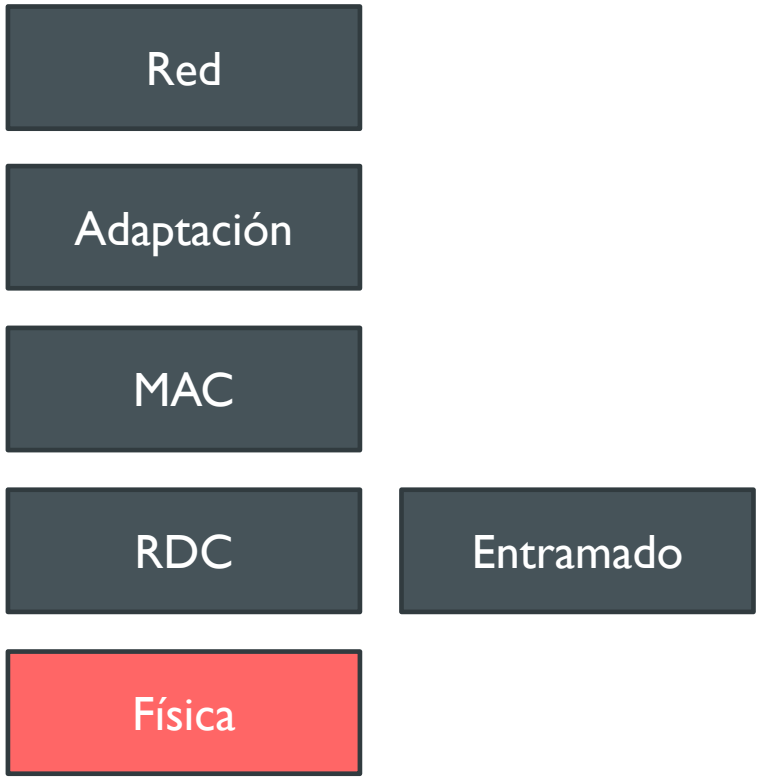
STACK DE RED



3) RDC

- Se despierta la radio.
- Se escucha el canal:
  - Si esta ocupado, le avisa a la capa MAC y vuelve a llevar la radio a LPM.
  - Si esta libre, envía los paquetes de la lista, esperando un ACK entre cada uno.

STACK DE RED



4) Se envía un paquete por la radio

- Le pasa los paquetes al hardware de la radio para que los ponga en el canal.

PLANIFICACIÓN DE CLASES

1. Introducción RSI
2. Modelado e introducción a IPv6
3. Plataforma de hardware
4. Plataforma de software: Contiki OS I
5. **Plataforma de software: Contiki OS II**
6. Capa de aplicación: CoAP / MQTT
7. Capa de red: RPL
8. Subcapa MAC
9. IEEE 802.15.4 / 6lowpan
10. Capa Física & antenas
11. IoT y las RSI

- ¿PREGUNTAS?