Layer 802.15.4 for LoRa Fabian

 $July\ 21st\ 2015$

1 Architecture

1.1 Global functioning

This documentation explains the implementation of the layer 802.15.4 between the radio layer and the contiki application. The main file is <code>layer802154_radio_lora.h</code> [Figure : 1]. This file provides the following functions:

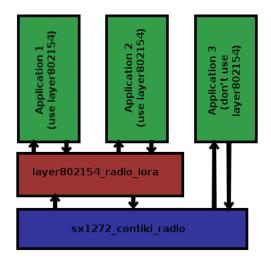


Figure 1: Layer 802.15.4

```
int layer802154_init(void);
Init the layer.
int layer802154_on(void);
Turn on the layer.
int layer802154_off(void);
Stop the radio.
```

```
int layer802154_channel_clear(void);
Call the function channel_clear() provided by the radio layer.
frame802154_lora_t layer802154_read();
Get and parse a 802.15.4 packet. A frame which contains the parsed packet is return.
SIGNALISATION_ON //Signalisation flag for the header
SIGNALISATION_OFF
DST_SHORT_FLAG //Short mode address for destination address in the header
DST_LONG_FLAG
```

int layer802154_send(const void *payload, unsigned short payload_len,
 uint8_t* destAddr, int signalisation, int dstShortSrcLongFlag);

Write the header and send the packet.

The function needs some parameters:

- payload: The payload of the packet.
- payload len: The length of the payload.
- destAddr: The MAC address of the destination.
- signalisation: the signalisation flag of the header.
- dstShortSrcLongFlag: The mode for the length of the source and destination address.

```
int layer802154_pending_packet(void);
```

Return if a packet is present into the radio layer.

1.2 802.15.4 packet specification

The structure frame 802154 lora t provides:

```
* \brief: Structure that contains the Frame Control Field
 */
typedef struct {
                                  //3 bits
  uint8_t _0_2_frame_type;
                                  //1 bit
  uint8_t _3_security_enabled;
                                  //1 bit
  uint8_t _4_frame_pending;
 \verb|uint8_t _5_ack_request|; //1 | bit|
  uint8_t _6_pan_id_compression; //1 bit //HERE WE DO NOT RESPECT THE STANDARD!
  We NEVER send PANID!
  //3 bits reserved
  uint8_t _10_11_dst_addr_mode; //2 bits
  uint8_t _12_13_frame_ver;
                                  //2 bits
  uint8_t _14_15_src_addr_mode; //2 bits
} frame802154_lora_fcf_t;
/**
```

```
* \brief: Structure that contains the 802.15.4 frame
 */
typedef struct {
 frame802154_lora_fcf_t fcf; //Frame control field
  uint8_t seq;//Sequence number
  uint8_t dest_addr[8];
                             //Destination address
  uint8_t src_addr[8];//Source address
  uint8_t *payload;
                    //Pointer to 802.15.4 frame payload
                     //Pointer to all the packet
  uint8_t *packet;
                     //Length of payload field
  int payload_len;
  int header_len;
                     //Length of header (-1 if an error occurs)
} frame802154_lora_t;
```

Note: dst_addr_mode must be different to src_addr_mode. The signalisation flag changes the last byte of the destination address. So, if the address is (0x00, 0x00), the message will contains (0x00, 0x80).

1	type (001)	security (0)	pe	ending (0)	ac	k (0)	pan_id (1)	(0)
(00)	(00) dst_addr_mode (10 or 11) version (01) src_add_mode (10 or 11)							.0 or 11)
MAC destination (0x00: 0 byte, 0x02: 2 bytes, 0x03: 8 bytes)								
MAC source (0x02: 2 bytes, 0x03: 8 bytes)								
Payload								

Figure 2: 802.15.4 packet structure

2 Using the layer

```
The following code is present in the file /platform/lorafabian/apps/frame_manager/frame_manager.c
```

2.1 Init the layer

The 802.15.4 layer works like the radio layer. We can use the following functions:

```
layer802154_init();
layer802154_on();
layer802154_off();
```

2.2 Read a packet

```
frame802154_lora_t frame = layer802154_read();
```

The user have the access on a frame which contains the parsed packet. So he can access to the attributes previously described. He has also access to the functions

int is_broadcast_addr(frame802154_lora_t *frame) which check if the packet is a broadcast message int is_my_mac(frame802154_lora_t *frame) which check if the destination MAC is the MAC of the contiki board (The MAC address is hardcoded in the file $frame802154_lora.c$).

```
frame802154_lora_t frame = layer802154_read();
size = frame.payload_len;
if(frame.header_len == -1)
  printf("Error: buffer is too small for headers");
else {
  //For the arduino
  int packetSize = size + frame.header_len;
  //Verify the destination of a message
  bool br_msg = is_broadcast_addr(&frame);
  bool my_mac = is_my_mac(&frame);
  if(br_msg) {
   printf("Broadcast message");
    if(!is_signaling(&frame) || debug_on_arduino)
      set_arduino_read_buf(frame.packet, packetSize);
  else if(my_mac) {
   printf("Message is for me");
    set_arduino_read_buf(frame.packet, packetSize);
  }
  else {
   printf("Message is not for me");
    if(debug_on_arduino)
      set_arduino_read_buf(frame.packet, packetSize);
}
```

2.3 Send a packet

This is how to send a COAP message:

```
/**
  * \brief: Send the coap_payload_beacon to layer802154
  */
void
coap_beacon_send_response() {
  uint8_t tx_buffer[512];

  size_t coap_packet_size;
  static coap_packet_t coap_request[1];

  unsigned short random_a = random_rand();
  coap_init_message(coap_request, COAP_TYPE_NON, COAP_POST, (coap_get_mid()+random_a)%65535);
  int sizeMSG = sizeof(coap_payload_beacon);
```

```
while(coap_payload_beacon[sizeMSG-1] == 0x00 && sizeMSG-1 != 0)
    --sizeMSG;
coap_set_payload(coap_request, (uint8_t *)coap_payload_beacon, sizeMSG);

coap_packet_size = coap_serialize_message(coap_request, (void *)(tx_buffer ));
int tx_buffer_index = coap_packet_size;
printf("We are sending the response to the coap beacon\n\r");

//We must write a destination address on 8 bytes
uint8_t destAddr[] = {0xfa,0x01};
layer802154_send(tx_buffer, tx_buffer_index, destAddr, SIGNALISATION_ON, DST_SHORT_FLAG);
}
```