Documentation Layer 802.15.4 pour LoRaFabian

25 Juin 2015

1 Architecture

1.1 Fonctionnement global

Cette documentation concerne la mise en place d'une couche 802.15.4 entre la couche radio et les applications du projet LoRaFabian. Les applications souhaitant utiliser cette couche doivent donc se servir du fichier *layer802154_radio_lora.h* [Figure : 1].
Ce fichier fournit alors à l'utilisateur la liste des fonctions et des constantes suivantes :

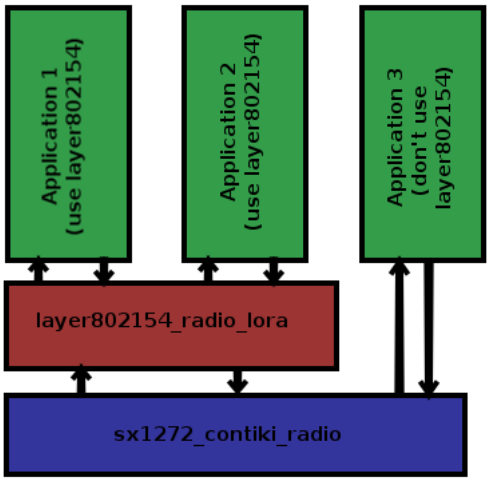

FIGURE 1 – Mise en place de la couche 802.15.4

```
int layer802154_init(void);
```

Pour initialiser le layer.

```
int layer802154_on(void);
```

Met en marche la couche radio.

```
int layer802154_off(void);
```

Arrête la couche radio.

```
int layer802154_channel_clear(void);
```

Appel la méthode channel_clear de la couche radio.

```
frame802154_lora_t layer802154_read(void *buf, unsigned short bufsize);
```

Récupère et parse un paquet 802.15.4. Une frame contenant le paquet parsé est renvoyé, de plus buf contient le paquet complet non parsé.

```
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);
```

Envoie un paquet et l'encapsule.
L'envoi d'un paquet nécessite la création de plusieurs arguments :

- payload : le message à encapsuler puis envoyer.
- payload_len : la taille du message.
- destAddr : l'adresse MAC du destinaire (SUR 8 OCTETS).
- signalisation : si le message est un message de signalisation.
- dstShortSrcLongFlag : le mode pour la taille de l'adresse du destinaire.

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

Retourne le nombre de paquet présent dans la couche radio.

1.2 Spécification d'un paquet 802.15.4

Lors de la lecture d'un paquet, un élément de type frame802154_lora_t est retourné. Ce qui permet à l'utilisateur de pouvoir accéder directement aux éléments suivants :

```
/**
 *  \brief: Structure that contains the Frame Control Field
 */
typedef struct {
  uint8_t _0_2_frame_type;       //3 bits
  uint8_t _3_security_enabled;   //1 bit
  uint8_t _4_frame_pending;      //1 bit
  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
  int payload_len;            //Length of payload field
  int header_len;             //Length of header (-1 if an error occurs)
} frame802154_lora_t;
```

Note : dst_addr_mode doit être différent de src_addr_mode. La signalisation se réalise en placant à 1 le premier bit de l'avant dernier octet de l'adresse de destination. Ainsi si l'adresse de destination est (0x00, 0x00), le message envoyé aura comme adresse de destination (0x00, 0x80).

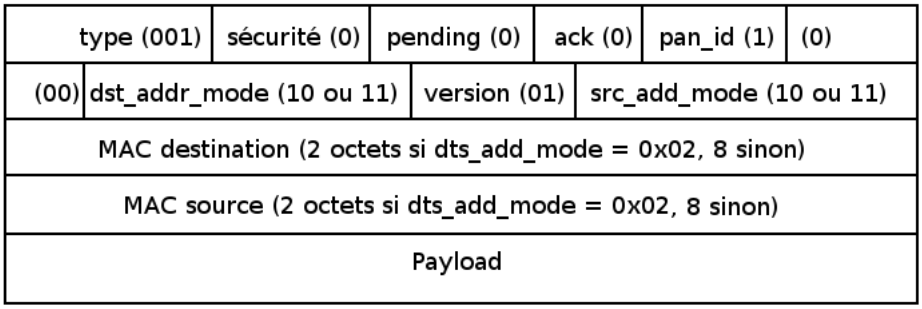

FIGURE 2 – Structure d'un paquet 802.15.4

2 Utilisation

Le code des exemples suivants est tiré du fichier *lorafab_beacon_answer_new.c* (présent dans */examples/lorafabian/lorafab_beacon_answer_new/*).

2.1 Initialisation

La couche 802.15.4 fonctionne de la même manière que la couche radio. Les fonctions destinés à l'initialisation sont :

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

2.2 Lecture d'un paquet

La lecture du paquet se réalise de cette manière :

```
frame802154_lora_t frame = layer802154_read(rx_msg, sizeof(rx_msg));
```

Le buffer rx_msg est utile pour récupérer le paquet depuis la couche radio. Il contiendra donc le paquet entier en sortie.

2.3 Traitement d'un paquet

L'utilisateur a ensuite l'accès à une frame qui contient le paquet parsé. Il peut donc accéder aux membres de la structure décrite plus haut. De plus, il a accès aux fonctions *int is_broadcast_addr(frame802154_lora_t *frame)* qui vérifie si le message est un message Broadcast et *int is_my_mac(frame802154_lora_t *frame)* qui vérifie que l'adresse de destination est la MAC de l'objet (cette adresse MAC est définie dans le fichier *frame802154_lora.c*). Voici un code d'exemple :

```
int size = frame.payload_len;
for(i = 0; i<size; i++)
  printf("%02x", frame.payload[i]);
printf("\n\r");

if(frame.header_len == -1)
  printf("Error: buffer is too small for headers");
else {

  //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");
  else if(my_mac) printf("Message is for me");
  else printf("Message is not for me");

  respond_if_coap_beacon(frame.payload, size);
}
```

2.4 Envoi d'un paquet

Voici un code d'exemple envoyant un message COAP :

```
char coap_payload_beacon[] = "{\"n\":\"zeta.s.ackl.io\"}";

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

  size_t coap_packet_size;
  //This way the packet can be treated as pointer as usual.
  static coap_packet_t coap_request[1];

  coap_init_message(coap_request, COAP_TYPE_NON,
                    COAP_POST, coap_get_mid());
```

```
  coap_set_payload(coap_request, (uint8_t *)coap_payload_beacon,
                   sizeof(coap_payload_beacon) - 1);

  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");

  //We must write a destination address on 8 bytes
  uint8_t destAddr[] = {0xfa,0x01,0x00,0x00,0x00,0x00,0x00,0x00};
  layer802154_send(tx_buffer, tx_buffer_index, destAddr,
                   SIGNALISATION_ON, DST_SHORT_FLAG);
}
```