Ilias RAMIM

Compte-Rendu RIO 201

Application IoT : Aide à l'endormissement

I] Définition et introduction de l'application IoT

Depuis des décennies, les français dorment moins. Pour la première fois, en 2017, le temps de sommeil moyen est descendu sous la barre symbolique des 7 heures, pour atteindre 6h42. Or, comme le montre le Dr Matthew Walker, neuroscientifique spécialiste du sommeil, dans son livre *Pourquoi nous dormons ?*, il ne faut pas négliger le sommeil. Dormir moins de 7h pendant 3 nuits consécutives réduit nos capacités cognitives de 20%. Cette réduction est exponentielle. Le sommeil est le seul moment où le corps : détruit les cellules cancéreuses ; détruit les substances toxiques produites la journée par le cerveau. Dormir suffisamment nous fait vieillir moins vite, prévient la maladie d'Alzheimer…

Des choses simples permettent d'améliorer la qualité de son sommeil. Le Dr Matthew Walker recommande ainsi, entre autres :

- De dormir entre 8 et 9 heures par nuit ;
- De se coucher et se lever à la même heure, pour imposer un rythme à son corps ;
- De dormir dans une chambre à 18 degrés.
- De n'aller dans son lit que pour dormir (ne pas rester dans son lit pour regarder un film par exemple), pour que le corps associe le lit au sommeil.

Nous pouvons ajouter à cela le fait d'avoir une période sans écran avant l'heure du coucher.

C'est ainsi qu'intervient l'application IoT que je propose. Cette application est composée d'un réseau de 5 appareils et capteurs :

- Un chauffage connecté ;
- Un capteur de température situé près du lit ;
- Une ampoule connectée ;
- Un accéléromètre positionné sur la porte de la chambre (détecte l'entrée d'une personne dans la chambre)
- Un « border router » faisant le lien entre ce réseau IoT et internet.

Ainsi, l'application fonctionne comme suit :

A l'heure du coucher, l'utilisateur peut, à distance (via le frontend SSH pour le prototype), initier le processus d'aide au sommeil.

Il peut également simplement entrer dans la chambre après une certaine heure programmable, ce qui sera détecté par l'accéléromètre, et initiera automatiquement le processus d'aide au sommeil (plus besoin de téléphone avant le coucher).

Ce principe d'activation automatique a pour but de favoriser une période de coucher sans écran, et régulière (déclenchement automatique du processus après une certaine heure lorsque l'utilisateur entre).

Description qualitative du processus d'aide au sommeil :

Lors de la mise en marche du processus d'aide au sommeil (forcée via le frontend, ou à l'entrée de l'utilisateur dans la chambre), plusieurs actions se déroulent en parallèle :

- La lumière est ajustée à 33% de la puissance maximale, pour limiter d'exposer l'utilisateur à trop de lumière avant l'endormissement ;
- Le chauffage récupère périodiquement la valeur de température du capteur près du lit. Si sa valeur est différente de 18 degrés, le chauffage régule la température jusqu'à atteindre la valeur consigne de 18 degrés.

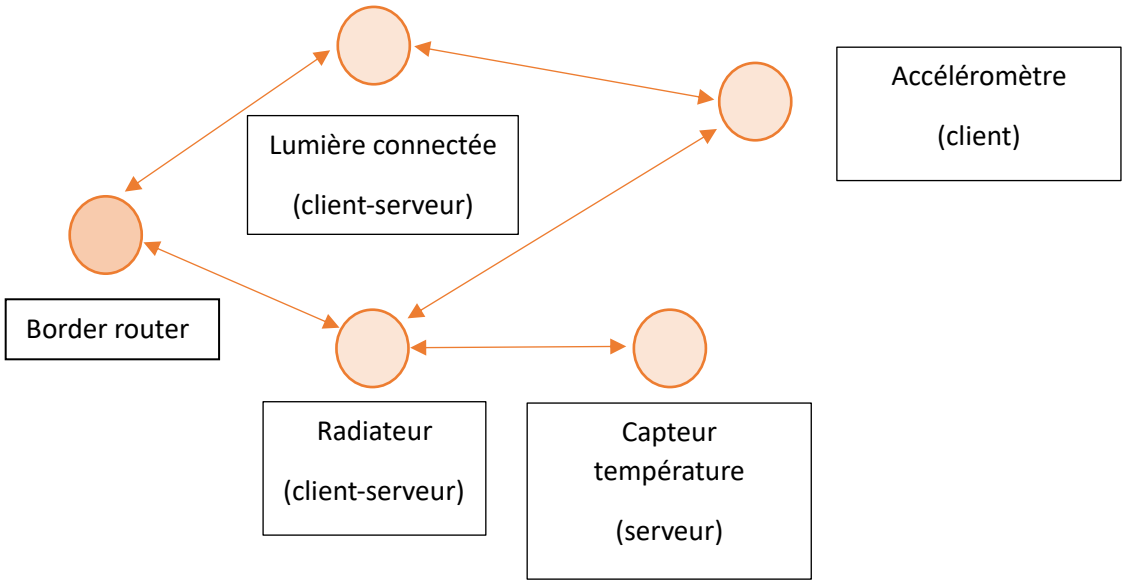
Perspective d'amélioration dans le futur :

Ce projet assez simple peut se voir améliorer dans le futur, grâce au développement d'une application utilisateur. Cette application enverrait à l'utilisateur des notifications à l'approche de l'heure du coucher, avec la possibilité de lancer le processus d'aide au sommeil à distance (remplaçant le frontend SSH).

Des ajouts de modules pourraient être envisagés, par exemple pour fermer automatiquement les volets/stores de la chambre si celle-ci en possède. Une extension pourrait rendre l'application compatible avec les assistants vocaux (type Alexa, Google Home) pour annoncer à l'utilisateur que la chambre est prête à la fin du processus, lui demander s'il souhaite éteindre les lumières, etc (ce qui éviterait là encore d'utiliser son smartphone avant d'aller dormir).

II] Architecture du système

Voici l'architecture du système (les nœuds du réseau):



: lien logique entre les nœuds du réseau

Commentaires sur les relations entre nœuds :

- La lumière et le radiateur sont connectés à l'accéléromètre (qui est client) car ce dernier envoie une requête POST à ces deux serveurs lorsque quelqu'un entre dans la pièce, pour débuter la procédure.
- Le radiateur est connecté au capteur de température (situé près du lit) pour pouvoir ajuster la commande du chauffage en fonction de cette température.
  La lumière et le radiateur sont connectés au border router pour que l'utilisateur puisse initier la procédure à distance via le frontend SSH.

Fonctions implémentées :

- Fonction permettant à l'utilisateur via le Frontend SSH d'initier la procédure d'aide au coucher.

Dans le cadre du prototype, cela consiste en une requête POST aux serveurs (radiateur + lumière) indiquant le début du processus. A la réception de la requête, les serveurs changent la valeur d'une variable globale «bedtime» à 1, indiquant qu'il faut commencer le processus.

- Fonction permettant au radiateur de communiquer avec le capteur de température

Lorsque la variable « bedtime » passe à 1, le radiateur commence alors la communication avec le capteur de température pour commander le chauffage et atteindre les 18 degrés (la commande du chauffage n'a pas été implémentée).

- Fonction permettant à l'accéléromètre (client) d'envoyer une alarme aux serveurs (radiateur + lumière) indiquant le début de la procédure.

III] Détail de l'implémentation et expériences :

- « Fonction permettant à l'utilisateur via le Frontend SSH d'initier la procédure d'aide au coucher. »

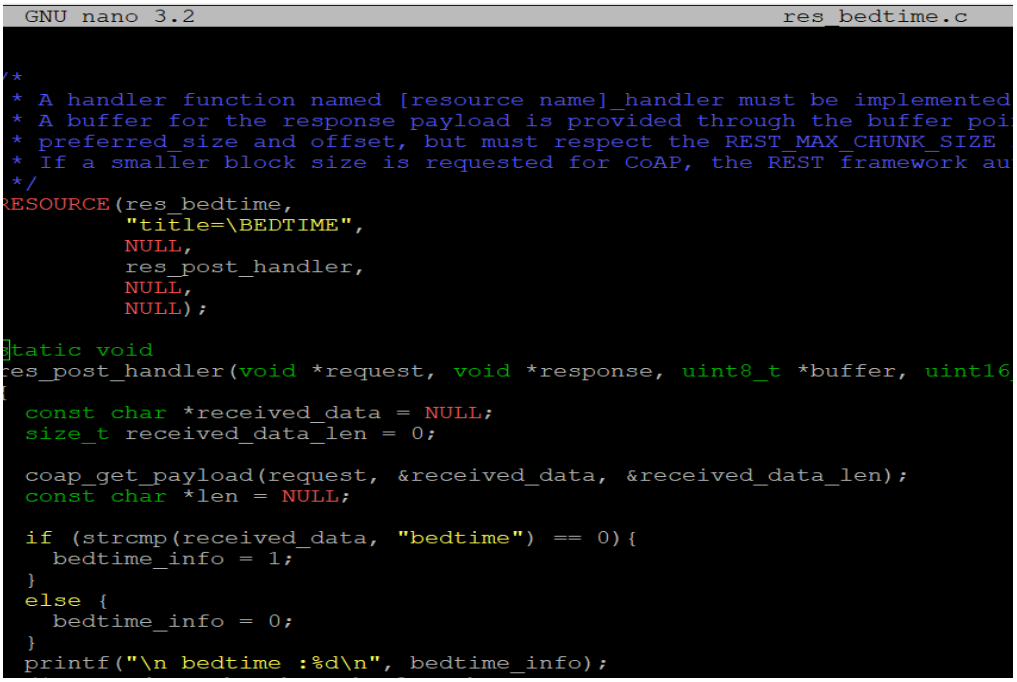
J'ai commencé par introduire une variable globale « bedtime_info », qui lorsqu'elle vaut 1, signifie qu'il faut initier le processus d'aide au sommeil.

```
  GNU nano 3.2                                          extern_var.h

extern int light_info;
extern int bedtime_info;
```

J'ai ensuite initialisé cette variable à 0 dans le code du client-serveur en C (puisque le radiateur/la lumière qui recevront les requêtes du frontend sont des clients-serveurs) :

```
  GNU nano 3.2                                          er-example-client.c

#include <string.h>
#include "contiki.h"
#include "contiki-net.h"
#include "er-coap-engine.h"
#include "dev/button-sensor.h"

#include "dev/serial-line.h"

#include "extern_var.h"
int bedtime_info = 0;
int previous_light_info = 0;
```

J'ai créé une nouvelle ressource res_bedtime, qui permet de mettre à jour la valeur de bedtime_info à 1 lorsque la requête POST « bedtime » est reçue par le serveur. La fonction res_post_handler récupère le payload de la requête dans «received data » et compare cette chaîne de caractères à «bedtime » :


GNU nano 3.2 res_bedtime.c
/*
* A handler function named [resource name]_handler must be implemented
* A buffer for the response payload is provided through the buffer poi
* preferred_size and offset, but must respect the REST_MAX_CHUNK_SIZE
* If a smaller block size is requested for CoAP, the REST framework au
*/
RESOURCE(res_bedtime,
"title=\BEDTIME",
NULL,
res_post_handler,
NULL,
NULL);
static void
res_post_handler(void *request, void *response, uint8_t *buffer, uint16
{
const char *received_data = NULL;
size_t received_data_len = 0;
coap_get_payload(request, &received_data, &received_data_len);
const char *len = NULL;
if (strcmp(received_data, "bedtime") == 0){
bedtime_info = 1;
}
else {
bedtime_info = 0;
}
printf("\n bedtime :%d\n", bedtime_info);

- Fonction permettant au radiateur (client-serveur) de communiquer avec le capteur de température (serveur) (Pour le prototype, j'ai pris la valeur du capteur de luminosité, mais la démarche serait la même pour la température en conditions réelles)

J'ai modifié le code du client pour qu'il n'envoie les requêtes de température au capteur seulement si la variable bedtime_info vaut 1 :

```
  GNU nano 3.2                                          er-example-client.c

  SENSORS_ACTIVATE(button_sensor);
  printf("Press a button to request %s\n", service_urls[uri_switch]);
#endif

  while(1) {
    PROCESS_YIELD();
    if(etimer_expired(&et)) {
      if(bedtime_info ==1){
        printf("--Toggle timer--\n");
        /* prepare request, TID is set by COAP_BLOCKING_REQUEST() */
        coap_init_message(request, COAP_TYPE_CON, COAP_GET, 0);
        coap_set_header_uri_path(request, service_urls[3]);

        const char msg[] = "Toggle!";

        coap_set_payload(request, (uint8_t *)msg, sizeof(msg) - 1);

        PRINT6ADDR(&server_ipaddr);
        PRINTF(" : %u\n", UIP_HTONS(REMOTE_PORT));
        COAP_BLOCKING_REQUEST(&server_ipaddr, REMOTE_PORT, request,
                              client_chunk_handler);

        printf("\n--Done--\n");
      }
        etimer_reset(&et);
```

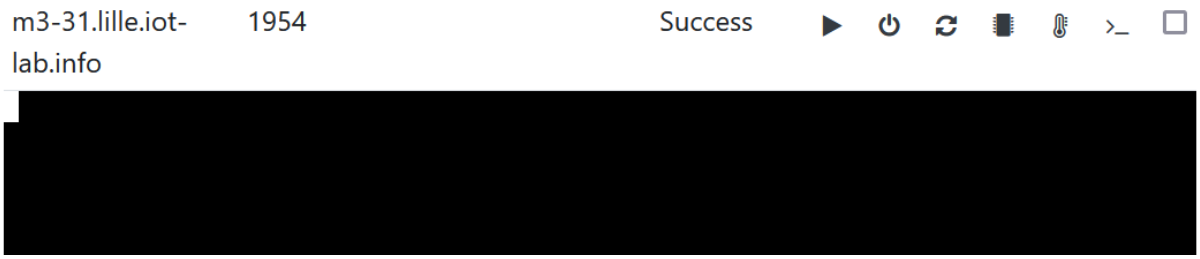
J'ai également modifié le code du serveur pour qu'il affiche clairement la température (ici luminosité = température, car les nœuds que j'utilise n'ont pas de capteur de température) :

```
client_chunk_handler(void *response)
{
  const uint8_t *chunk;

  int len = coap_get_payload(response, &chunk);

  printf("luminosity value :%.*s", len, (char *)chunk);
}
```

Nous avons désormais la valeur de température, que l'on pourra comparer avec la température consigne de 18 degrés.

Vérifions maintenant que l'ensemble des deux fonctions fonctionne :

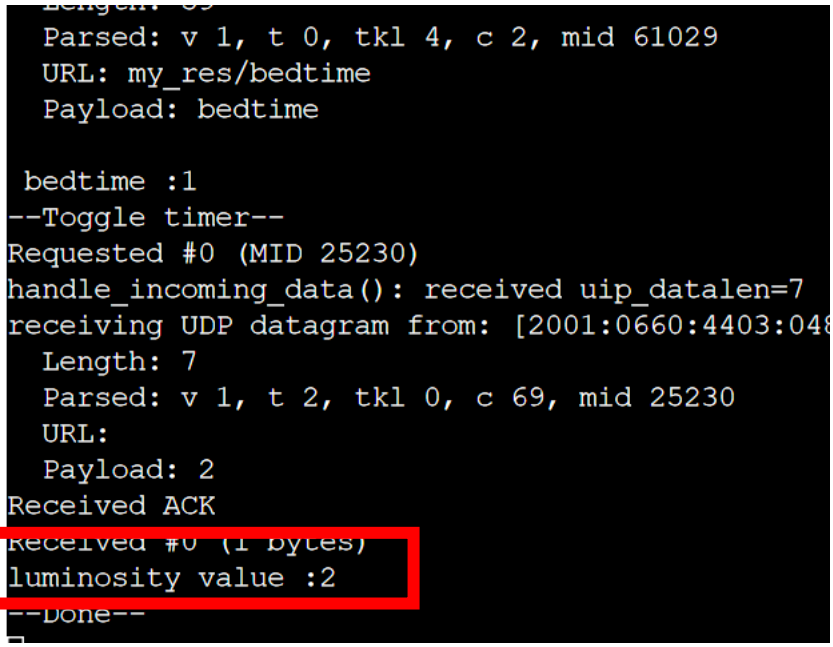
D'abord, lorsque le client-serveur (radiateur) est en route avec bedtime_info = 0, le client n'envoie aucune requête :



Faisons alors des tests pour voir si seule une requête POST avec le payload « bedtime » met à jour la valeur de betime_info :

```
riotp18@lille:~/iot-lab/parts/contiki/examples/iotlab/04-er-rest-example$ aiocoap-client coap://[2001:660
4]:5683/my_res/new_alarm
Bedtime is 0
(No newline at end of message)
riotp18@lille:~/iot-lab/parts/contiki/examples/iotlab/04-er-rest-example$ aiocoap-client coap://[2001:660
4]:5683/my_res/bedtime -m POST --payload "test"
riotp18@lille:~/iot-lab/parts/contiki/examples/iotlab/04-er-rest-example$ aiocoap-client coap://[2001:660
4]:5683/my_res/new_alarm
Bedtime is 0
(No newline at end of message)
riotp18@lille:~/iot-lab/parts/contiki/examples/iotlab/04-er-rest-example$ aiocoap-client coap://[2001:660
4]:5683/my_res/bedtime -m POST --payload "bedtime"
riotp18@lille:~/iot-lab/parts/contiki/examples/iotlab/04-er-rest-example$ aiocoap-client coap://[2001:660
4]:5683/my_res/new_alarm
Bedtime is 1
(No newline at end of message)
riotp18@lille:~/iot-lab/parts/contiki/examples/iotlab/04-er-rest-example$
```

On remarque qu'une requête POST avec le mauvais payload (ici « test ») ne modifie pas la valeur de bedtime. En revanche, le payload « bedtime » met à jour bedtime_info à 1. Je précise avoir modifié la ressource new_alarm pour qu'elle affiche la valeur de bedtime_info. Maintenant que bedtime_info vaut 1, vérifions que le client récupère les données de température :



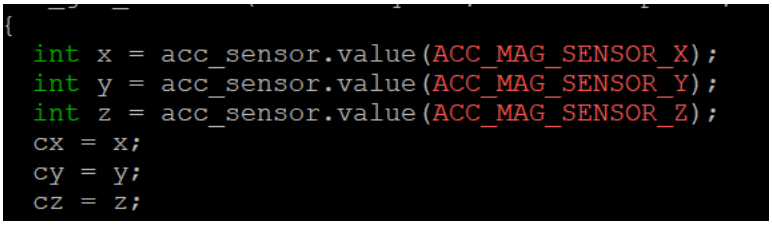
On voit bien les requêtes, contenant la valeur de la température (qui est ici la valeur du capteur de luminosité, d'où « luminosity value »).

Les fonctions implémentées fonctionnent correctement.

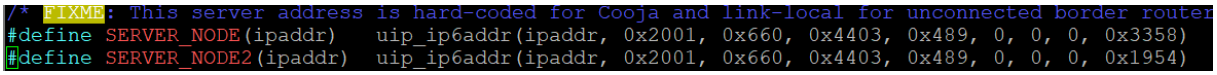
- Fonction permettant à l'accéléromètre (client) d'envoyer une alarme aux serveurs (radiateur + lumière) indiquant le début de la procédure.

Dans le code du client (accéléromètre), j'ai commencé par définir les valeurs cx, cy, cz (pour current x,y,z) et les valeurs px,py,pz (pour previous x,y,z).

J'ai ensuite mis les valeurs de l'accéléromètre dans les valeurs courantes cx,cy,cz :



J'ai également défini les deux adresses IPv6 de chaque serveur :



J’ai ensuite modifié le code principal du client pour qu’il envoie des requêtes POST aux deux serveurs, pour leur indiquer que nous entrons en mode «bedtime», seulement si la valeur de l’accéléromètre a changé :

```
while(1) {
  PROCESS_YIELD();
  if(etimer_expired(&et)) {
    cx = acc_sensor.value(ACC_MAG_SENSOR_X);
    cy = acc_sensor.value(ACC_MAG_SENSOR_Y);
    cz = acc_sensor.value(ACC_MAG_SENSOR_Z);
    if((px != 0) &&(py!=0)&&(pz!=0)){
       if((cx!=px)||(py!=cy)||(pz!=cz)){
          bedtime_info = 1;
          printf("\n alarme: \n avant : %d,%d,%d\n apres : %d,%d,%d \n", px, py, pz, cx, cy, cz);
          px = cx;
          py = cy;
          pz = cz;

          coap_packet_t request;

          coap_init_message(&request, COAP_TYPE_CON, COAP_POST, 0);
          coap_set_header_uri_path(&request, service_urls[5]);
          coap_set_payload(&request, "bedtime", 25);
          COAP_BLOCKING_REQUEST(&server_ipaddr, REMOTE_PORT, &request, &client_chunk_handler);

          coap_init_message(&request, COAP_TYPE_CON, COAP_POST, 0);
          coap_set_header_uri_path(&request, service_urls[5]);
          coap_set_payload(&request, "bedtime", 25);
          COAP_BLOCKING_REQUEST(&server_ipaddr2, REMOTE_PORT, &request, &client_chunk_handler);
       }
       else{
          printf("ok: %d,%d,%d", cx, cy, cz);
       }
    }
    else{
       px = cx;
       py = cy;
       pz = cz;
    }
      etimer_reset(&et);
```

Nous pouvons constater que les requêtes s’envoient bien lorsque la valeur de l’accéléromètre change :

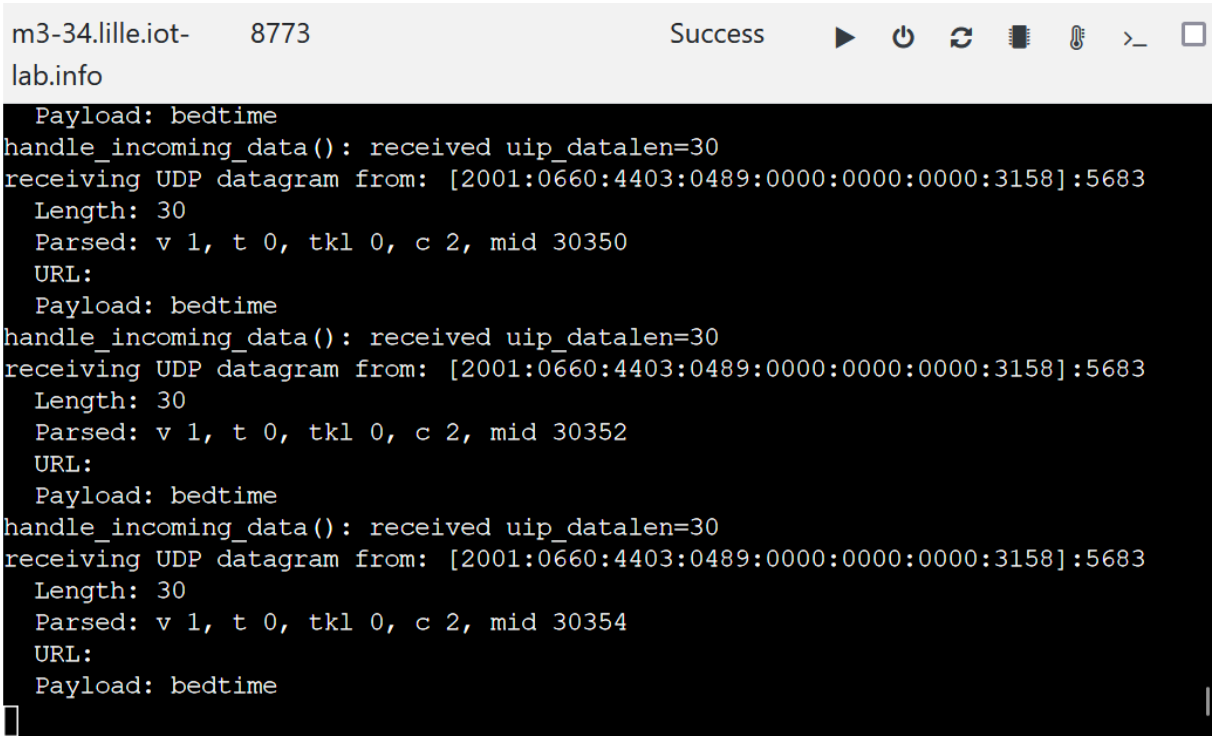
```
 apres : -51,1037,36
Requested #0 (MID 30301)
handle_incoming_data(): received uip_datalen=4
receiving UDP datagram from: [2001:0660:4403:0489:0000:0000:0000:1954]:5683
  Length: 4
  Parsed: v 1, t 2, tkl 0, c 69, mid 30301
  URL:
  Payload:
Received ACK
Received #0 (0 bytes)
temperature value :Requested #0 (MID 30302)
handle_incoming_data(): received uip_datalen=4
receiving UDP datagram from: [2001:0660:4403:0489:0000:0000:0000:8773]:5683
  Length: 4
  Parsed: v 1, t 2, tkl 0, c 132, mid 30302
  URL:
  Payload:
Received ACK
Received #0 (0 bytes)
```

Terminal du client (accéléromètre)

m3-31.lille.iot-lab.info 1954 Success

```
  Payload: bedtime

 bedtime :1
handle_incoming_data(): received uip_datalen=45
receiving UDP datagram from: [2001:0660:4403:0489:0000:0000:0000:3158]:5683
  Length: 45
  Parsed: v 1, t 0, tkl 0, c 2, mid 30327
  URL: my_res/bedtime
  Payload: bedtime

 bedtime :1
handle_incoming_data(): received uip_datalen=45
receiving UDP datagram from: [2001:0660:4403:0489:0000:0000:0000:3158]:5683
  Length: 45
  Parsed: v 1, t 0, tkl 0, c 2, mid 30329
  URL: my_res/bedtime
  Payload: bedtime

 bedtime :1
```

Terminal du client-serveur (radiateur)



Terminal du serveur (lumière connecté)

Chaque serveur reçoit le payload «bedtime», amorçant le processus d'aide au sommeil.

IV] Évaluation des performances

Je n'ai pas eu le temps de comparer expérimentalement http et CoAP, mais je peux néanmoins proposer une démarche expérimentale et une approche théorique pour les comparer.

Nous avons vu en cours que http est plus long que coap, nous pourrions alors exécuter en parallèle des clients-serveurs CoAP et des clients-serveurs http, et comparer les temps que mettent les paquets à être transmis.

CoAP devrait également être adapté aux applications IoT et consommer moins de batterie que http. Nous pourrions exécuter deux instances en parallèle : une en http, et une en CoAP, et comparer la consommation d'énergie des deux instances.