Section I :

Introduction :

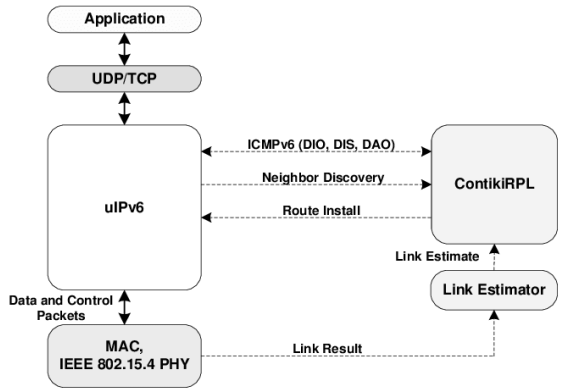


Figure 1 Structure de contiki os/cooja

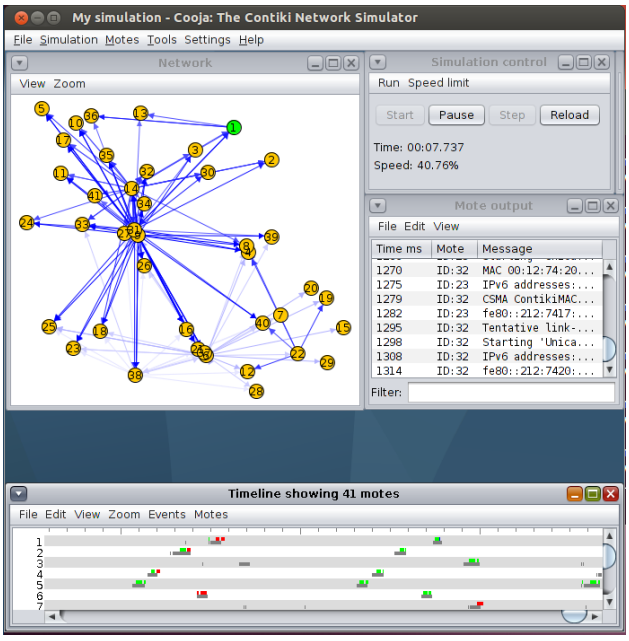


Figure 2 réseau Rpl

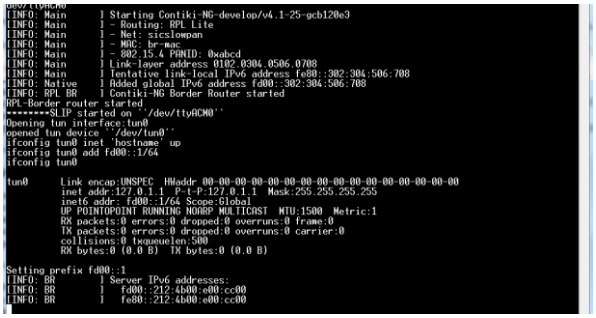
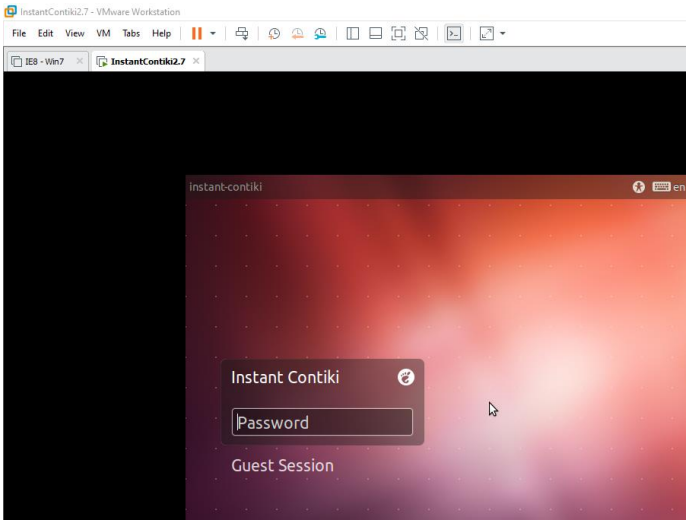
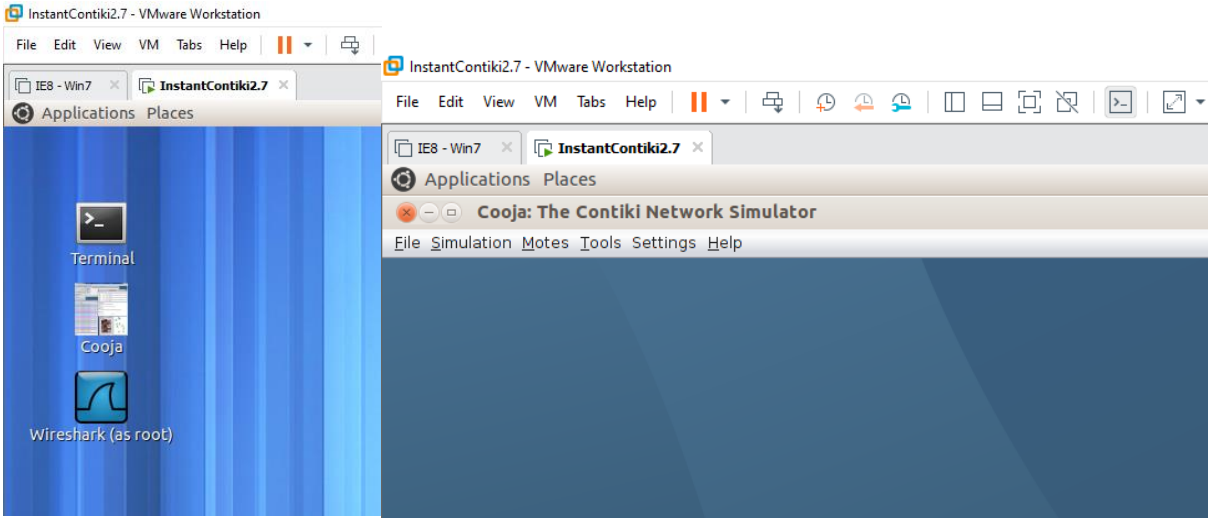


Figure 3 démarrage et console de conitki

Contiki est basé sur ubuntu avec comme mot de passe par défaut user





Par défaut le clavier est en qwerty donc il faut le régler paramètres -> préférence -> keyboard input

Interface :

Carte des capteurs

Graphique de réseau

Graphiques liés aux capteurs

Celles-ci dépendent de la disponibilité d'un type particulier de capteur sur les motos.

Capteur de température - Température moyenne et température.

Capteur d'humidité - Humidité relative.

Capteur de batterie - Indicateur de batterie et tension de la batterie.

Capteur de lumière - Lumière 1 et lumière 2.

Graphiques liés aux mesures du réseau

Voisins

Intervalle entre les balises

Le réseau Hops

Au fil du temps

Par nœud

Métrique du routeur (au fil du temps)

Instantané

Moyenne

ETX (au fil du temps)

Le prochain saut (au fil du temps)

Latence

Paquets perdus (au fil du temps)

Paquets reçus

Au fil du temps

Par nœud

Toutes les 5 minutes

Intrigues liées au pouvoir

Puissance moyenne

Puissance instantanée

Histoire du pouvoir

Cycle d'utilisation de la radio

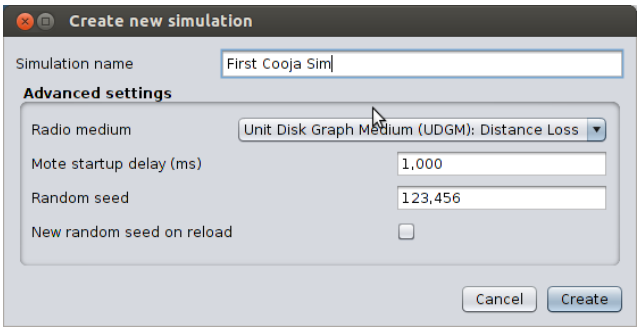
Autres onglets

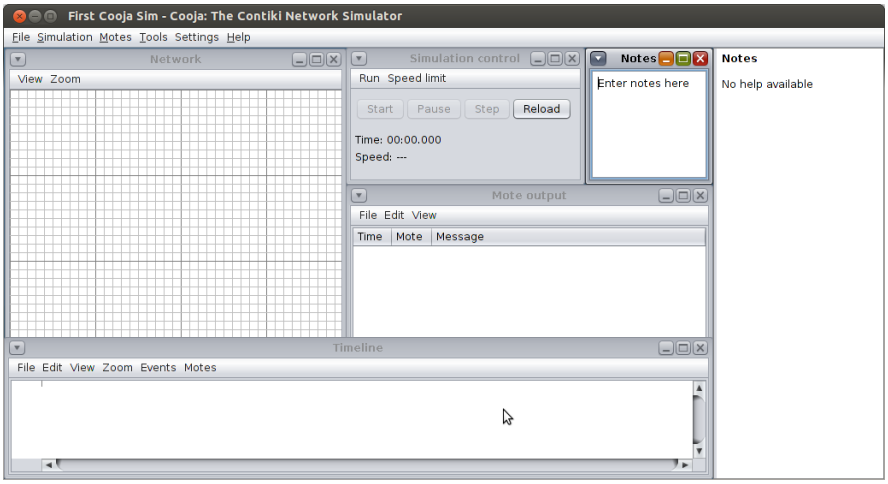
Informations sur les nœuds - Cette section donne un résumé de tous les nœuds et de leurs statistiques.

Console série - L'utilisateur peut interagir directement avec le mote, en envoyant des commandes pour lire et transmettre des données.

Section III :

Pratique :





Réseau - Indique l'emplacement de chaque nœud du réseau. Peut être utilisé pour visualiser l'état de chaque nœud, y compris les LED, les ID de mote, les adresses, les sorties de lof, etc. Au départ, cette fenêtre est vide et nous devons la remplir avec nos capteurs.

Contrôle de la simulation - Ce panneau est utilisé pour démarrer, mettre en pause, recharger ou exécuter les étapes de la simulation. Il indique le temps d'exécution et la vitesse de la simulation. Cela signifie que nous pouvons exécuter les événements plusieurs fois plus vite qu'il ne le faudrait en exécution en temps réel.

Notes - Il s'agit d'un simple bloc-notes permettant de prendre des notes sur la simulation.

Mote output - Affiche toutes les sorties de l'interface série des nœuds. Il est possible d'activer une fenêtre de sortie du Mote pour chaque nœud de la simulation.

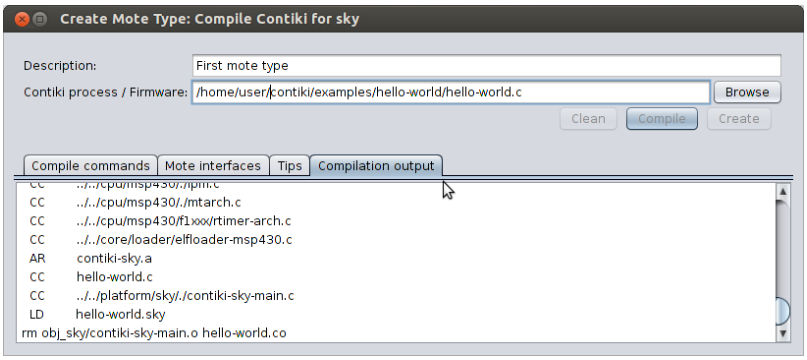
Chronologie - Chronologie de la simulation où sont affichés les messages et les événements tels que le changement de canal, le changement de DEL, les sorties de journal, etc.

En plus des outils par défaut, il est possible d'afficher d'autres outils tels que les points d'arrêt, les messages radio, l'éditeur de script, la vue de la mémoire tampon et le cycle d'utilisation de Mote, qui peuvent être activés dans le menu Outils.

Créer un nouveau type de mote

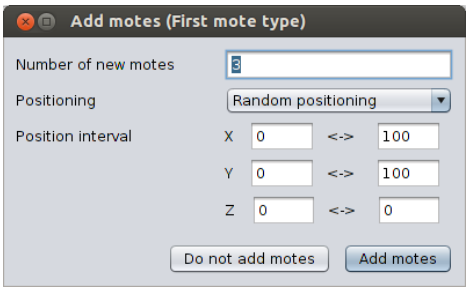
Vous devez créer un nouveau type de mote avant de commencer toute simulation. Vous pouvez le faire dans le menu Motes > Add motes > Create new motes type. Sélectionnez Mote Ciel afin de créer un mote du même type que le mote Ciel utilisé.

La fenêtre qui s'affiche (voir ci-dessous) demande la description du nouveau type de mote et le processus Contiki / Firmware. Vous pouvez nommer votre type de mote comme Premier type de mote et vous pouvez sélectionner le micrologiciel qui sera utilisé pendant la simulation en utilisant le bouton Parcourir. Après avoir sélectionné le microprogramme souhaité, vous pouvez tester la compilation en cliquant sur le bouton Compiler. Dans cet exemple, nous utiliserons le fimware Hello World, qui se trouve généralement dans /contiki/examples/hello-world/hello-world.c. Si le processus de compilation est réussi, vous verrez un message final : LD hello-world.sky dans l'onglet "Compilation output".



Ajouter des mots et exécuter la simulation

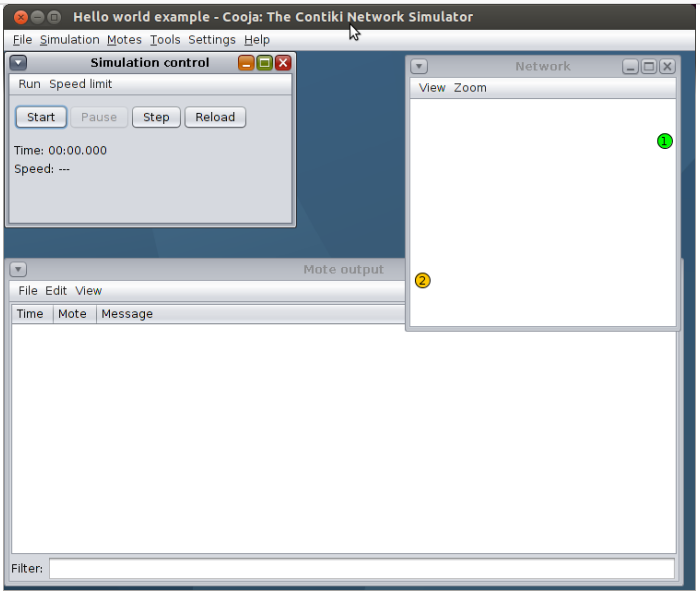
Après avoir compilé avec succès le microprogramme à la dernière étape, vous devez cliquer sur le bouton Créer (voir la figure ci-dessus). Un nouveau type de mote sera créé et vous pourrez ajouter quelques nœuds de ce même type dans votre simulation. Vous verrez la fenêtre ci-dessous, où vous pourrez définir le nombre de nœuds qui seront créés et spécifier leur positionnement. Dans cet exemple, nous allons créer 3 nouveaux nœuds et ils auront un positionnement aléatoire.



Suivi de control s pour sauvegarder extension csc

Pour débugger

```
make TARGET=cooja <simulation_file>.csc
```



Pour démarrer la simulation, cliquez sur "Démarrer" dans la fenêtre "Contrôle de la simulation".

Pendant la simulation, la fenêtre "Mote output" imprime les informations sur l'émetteur et le récepteur.

L'onglet "Affichage" de la fenêtre "Réseau" peut être utilisé pour afficher les caractéristiques du mote, telles que

- Mote IDs- Affiche l'identifiant de la mote

- Adresses : IP ou Rime- Affiche l'adresse IPv6 du mote

- Trafic radio - Permet une animation montrant la communication entre les motos

- Positions- Affiche les coordonnées du mote

- Environnement radio - En cliquant sur un mote particulier, vous pouvez afficher sa zone de couverture

```
PROCESS(name,strname)
```

Chaque processus doit être défini via la macro PROCESS. PROCESS a deux arguments : la variable de la structure du processus, et un nom de chaîne lisible par l'homme, qui est utilisé lors du débogage.

name : Le nom de la variable de la structure du processus.

strname : La représentation sous forme de chaîne de caractères du nom du processus.

```
AUTOSTART_PROCESS(struct process &)
```

AUTOSTART_PROCESSES lance automatiquement le(s) processus indiqué(s) dans le(s) argument(s) au démarrage du module.

&name : Référence au nom du processus

```
broadcast_recv(struct broadcast_conn *, const rimeaddr_t *)
```

Cette fonction analyse un paquet entrant et affiche le message et l'adresse de l'expéditeur. En la définissant comme la fonction de rappel désignée de la diffusion, broadcast_recv est automatiquement appelée lorsqu'un paquet est reçu.

broadcast_conn * : Cette structure qui a 2 structures : abc_conn, broadcast_callbacks *. L'abc_conn est le type de connexion de base sur lequel la connexion de diffusion est développée. Et, les broadcast_callbacks pointent vers les fonctions recv et sent (dans cet exemple, juste recv).

rimeaddr_t * : Il s'agit d'une union qui possède un tableau de caractères u8 [RIMEADDR_SIZE].

```
PROCESS_THREAD(name, process_event_t, process_data_t)
```

Dans Contiki, un processus consiste en une seule référence à un "fil de prototouche". Cette fonction est utilisée pour définir le fil d'essai d'un processus. Le processus est appelé chaque fois qu'un événement se produit dans le système. Chaque processus du module nécessite un gestionnaire d'événements sous la macro PROCESS_THREAD.

name : Le nom de la variable de la structure du processus.

process_event_t : Si cette variable est identique à PROCESS_EVENT_EXIT, alors PROCESS_EXITHANDLER est invoqué.

Dans le corps de PROCESS_THREAD, il y a 3 tâches principales :

Initialisation

allouer des ressources

définir les variables

commencer le processus

Boucle infinie

while(1) est utilisé pour créer une boucle infinie dans laquelle la réponse réelle de traitement de l'événement a lieu

Deallocation

processus final

désaffecter les ressources

```
PROCESS_EXITHANDLER(handler)
```

Spécifie une action lorsqu'un processus se termine. NOTE : Cette déclaration doit précéder immédiatement la macro PROCESS_BEGIN().

handler : L'action à exécuter.

```
PROCESS_BEGIN()
```

Cette macro définit le début d'un processus et doit toujours apparaître dans une définition de PROCESS_THREAD() Cette macro lance PT_BEGIN(), qui est déclaré au point de départ d'un protothread. Toutes les déclarations C au-dessus de l'invocation de PT_BEGIN() seront exécutées à chaque fois que le protothread est programmé.

```
broadcast_close(struct broadcast_conn *)
```

Cette fonction ferme une connexion de diffusion qui a été précédemment ouverte avec broadcast_open() Cette fonction est généralement appelée gestionnaire de sortie.

broadcast_conn : C'est la même chose que la variable de broadcast_recv().

```
PROCESS_END()
```

Cette macro définit la fin d'un processus. Elle doit apparaître dans une définition de PROCESS_THREAD() et doit toujours être incluse. Le processus se termine lorsque la macro

PROCESS_END() est atteinte. Cette macro lance PT_END() et doit toujours être utilisée avec une macro PT_BEGIN() correspondante.

```
broadcast_open(struct broadcast_conn *, uint16_t ,const struct
broadcast_callbacks *)
```

Établit une connexion de diffusion identifiée comme la meilleure solution. L'appelant alloue la mémoire pour la structure broadcast_conn, généralement en la déclarant comme une variable statique. Le pointeur de la struct broadcast_callbacks pointe sur une structure contenant un pointeur vers une fonction qui sera appelée lorsqu'un paquet arrivera sur le canal. Cette fonction ouvre une connexion de type abc_conn et fixe les callbacks à la structure passée. Elle pointe également vers la fonction channel_set_attributes().

broadcast_conn : Un pointeur vers une structure broadcast_conn

uint16_t : Le canal sur lequel la connexion fonctionnera

broadcast_callbacks : Une structure broadcast_callbacks avec des pointeurs de fonctions vers des fonctions qui seront appelées lorsqu'un paquet aura été reçu

```
etimer_set(struct etimer *, clock_time_t)
```

Cette fonction est utilisée pour régler un minuteur d'événement pour une période future. Lorsque le minuteur d'événement expire, l'événement PROCESS_EVENT_TIMER sera affiché dans le processus qui a appelé la fonction etimer_set().

etimer : Un pointeur vers le minuteur d'événements

clock_time_t : L'intervalle avant l'expiration de la minuterie.

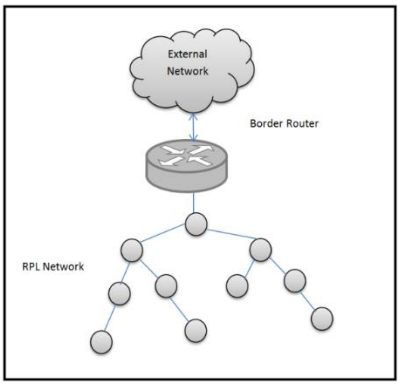
```
static struct broadcast_conn
```

The broadcast module sends a packet to all local area neighbors with a header that identifies the sender. It also adds a single-hop address as a packet attribute to out going packets. broadcast_conn structure consists of two structures

- abc_conn struct: the abc(Anonymous best effort local area Broadcast) module sends packets to all local area neighbors. It uses one channel.

broadcast_callbacks struct: this is called when a packet has been received by the broadcast module. The struct broadcast_callbacks pointer is used in broadcast_open to point to a function that will be called when a packet arrives on the channel.

Les routeurs de commandes sont des routeurs que l'on peut trouver à la périphérie d'un réseau. Leur fonction est de connecter un réseau à un autre. Dans ce tutoriel, nous apprendrons à effectuer une simulation en utilisant le routeur de frontière dans Contiki 2.7. Dans l'exemple dont nous parlerons dans la suite de ce tutoriel, le routeur frontalier sera utilisé pour acheminer des données entre un réseau WSN (réseau RPL) et un réseau IP externe.



Objectif

L'objectif de ce tutoriel est de vous donner une compréhension du code RPL Border Router dans l'OS Contiki 2.7 et d'apprendre comment simuler un réseau avec un routeur frontalier sur Cooja

Bloc de code

1. Comprendre le fonctionnement du code du routeur frontalier rpl

2. Démarrage d'une simulation sur le simulateur Cooja.

3. Observer et valider les résultats

Les blocs de construction du code.

Nous utiliserons les fichiers suivants

1. frontière-routage.c

2. udp-server.c (udp-client.c peut également être utilisé)

3. slip-bridge.c (Il contient une fonction de rappel pour le traitement d'une demande de connexion SLIP)

4. httpd-simple.c (un simple serveur web qui transmet la génération de pages à un protothread)

Les nœuds du serveur udp formeront un DAG avec le routeur de frontière défini comme racine. Le routeur frontalier recevra le préfixe par le biais d'une connexion [SLIP][1] (Serial Line Interface Protocol) et il sera communiqué au reste des nœuds du réseau RPL.

Reportez-vous aux extraits de code suivants dans le fichier border-router.c Dans cet extrait de code, le nœud configuré comme routeur de frontière attend que le préfixe soit défini. Une fois qu'il a reçu le préfixe, le routeur de frontière est défini comme la racine du DAG, après quoi il définit le préfixe des autres nœuds du réseau.

```
while(!prefix_set) {
   etimer_set(&et, CLOCK_SECOND);
   request_prefix();
   PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
 }


 dag = rpl_set_root(RPL_DEFAULT_INSTANCE,(uip_ip6addr_t *)dag_id);
 if(dag != NULL) {
   rpl_set_prefix(dag, &prefix, 64);
   PRINTF("nv RPl créé dag \n");
 }
```

Par défaut, le routeur frontalier héberge une simple page web. Toutefois, il est possible de la désactiver en définissant WEBSERVER comme indiqué dans le code ci-dessous. Cette page web est affichée lorsque l'adresse IPv6 du routeur frontalier est saisie dans le navigateur. Reportez-vous au fichier httpd-simple.c pour obtenir le code suivant

```
PROCESS(border_router_process, "Border router process");
#if WEBSERVER==0
/* No webserver */
AUTOSTART_PROCESSES(&border_router_process);
#elif WEBSERVER>1
/* Use an external webserver application */
#include "webserver-nogui.h"
AUTOSTART_PROCESSES(&border_router_process,&webserver_nogui_process);
```

Compilation du code

Le code du routeur frontalier RPL se trouve à l'adresse /contiki-2.7/examples/ipv6/rpl-border-router. Utilisez la commande suivante pour compiler le code du routeur frontalier

```
cd /contiki-2.7/examples/ipv6/rpl-border-router
make TARGET=sky.
```

Une fois que cette commande a été exécutée avec succès, un fichier nommé border-router.sky sera créé. Ce fichier sera utilisé pour programmer l'un des moteurs comme routeur frontalier sur Cooja. Pour les besoins de ce tutoriel, nous sélectionnerons TMote Sky comme cible.

Afin de démontrer la fonctionnalité du routeur frontalier, nous allons créer un réseau de nœuds avec le routeur frontalier comme racine. Pour créer un tel réseau, nous utiliserons le code udp-server.c. Ce code peut être trouvé à l'adresse /contiki-2.7/examples/ipv6/rpl-udp . Utilisez la commande suivante pour compiler le code pour rpl-udp

```
cd /contiki-2.7/examples/ipv6/rpl-udp
make TARGET=sky
```

Une fois que cette commande a été exécutée avec succès, un fichier nommé udp-server.sky sera créé. Ce fichier sera utilisé pour programmer les nœuds restants dans le simulateur Cooja. Ces nœuds formeront un DAG avec le routeur de frontière rpl comme racine.

Test sur Cooja

Une fois ces étapes franchies avec succès, nous sommes prêts à créer une simulation à Cooja. Démarrez le simulateur de Cooja en utilisant la commande suivante

```
cd /contiki-2.7/tools/cooja
ant run
```

Lorsque l'interface graphique est lancée, exécutez les étapes suivantes pour créer une simulation.

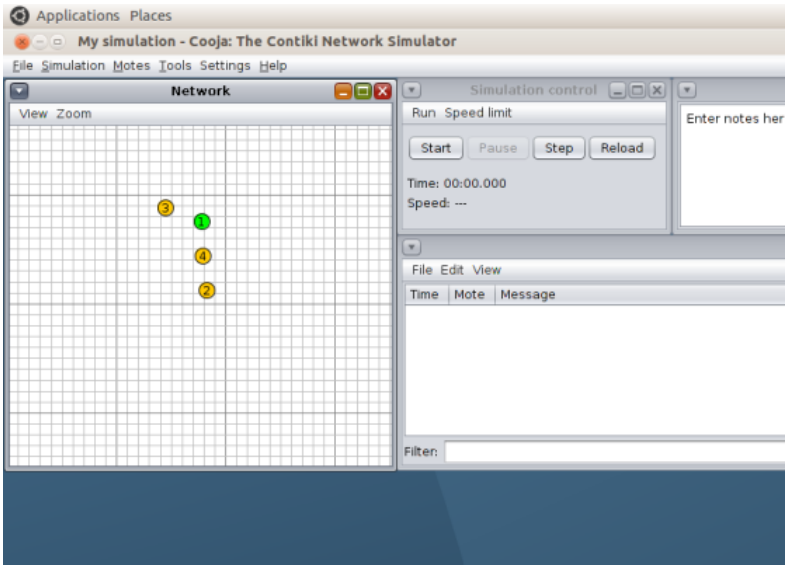
1. Dans "Fichier", sélectionnez "Nouvelle simulation". Sélectionnez "UDGM", entrez le nom de la simulation et cliquez sur "créer".

2. Cliquez sur "Motes". Dans le menu déroulant, sélectionnez "Add New Motes", puis "Create new motes" et sélectionnez le type de mote "Sky".

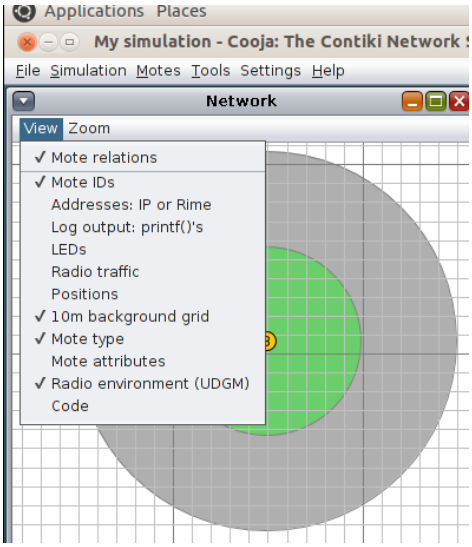
3. Naviguez jusqu'à l'emplacement "/contiki-2.7/examples/ipv6/rpl-border-router" et sélectionnez le fichier rpl-border-router.sky. Cliquez sur "Créer". Ajoutez un mote de ce type.

4. Répétez les étapes 2 et 3, mais cette fois, allez à l'emplacement "/contiki-2.7/examples/ipv6/rpl-udp" et sélectionnez le fichier udp-server.sky. Ajoutez 3 - 4 mots du type udp-server

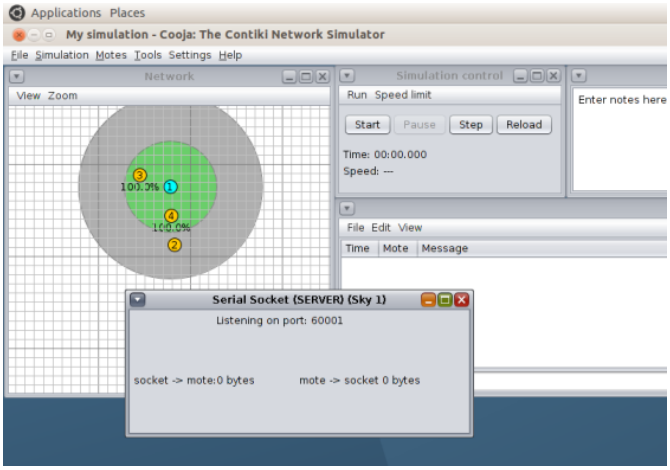
5. Une fois que les mottes ont été ajoutées, vous pouvez les positionner.

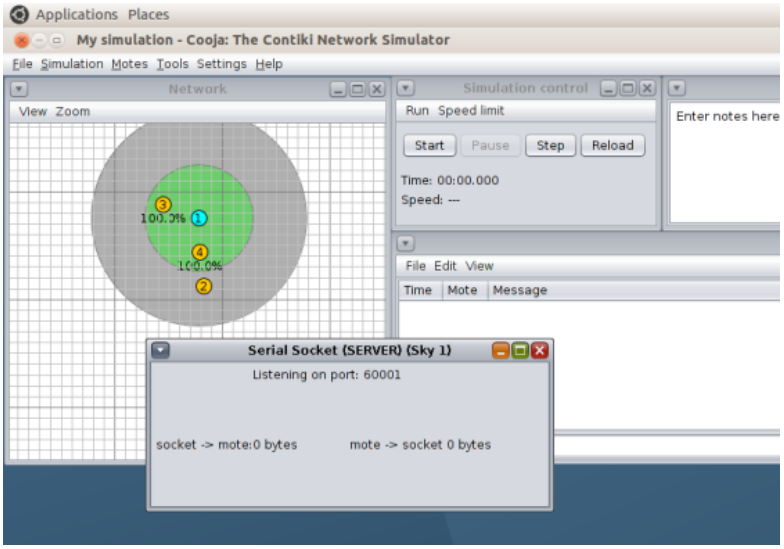


Sélectionnez les options de la rubrique Affichage comme indiqué ci-dessous. Celles-ci vous aideront à créer une topologie de votre choix.



6. Maintenant, nous devons créer un pont entre le réseau RPL simulé sur Cooja et la machine locale. Cela peut être fait en cliquant avec le bouton droit de la souris sur le mote qui est programmé comme routeur de frontière. Sélectionnez "Autres outils pour le routeur frontalier", puis "Prise série (SERVER)". Vous obtiendrez le message suivant lorsque cette étape sera terminée avec succès : "Écoute sur le port 60001".





7. Ne reste plus qu’à lancer

Utilitaire Tunslip

Comme mentionné dans l'introduction, un routeur frontalier permet de relier un réseau à un autre. Dans cet exemple, le routeur frontalier est utilisé pour acheminer des données entre un réseau RPL et un réseau externe. Jusqu'à présent, nous n'avons créé que le réseau RPL. Nous devons maintenant simuler le scénario dans lequel ce réseau RPL est connecté à un réseau externe. Pour ce faire, nous utiliserons l'utilitaire Tunslip fourni dans Contiki. Dans cet exemple, tunslip crée un pont entre le réseau RPL et la machine locale. tunslip6.c peut être trouvé dans /contiki-2.7/tools Compiler le code de tunslip6.

```
make tunslip6
```

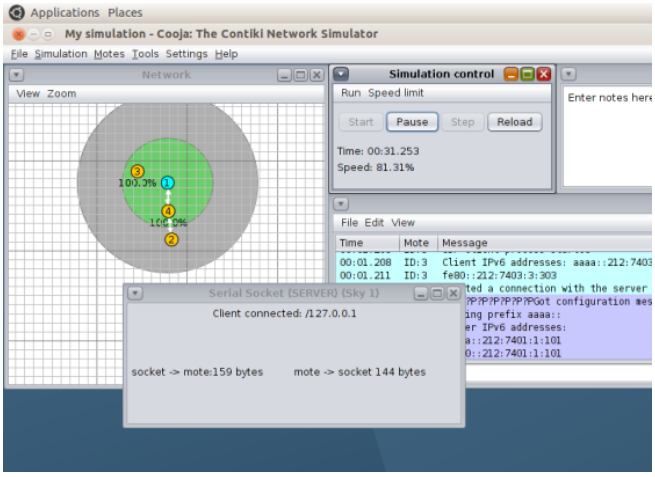
Établissez une connexion entre le réseau RPL et votre machine locale.

```
slip connected to ``127.0.0.1:60001''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` up
ifconfig tun0 add aaaa::1/64
ifconfig tun0 add fe80::0:0:0:1/64
ifconfig tun0
```

```
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-
00-00-00-00-00
          inet addr:127.0.1.1  P-t-P:127.0.1.1  Mask:255.255.255.255
          inet6 addr: fe80::1/64 Scope:Link
          inet6 addr: aaaa::1/64 Scope:Global
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:500
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

```
Rime started with address 0.18.116.1.0.1.1.1
MAC 00:12:74:01:00:01:01:01 Contiki 2.7 started. Node id is set to 1.
CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
Tentative link-local IPv6 address
fe80:0000:0000:0000:0212:7401:0001:0101
Starting 'Border router process' 'Web server'
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::212:7401:1:101
 fe80::212:7401:1:101
```

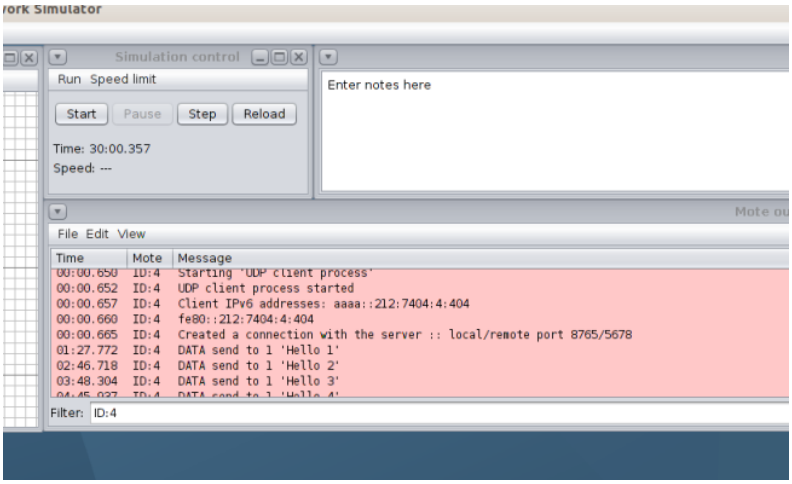
On retorune sur la gui de cooja on voit client conecté 127.0.0.1



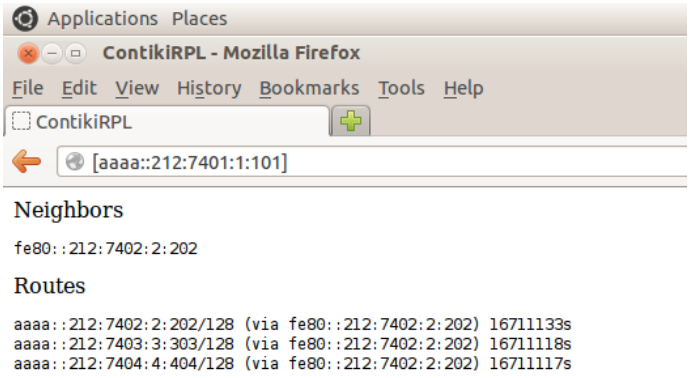
Pour vérfier es résulatts :

```
user@instant-contiki:~/contiki-2.7$ ping6 aaaa::212:7401:1:101
```

si le ping est reçu c'est que c'est bon le premier ping peutetre perdu car c'est une phase de shakehand.



Ou en entrant lip sur un navigateur :



Problèmes courant :

1. Assurez-vous que vous entrez correctement l'adresse IPv6 dans la commande sudo ./tunslip6 -a 127.0.0.1 aaaa::1/64 . Il ne doit y avoir aucun espace dans aaaa::1/64 . Sinon, le code donnera un comportement inattendu.

2. Lorsque vous mettez la simulation en pause, la connexion SLIP est perdue. Il se peut qu'elle ne fonctionne pas correctement lors de la reprise de la simulation. Dans ce cas, rechargez la simulation. Créez une nouvelle connexion SLIP, puis relancez la simulation.

Tout da'rd on télacharge la obilité

http://sourceforge.net/p/contikiprojects/code/HEAD/tree/sics.se/mobility/

Create a new directory at

cd contiki/tools/cooja/apps

mkdir mobility

on construit le plugin :

cd contiki/tools/cooja/apps/mobility

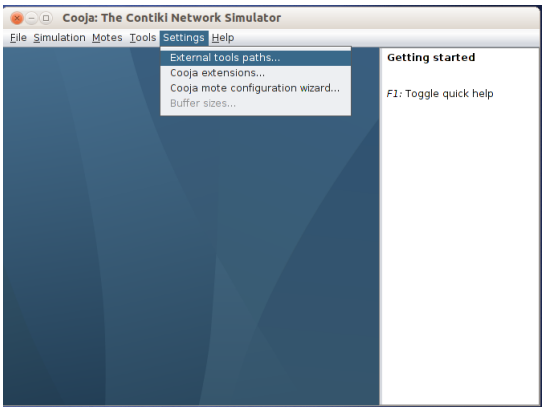
sudo ant jar

On la'ctive :

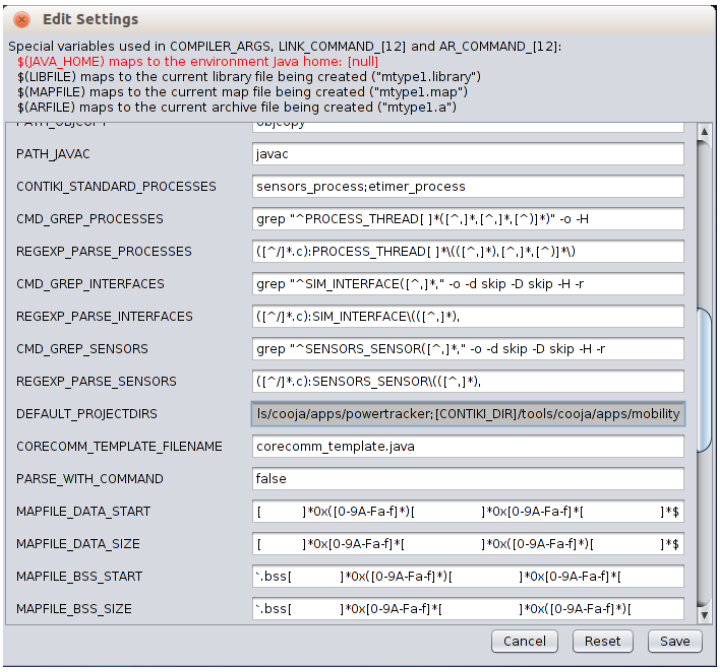
cd contiki/tools/cooja

sudo ant run

Settings > External Tools Path…



On a :



Faites défiler l'écran jusqu'à DEFAULT_PROJECTDIRS

Ce champ est utilisé pour spécifier les chemins d'accès à tous les plugins du Simulateur de Cooja

Ici, vous devez ajouter un chemin vers le plugin téléchargé. Nous faisons cela pour lier le plugin avec Cooja Simulator

Maintenant, nous devons ajouter le chemin du plugin aux chemins existants du plugin

Allez à la fin de ce champ DEFAULT_PROJECTDIRS

Insérer le symbole " ; ".

Ajouter le chemin du plugin

[CONTIKI_DIR]/outils/cooja/apps/mobilité

Cliquez sur "Sauvegarder

Fermez le simulateur de Cooja et recommencez.

cd contiki/tools/cooja/

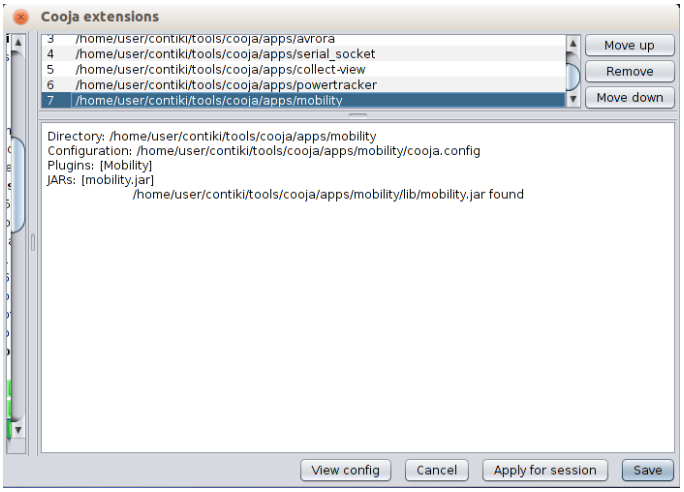
sudo ant run

Maintenant, allez à

Paramètres>Extensions de Cooja

La fenêtre des extensions de Cooja s'ouvre.

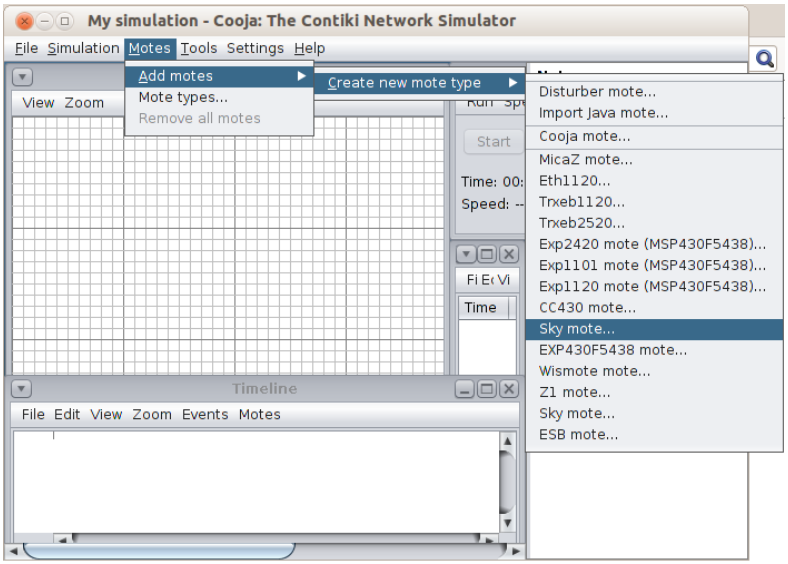
Faites défiler vers le bas jusqu'à mobilité, sélectionnez-la et cliquez sur "Appliquer pour la session".



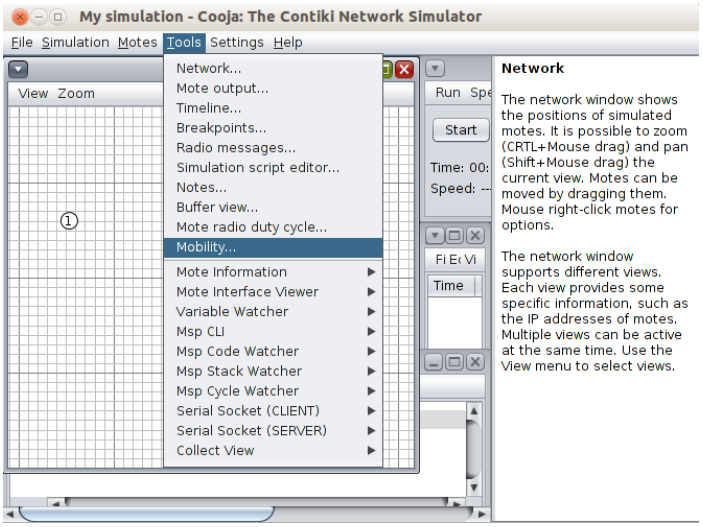
Sous l'onglet Outils dans Cooja, vous devriez voir une nouvelle option "Mobilité..." maintenant

Maintenant pour l'essayer :

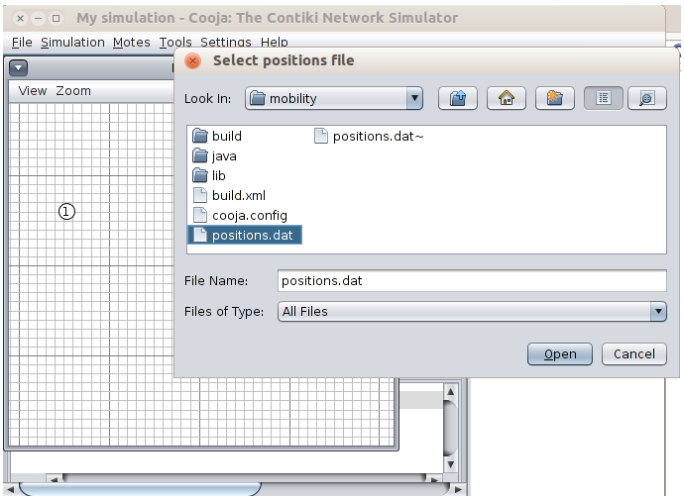
Ouveau fichier -> simulation :



Javac hhelo-word.c



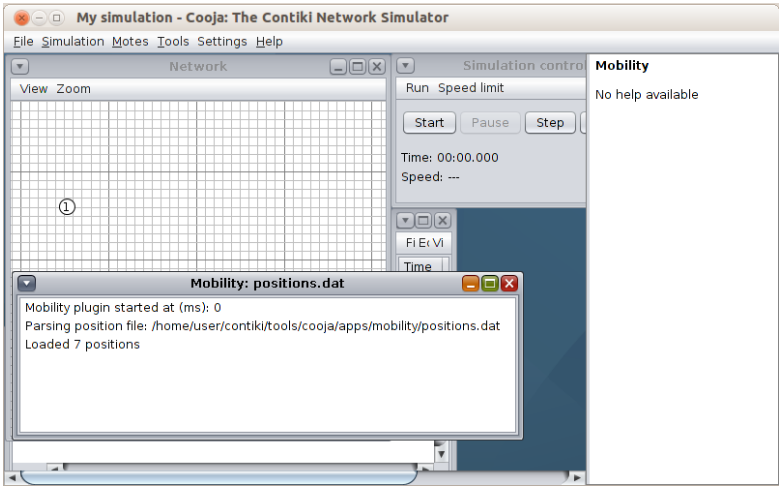
Puis :



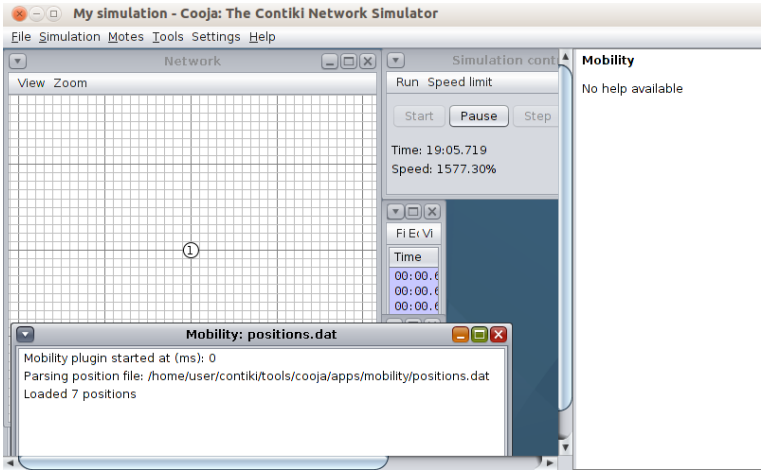
Nous avons demandé à notre plugin de mobilité d'obtenir les informations de position à partir de ce fichier positions.dat

Cliquez sur "Ouvrir

Une petite fenêtre s'ouvrira comme indiqué ci-dessous



Maintenant, cliquez sur "Démarrer la simulation".



Voilà ! Votre mote commencera à se déplacer selon les positions spécifiées Fichier .dat

Positions.dat Structure du fichier

Le fichier Position.dat contient ces informations

#node time(s) x y

0 0.0 0 10

0 1.0 10 10

0 2.0 10 0

0 3.0 0 0

0 4.0 0 10

La première colonne précise le numéro de nœud

La deuxième colonne est l'horodatage

La troisième colonne est la coordonnée x

La quatrième colonne est la coordonnée y

#node time(s) x y

0 0.0 0 10

Cette ligne signifie que le nœud 0 (mote 1) à 0,0sec sera aux coordonnées de position (0, 10)

Note : Ici, le nœud 0 est le "mote 1" de votre simulateur de cooja

Si vous indiquez le nœud 1 dans la première colonne, il s'agit du "mote 2".

Ainsi, le nœud positions.dat (n) est en fait un mote (n+1) sur le simulateur Cooja

Changement de positions Fichier .dat ou fichiers de plugin

Changement de position fichier .dat

Si vous souhaitez modifier le fichier position.dat. Remplacez le fichier position.dat par le nouveau fichier

Sous Outils> Mobilité

Ajoutez le nombre de nœuds requis dans le simulateur et démarrez votre simulation.

Les nouvelles positions seront chargées maintenant et les nœuds commenceront à se déplacer selon les nouvelles spécifications de position.

Changement des fichiers du plugin de base

Si vous souhaitez modifier les fichiers du plugin de base en fonction de vos besoins.

Après avoir mis en œuvre vos changements et modifié les fichiers. Vous devez exécuter

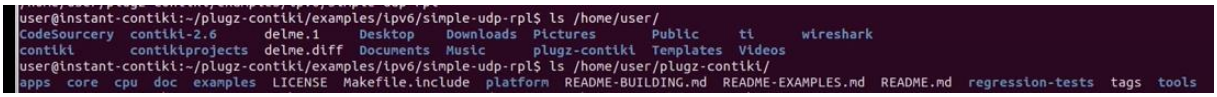
sudo ant clean

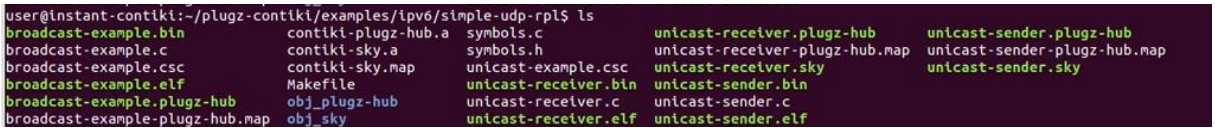
pot sudo ant

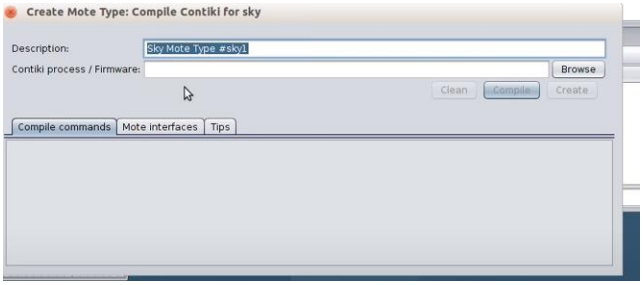
A l'intérieur de la

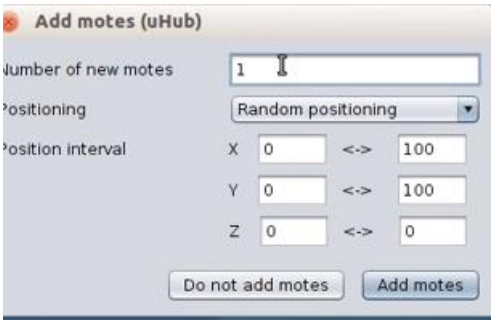
dossier contiki/outils/cooja/apps/mobility

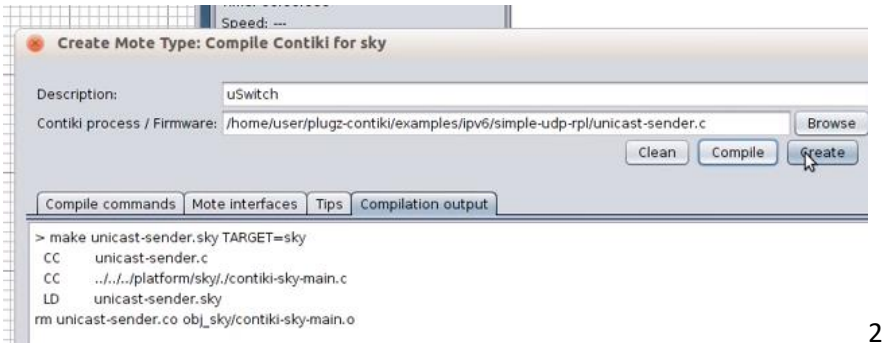
Coitki :



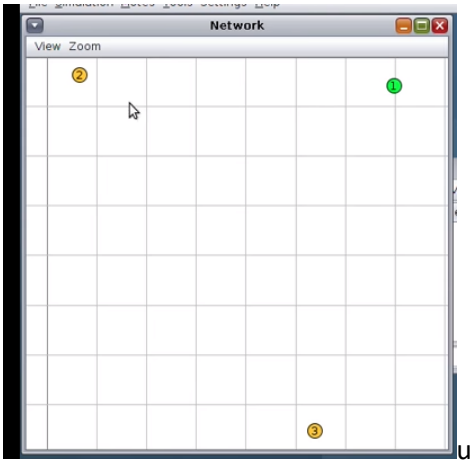






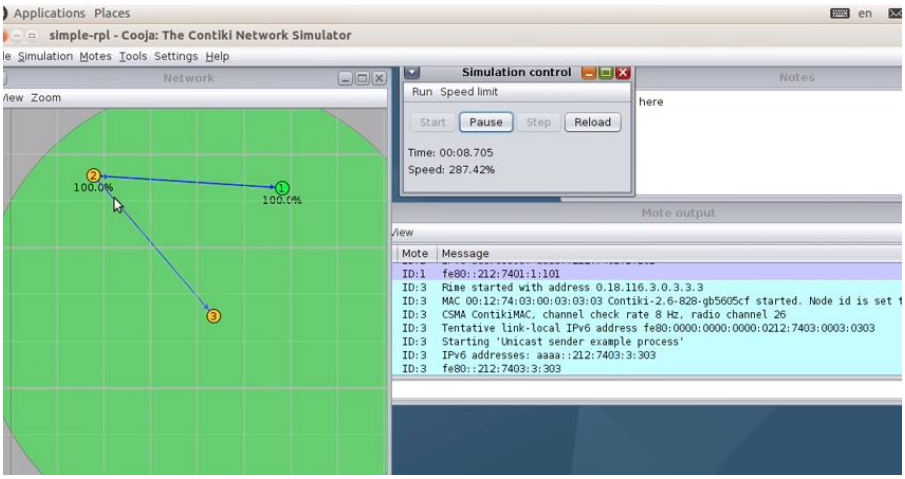


2

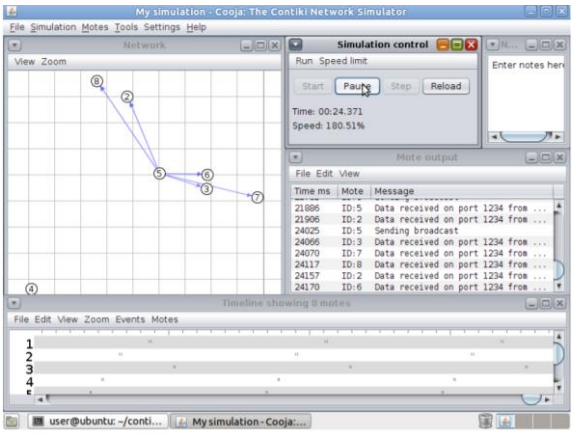


u switch orange uhub vert

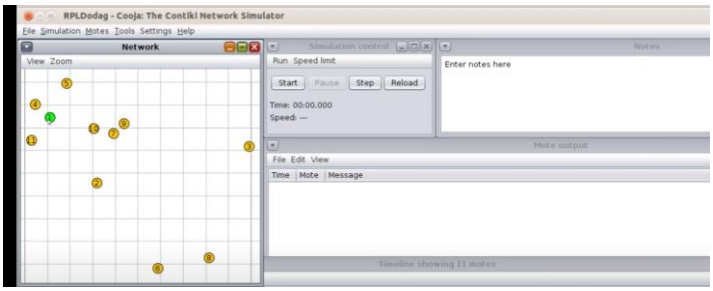
On peut voir le trafic radio



3 passe par 4 pour communiquer à 1 et à 2

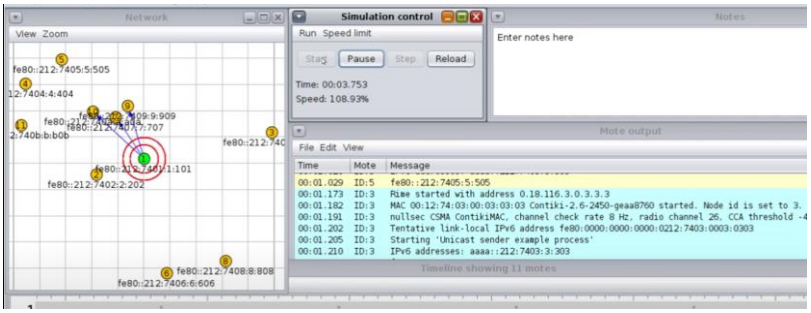


Simulation avec plusieurs / 11 points :

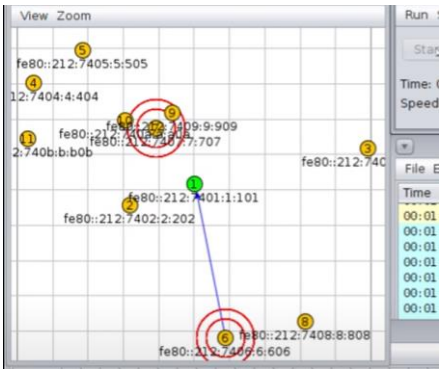


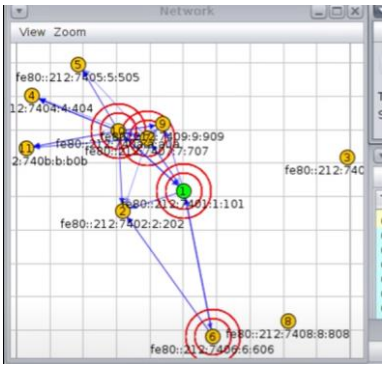
Portée :

Lorsque la simulation est lancée :

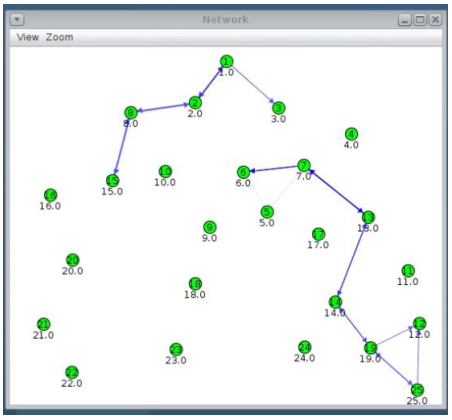


Pour aller vers le point 7 :

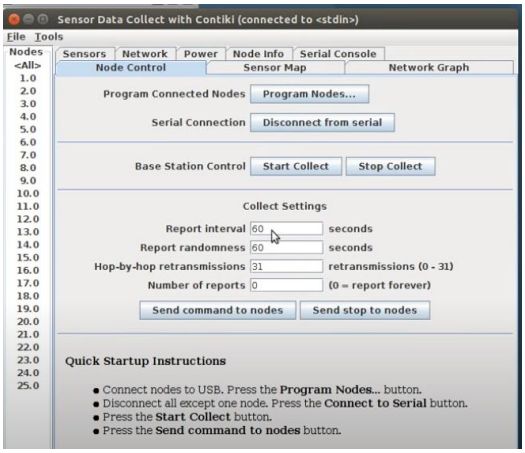




Avec switch motes/sky :

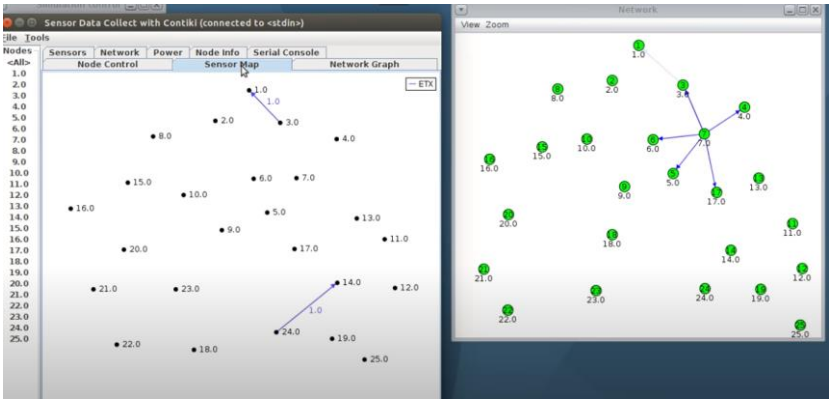


Node control :

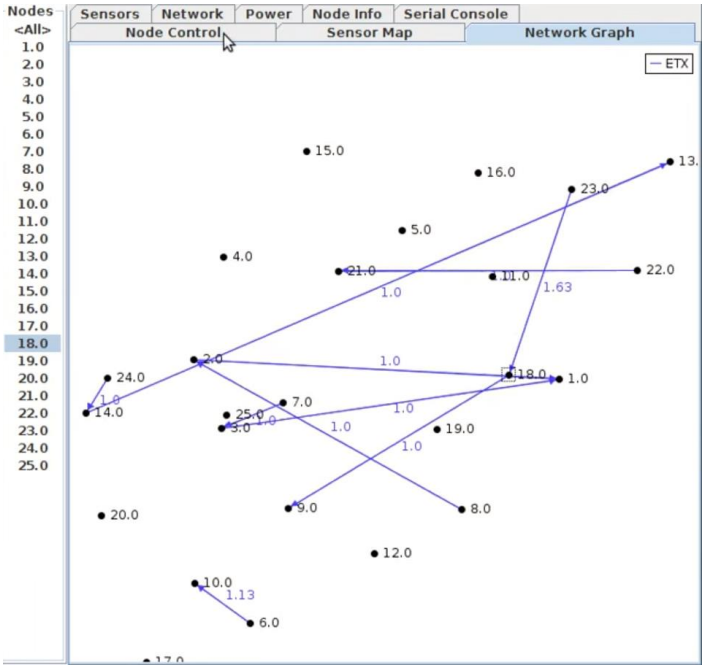


Send command to roles -> start collect

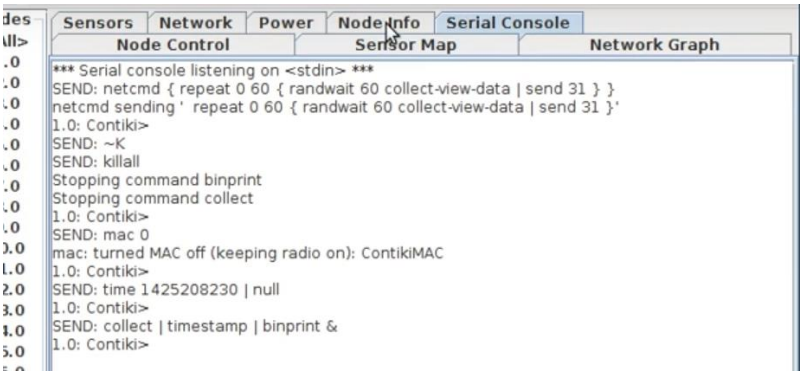
Voici le sensor map :



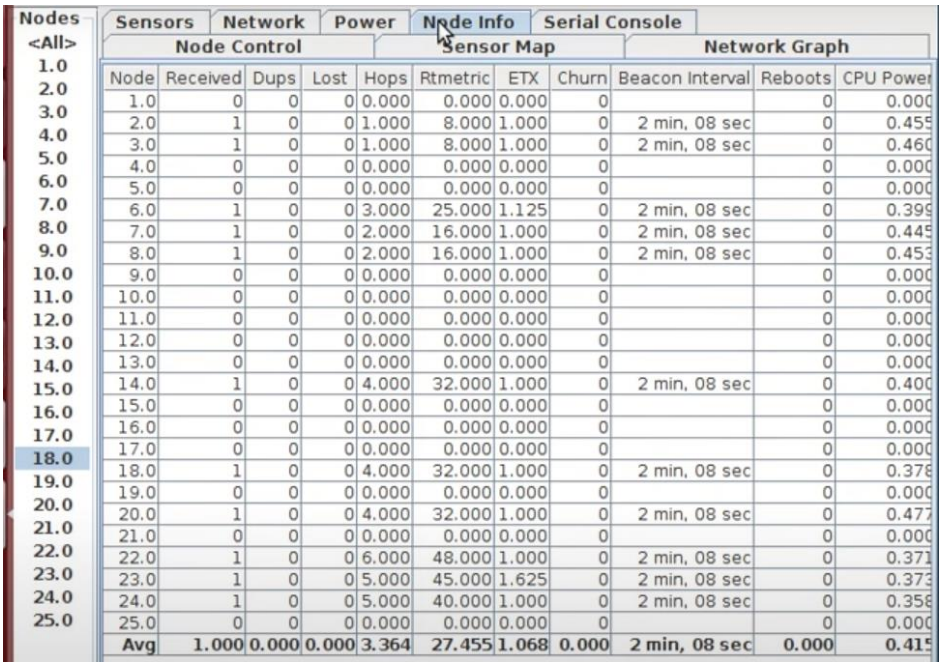
Voici le network graph :



Voici la console série comme sur Arduino :



Voici le node info :

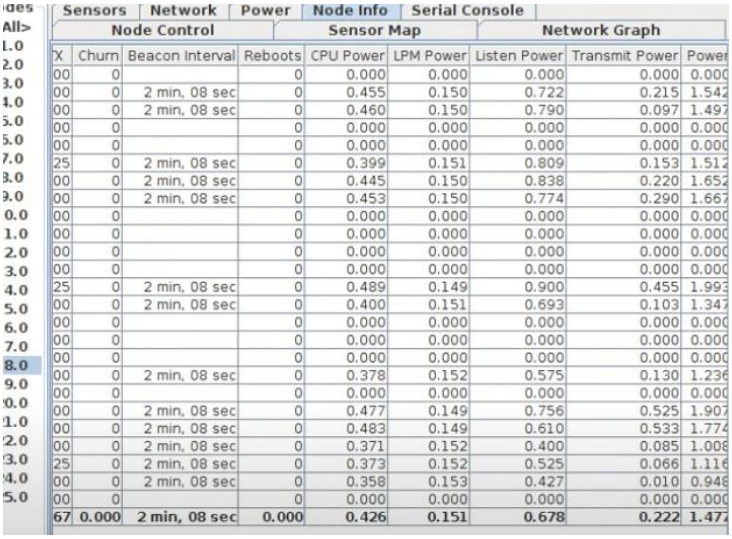


Nodes | Sensors | Network | Power | Node Info | Serial Console | Node Control | Sensor Map | Network Graph

<All>
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0

Node	Received	Dups	Lost	Hops	Rtmetric	ETX	Churn	Beacon Interval	Reboots	CPU Power
1.0	0	0	0	0.000	0.000	0.000	0		0	0.000
2.0	1	0	0	1.000	8.000	1.000	0	2 min, 08 sec	0	0.455
3.0	1	0	0	1.000	8.000	1.000	0	2 min, 08 sec	0	0.460
4.0	0	0	0	0.000	0.000	0.000	0		0	0.000
5.0	0	0	0	0.000	0.000	0.000	0		0	0.000
6.0	1	0	0	3.000	25.000	1.125	0	2 min, 08 sec	0	0.399
7.0	1	0	0	2.000	16.000	1.000	0	2 min, 08 sec	0	0.445
8.0	1	0	0	2.000	16.000	1.000	0	2 min, 08 sec	0	0.453
9.0	0	0	0	0.000	0.000	0.000	0		0	0.000
10.0	0	0	0	0.000	0.000	0.000	0		0	0.000
11.0	0	0	0	0.000	0.000	0.000	0		0	0.000
12.0	0	0	0	0.000	0.000	0.000	0		0	0.000
13.0	0	0	0	0.000	0.000	0.000	0		0	0.000
14.0	1	0	0	4.000	32.000	1.000	0	2 min, 08 sec	0	0.400
15.0	0	0	0	0.000	0.000	0.000	0		0	0.000
16.0	0	0	0	0.000	0.000	0.000	0		0	0.000
17.0	0	0	0	0.000	0.000	0.000	0		0	0.000
18.0	1	0	0	4.000	32.000	1.000	0	2 min, 08 sec	0	0.378
19.0	0	0	0	0.000	0.000	0.000	0		0	0.000
20.0	1	0	0	4.000	32.000	1.000	0	2 min, 08 sec	0	0.477
21.0	0	0	0	0.000	0.000	0.000	0		0	0.000
22.0	1	0	0	6.000	48.000	1.000	0	2 min, 08 sec	0	0.371
23.0	1	0	0	5.000	45.000	1.625	0	2 min, 08 sec	0	0.373
24.0	1	0	0	5.000	40.000	1.000	0	2 min, 08 sec	0	0.358
25.0	0	0	0	0.000	0.000	0.000	0		0	0.000
Avg	**1.000**	**0.000**	**0.000**	**3.364**	**27.455**	**1.068**	**0.000**	**2 min, 08 sec**	**0.000**	**0.415**

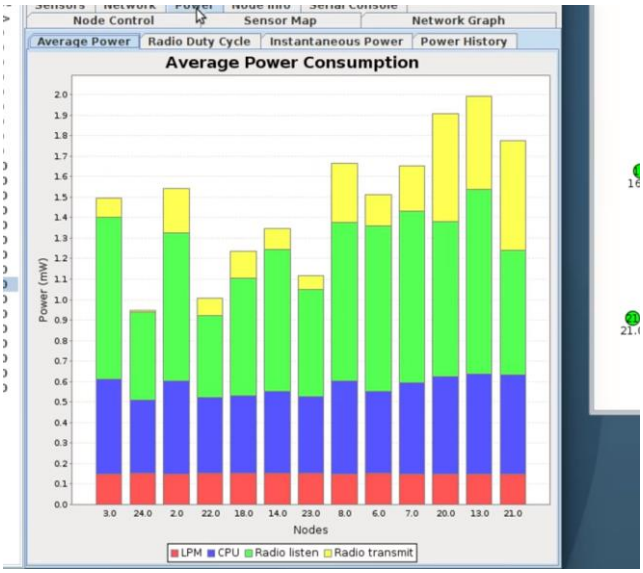
On voit les temps des beacon, le numéro du node, le nombre de paquets recçu, perdus, le nombre de saut, la puissance du CPU,



Sensors | Network | Power | Node Info | Serial Console | Node Control | Sensor Map | Network Graph

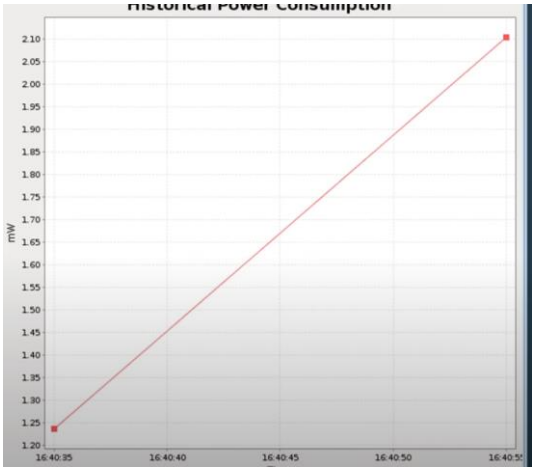
X	Churn	Beacon Interval	Reboots	CPU Power	LPM Power	Listen Power	Transmit Power	Power
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0	2 min, 08 sec	0	0.455	0.150	0.722	0.215	1.542
00	0	2 min, 08 sec	0	0.460	0.150	0.790	0.097	1.497
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0		0	0.000	0.000	0.000	0.000	0.000
25	0	2 min, 08 sec	0	0.399	0.151	0.809	0.153	1.512
00	0	2 min, 08 sec	0	0.445	0.150	0.838	0.220	1.652
00	0	2 min, 08 sec	0	0.453	0.150	0.774	0.290	1.667
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0		0	0.000	0.000	0.000	0.000	0.000
25	0	2 min, 08 sec	0	0.489	0.149	0.900	0.455	1.993
00	0	2 min, 08 sec	0	0.400	0.151	0.693	0.103	1.347
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0	2 min, 08 sec	0	0.378	0.152	0.575	0.130	1.236
00	0		0	0.000	0.000	0.000	0.000	0.000
00	0	2 min, 08 sec	0	0.477	0.149	0.756	0.525	1.907
00	0	2 min, 08 sec	0	0.483	0.149	0.610	0.533	1.774
00	0	2 min, 08 sec	0	0.371	0.152	0.400	0.085	1.008
25	0	2 min, 08 sec	0	0.373	0.152	0.525	0.066	1.116
00	0	2 min, 08 sec	0	0.358	0.153	0.427	0.010	0.948
00	0		0	0.000	0.000	0.000	0.000	0.000
67	**0.000**	**2 min, 08 sec**	**0.000**	**0.426**	**0.151**	**0.678**	**0.222**	**1.477**

L’average power indique les ifos d’alimentation :

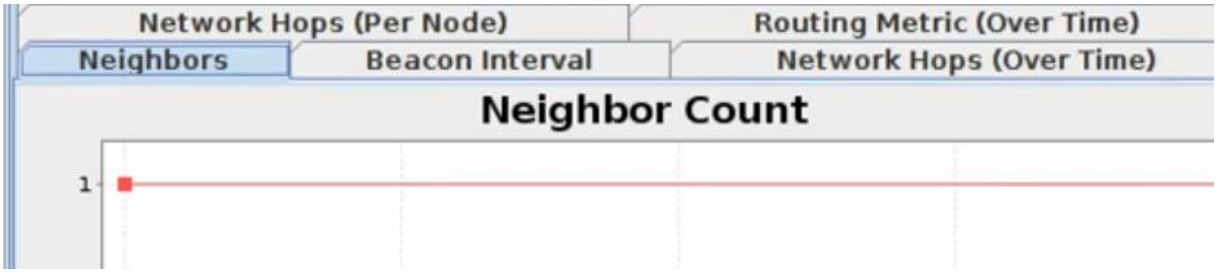


On trouve aussi ceux-là :

Comme celui-là :

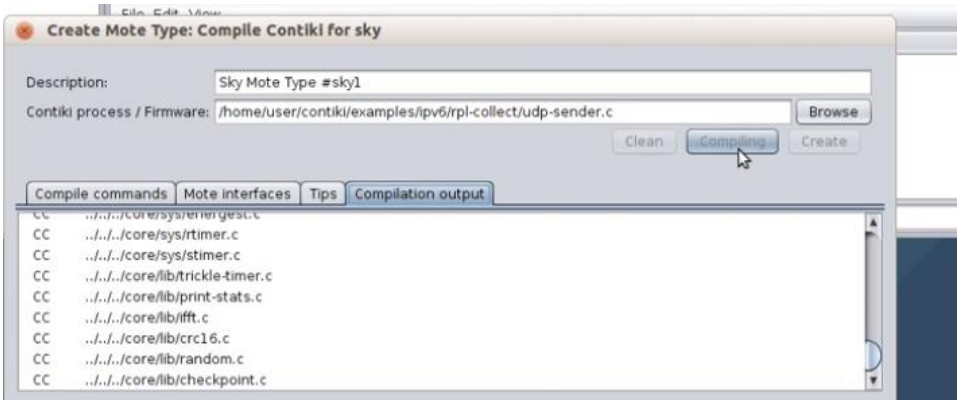


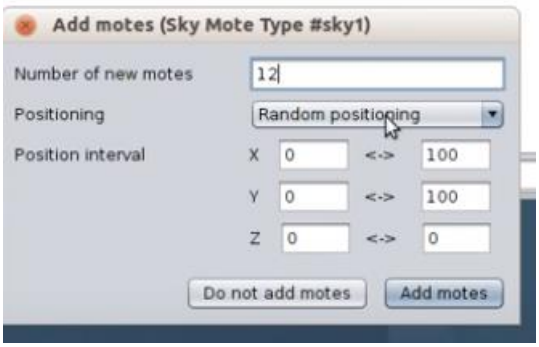
Le neighbor count :



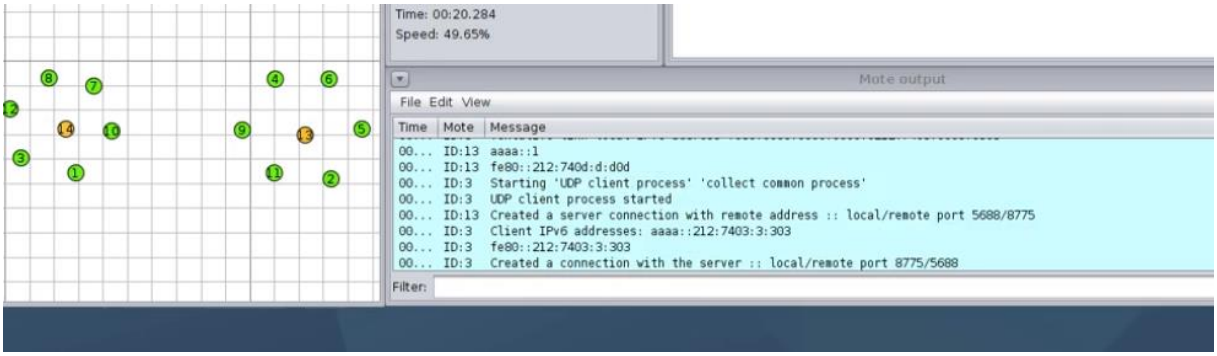
En général on utilise du udp

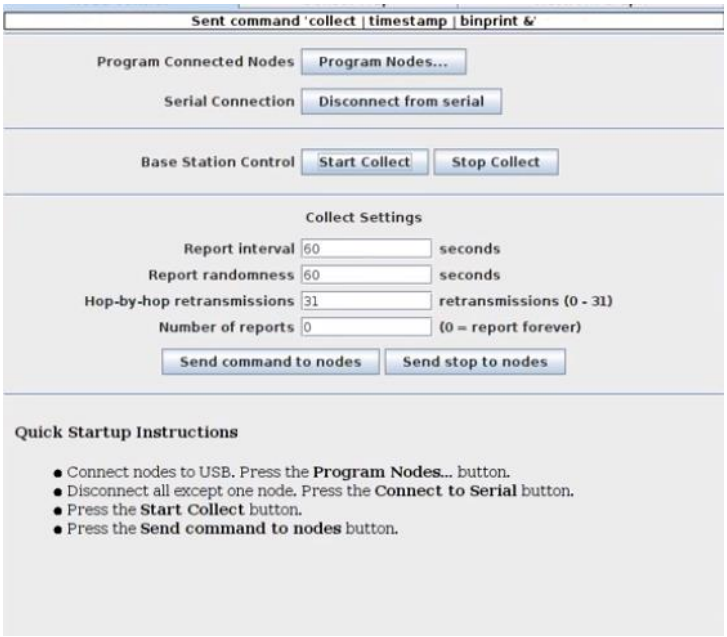
On peut compiler directemnt depuis cooja :





On peut simuler des réseaux différents :



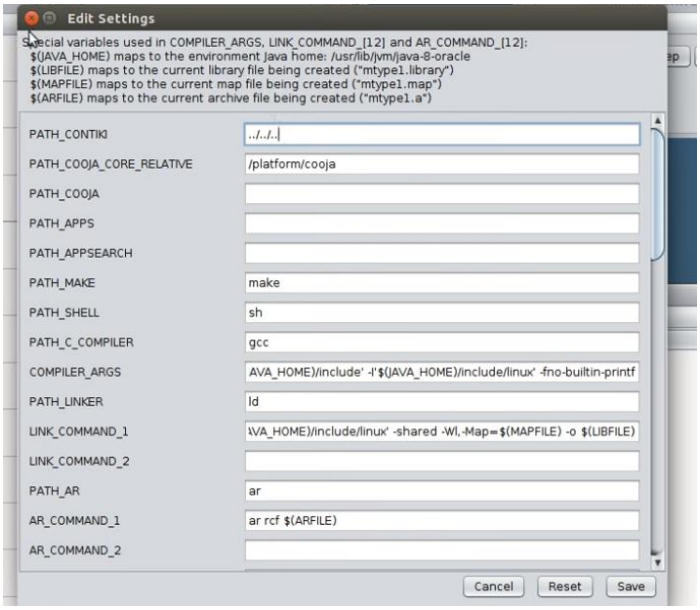


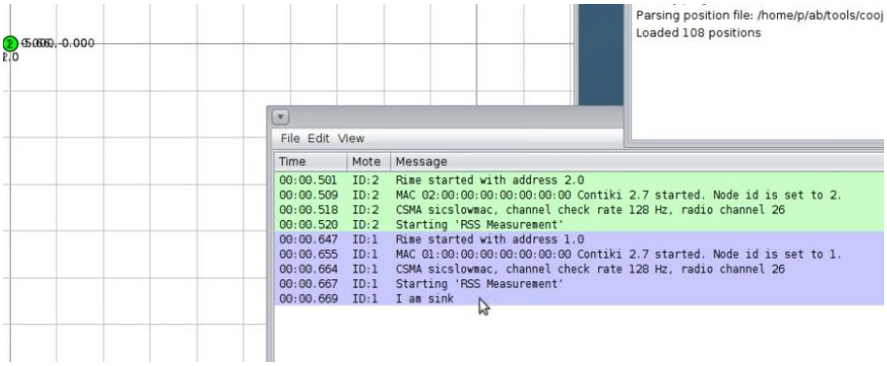
On peut aussi personnaliser l’arrière-plan de contiki

https://www.nsnam.com

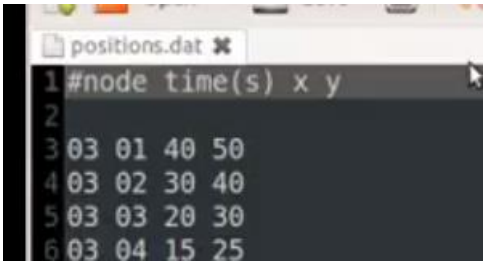
~contiki-2.7/examples/ipv6/rpl-border-router/border-router.c (This firmware is used for one of the sky mote)

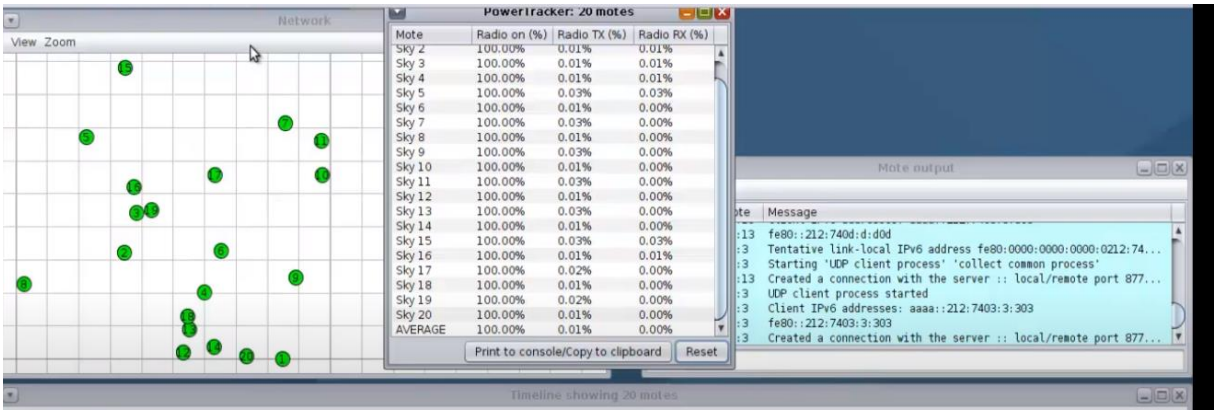
~contiki-2.7/examples/skywebsense/skywebsense.c (this is programmed with other 5 nodes)



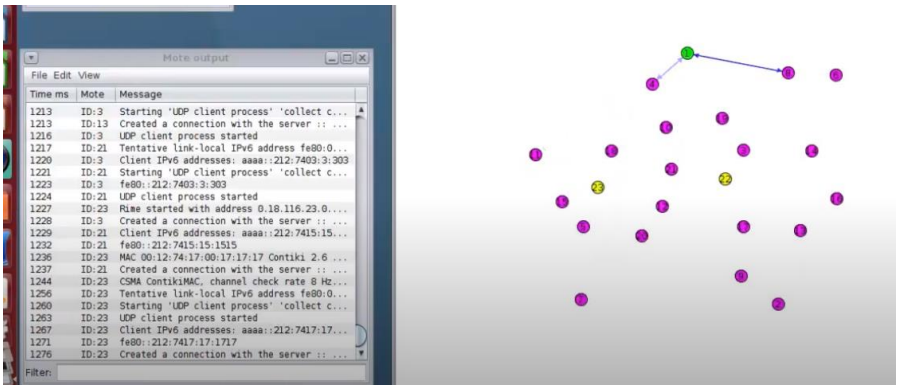


Position.dat : sont indiquer dans ce fchier les positions



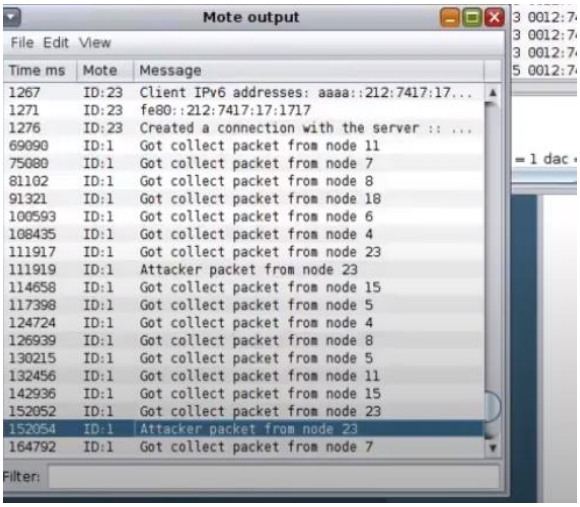


On peut détecter des intrusions avec conitikii :



Grâce au radio message :

En surveillant le mode output :



Sujet de TP annexe section III

Démarrer la machine virtuelle **Instant Contiki**. Se logguer avec le mot de passe **user**

Démarrer l'environnement d'émulation **Cooja** et créer une nouvelle simulation

L'environnement présente différentes fenêtres dont :

- **Network** : fenêtre représentant les différents motes et leur positionnement
- **Simulation Control** : fenêtre permettant de contrôler le déroulement de la simulation
- **Mote output** : fenêtre permettant de récupérer les logs des motes

Fiare un hello world

Dans **Applications/Accessories/Text Editor**

On ouvre `~/contiki/examples/hello-world/hello-world.c  ou avec nano`

1) Quelles sont les macros nécessaires à un programme minimal Contiki

2) A quelle librairie supplémentaire ce programme fait appel, et dans quel but ?

Pour compiler ce premier exemple, il faut d'abord créer un mote dans l'environnement que nous chargerons avec le firmware de ce programme.

Créer un mote dans l'environnement **Cooja** à partir du menu **Motes/Add motes**. Choisissez le type **Sky mote**

Dans la fenêtre de dialogue, indiquer le chemin vers le fichier **hello-world.c**

Lancer la compilation du programme par le bouton **Compile**

3) A quoi correspondent les nombreux fichiers C compilés avec votre programme ? Pourquoi sont-ils compilés ?

Créer le mote une fois la compilation terminée

Une fois le mote positionné dans la fenêtre **Network**, vous pouvez démarrer la simulation à partir de la fenêtre **Simulation Control** et observez le résultat dans la fenêtre **Mote Output**.

4) A quoi correspondent les premières lignes affichées ?

Précision d'un timer sous Linux

Cette expérimentation vise à montrer la relativité des timers sous Linux et de l'impact de l'architecture du système dans le traitement des interruptions.

Ouvrez l'éditeur de fichier dans la Machine Virtuelle **Applications/Accessories/Text Editor**

Saisissez le programme suivant et nommez le *usleep_test.c* :

```
#include <stdio.h>
#include <sys/time.h>

#define SIZE_WINDOW 100

unsigned int usecs = 20000;
struct timeval start, end, diff;
double elapsed;
int t;
int mean_t;
int last_t[SIZE_WINDOW];
int counter = 0;
int ready_to_print = 0;

void main() {
      while (1) {
               gettimeofday(&start, 0);
               usleep(usecs);
               gettimeofday(&end, 0);
               timersub(&end, &start, &diff);
               t = diff.tv_usec;
               last_t[counter++] = t;
               counter %= SIZE_WINDOW;
               ready_to_print = (counter == SIZE_WINDOW - 1);
               if (ready_to_print)
               {
                    mean_t = 0;
                    for (int i = 0; i < SIZE_WINDOW; ++i)
```

```
                    {
                        mean_t += last_t[i];
                    }
                    mean_t /= SIZE_WINDOW;
                    printf("%d us\n", mean_t);
                }
        }
}
```

5)En vous aidant des pages man, donnez le sens des appels système `gettimeofday` et `usleep`.

6)Quelle est la finalité de ce programme ?

Sauvegardez et compiler ce programme depuis un terminal, puis executez le.

```
$ gcc -std=c99 -D_BSD_SOURCE -o usleep_test usleep_test.c
$ ./usleep_test
```

7)Que constatez vous ? Expliquez le.

Nous allons observer l'influence de la charge du système sur le réveil de ce programme.

Tout en laissant le programme s'exécuter, lancer la commande suivante dans un autre terminal

```
$ cat /dev/zero | gzip - > /dev/null
```

Cette commande a pour effet d'occuper le temps CPU au maximum. Vous pouvez vérifier en observant la valeur de la charge par la commande `top`.

8)Que constatez vous pour le programme usleep ? Expliquez le.

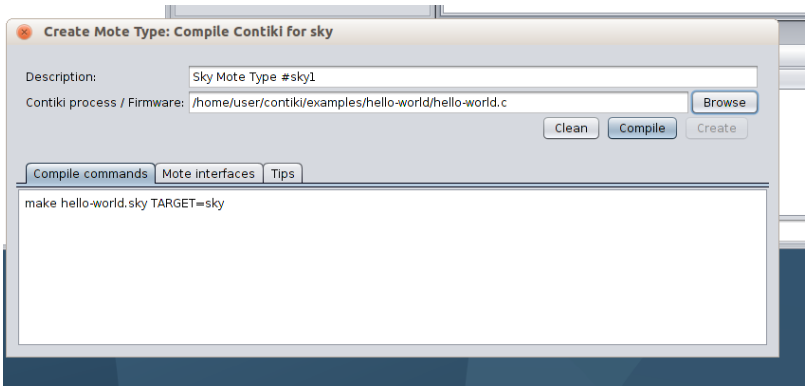
Solution :

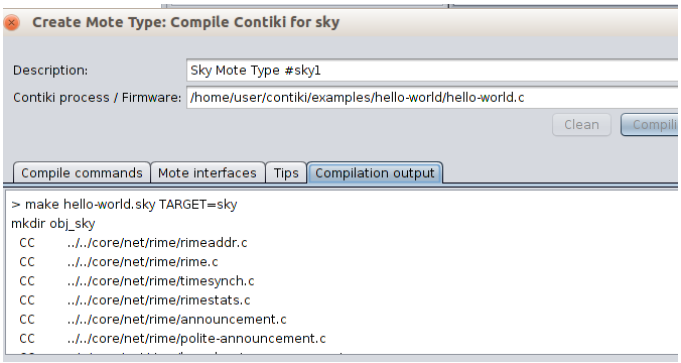
1) Process, autotsart_processes, process_thread, process_begin, process_end

```c
#include "contiki.h"

#include <stdio.h> /* For printf() */
/*---------------------------------------------------------------------------*/
PROCESS(hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(hello_world_process, ev, data)
{
  PROCESS_BEGIN();

  printf("Hello, world\n");

  PROCESS_END();
}
/*---------------------------------------------------------------------------*/
```

2) lirairies : stdio.h pour le c et printf et tt contiki pour gérer les fonction process_therad etc

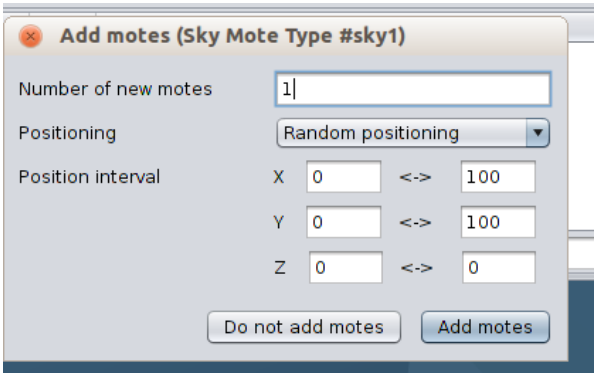
3)

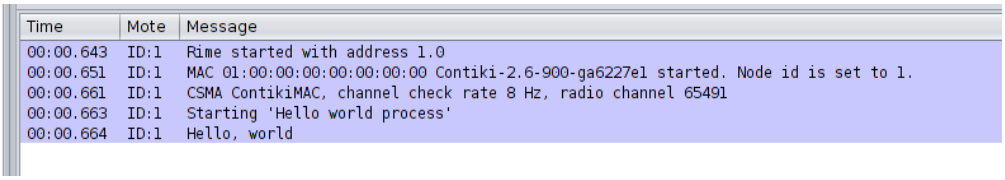




3) A quoi correspondent les nombreux fichiers C compilés avec votre programme ? *A limport de la librarie des dépendances* requises Pourquoi sont-ils compilés ? Ils sont compilés car c'est du c c'est un langage compilé et non iter^rété comme le python

Ce sont l'ensemble des librairies constituant le système Contiki. Ils sont compilés pour constituer le firmware qui sera exécuté sur le mote.





Time	Mote	Message
00:00.643	ID:1	Rime started with address 1.0
00:00.651	ID:1	MAC 01:00:00:00:00:00:00:00 Contiki-2.6-900-ga6227e1 started. Node id is set to 1.
00:00.661	ID:1	CSMA ContikiMAC, channel check rate 8 Hz, radio channel 65491
00:00.663	ID:1	Starting 'Hello world process'
00:00.664	ID:1	Hello, world

4) A quoi correspondent les premières lignes affichées ? Cela correspond au nombre dadresse ladresse mac le contiki version ; le radio channel et le message hello world A l'initialisation du système Contiki

5) `gettimeofday permet de récuperer lheure et le smiutes` et `usleep` permet de ssuprenndre des actions pedant un certains temps

gettimeofday: date du système au moment de l'appel à la fonction

usleep: Mise en pause du programme pendant X micro-secondes

6)Quelle est la finalité de ce programme ? Il sert à définir à miuteur

Mesurer le temps exact entre la mise en pause et le réveil du programme

```
usleep_test.c: In function ‘main’:
usleep_test.c:28:20: error: ‘for’ loop initial declarations are only allowed in
C99 mode
usleep_test.c:28:20: note: use option -std=c99 or -std=gnu99 to compile your cod
e
usleep_test.c:35:7: error: expected declaration or statement at end of input
user@instant-contiki:~/contiki/examples/mysamples$ man gettimeofday
user@instant-contiki:~/contiki/examples/mysamples$ man usleep7
No manual entry for usleep7
user@instant-contiki:~/contiki/examples/mysamples$ man usleep
user@instant-contiki:~/contiki/examples/mysamples$ gcc -std=c99 -D_BSD_SOURCE -o
 usleep_test usleep_test.c
usleep_test.c: In function ‘main’:
usleep_test.c:18:15: warning: implicit declaration of function ‘usleep’ [-Wimpli
cit-function-declaration]
usleep_test.c:35:7: error: expected declaration or statement at end of input
user@instant-contiki:~/contiki/examples/mysamples$
```

7)Que constatez vous ? Expliquez le. *U erreue vcar expected declartion at end statement inpute* Le programme retourne un temps légèrement supérieur à 20000us. Cela correspond au temps de passage entre le mode kernel où est traité l'interruption de réveil et le mode user où le programme reprend. On constate de plus que cette différence varie.

8)Que constatez vous pour le programme usleep ? Expliquez le. *Usleep bloque les ressources de la ppereil* La différence entre le temps réel de réveil et le temps programmé augmente en moyenne. Cela est dû à l'ordonnancement des processus.

10) Sachant qu'il y a 128 TICK par seconde, le nombre retourné correspond-t'il à ce qui est attendu ?

== Second programme : Faire clignoter une LED avec Contiki ==