



# **TP IoT : Simulation de réseaux de capteurs avec Cooja**

Dosseh KOUTO

Novembre 2019

## Table des matières

<b>1. Introduction.....</b>	<b>3</b>
<b>2. Description du simulateur .....</b>	<b>3</b>
<b>3. Les différents outils nécessaires .....</b>	<b>3</b>
<b>4. Répertoires pertinents.....</b>	<b>4</b>
<b>5. Lancement du simulateur Cooja .....</b>	<b>4</b>
<b>6. Création d'une nouvelle simulation .....</b>	<b>6</b>
<b>7. Fenêtre de simulation .....</b>	<b>7</b>
<b>8. Création d'un nouveau type de mote .....</b>	<b>8</b>
<b>9. Description de la simulation de base effectuée avec Cooja (Hello World) .....</b>	<b>9</b>
a. Ajout de motes et exécution de la simulation .....	9
b. Enregistrement du fichier de simulation.....	9
c. Code du hello world.....	10
d. Modification du hello world .....	10
e. Aperçu de la simulation .....	11
<b>10. Diffusion d'un message à travers le réseau.....</b>	<b>11</b>
<b>11. Test de la consommation d'énergie avec powertrace .....</b>	<b>12</b>
<b>12. Mise en pratique .....</b>	<b>14</b>
a. Exercice .....	14
b. Exemple de mise en œuvre .....	14
c. Affichage de Consommation la d'énergie .....	15
d. Réalisation de l'application .....	15
<b>13. Conclusion .....</b>	<b>27</b>
<b>14. Références .....</b>	<b>27</b>

## 1. Introduction

Ce tutoriel vise à présenter le simulateur Cooja et à guider le lecteur dans un exercice de simulation des capteurs. Nous allons travailler dans un premier temps avec l'exemple de Hello World pour vous familiariser avec son code source, Broadcast pour la Diffusion d'un message à travers le réseau et powertrace pour tester la consommation d'énergie des nœuds.

## 2. Description du simulateur.

COOJA est l'acronyme de Contiki OS Java Simulator. Pour développer les programmes au sein de Contiki, le système met à disposition un simulateur réseau appelé Cooja. Le logiciel permet d'émuler des nœuds et de charger un programme compilé. Ceci est particulièrement utile pour tester les programmes avant de les mettre dans la mémoire flash des nœuds réels, puisque le logiciel simule les conditions d'exécution et de mémoire de la plateforme TI MSP430. Les données collectées provenant du sink via sa sortie standard peuvent être enregistrées dans des fichiers ou lues par des logiciels qui peuvent par la suite traiter et présenter les données à l'utilisateur. On peut par exemple citer le logiciel collect-view intégré dans Contiki qui permet de visualiser les valeurs des capteurs et des données de supervision du réseau. En revanche, Cooja a un intérêt limité si l'on veut tester des algorithmes de géolocalisation basés sur le RSSI. En effet, le logiciel utilise un modèle linéaire pour simuler ces valeurs en fonction de la distance, alors qu'en réalité, le modèle est logarithmique. De plus, l'environnement simulé ne reflète pas la réalité en raison de la non prise en compte des perturbations engendrées par les murs, sols...

## 3. Les différents outils nécessaires :

### ➤ Ant :

- Ant est un logiciel créé par la fondation Apache qui vise à automatiser les opérations répétitives du développement de logiciel telles que la compilation, la génération de documents (Javadoc) ou l'archivage au format JAR, à l'instar des logiciels Make (source Wikipedia).
- Pour installer ce premier paquet, il faudra utiliser la commande suivante dans un terminal : **sudo apt-get install ant**

### ➤ Le compilateur MSP430 :

- Cooja simule les communications réseau en utilisant l'émulateur MSPSim pour émuler finement (au niveau des instructions) l'exécution d'un programme sur une plateforme basé sur un processeur MSP430.

- Il nous faudra donc un compilateur adapté à cette architecture.
- La commande suivante permet son installation :  
**sudo apt-get install gcc-msp430.**

➤ **Le JDK :**

- Apache Ant étant écrit en Java, il a besoin d'une machine virtuelle (JVM : Java Virtual Machine) pour fonctionner.
- Nous installerons donc Open-JDK (Java Development Kit) pour pouvoir l'utiliser. **sudo apt-get install openjdk-8-jdk-headless.**

➤ **Téléchargement et installation de Contiki :**

- Contiki 2.7 peut être téléchargé à l'adresse suivante :  
<https://sourceforge.net/projects/contiki/files/Contiki/Contiki%202.7/contiki-2.7.zip/download>.
- Il faut ensuite décompresser le fichier obtenu dans le répertoire Home de votre utilisateur : **unzip contiki-2.7.zip**

## 4. Répertoires pertinents

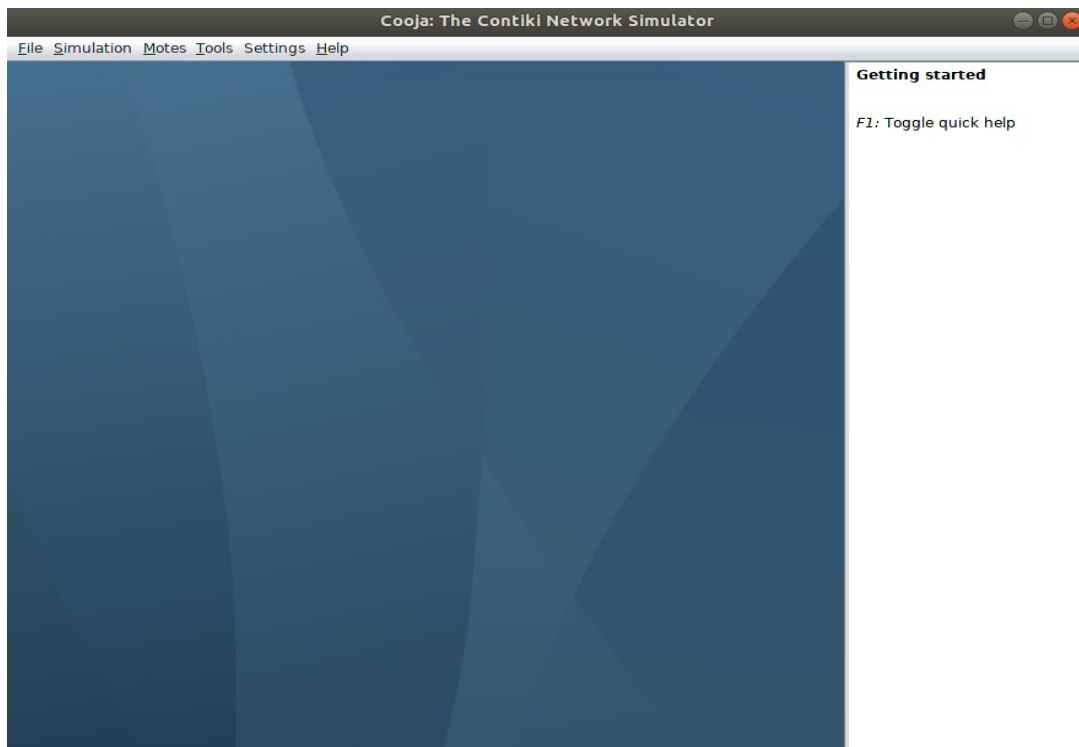
- **/ contiki / tools / cooja / :** Ce dossier contient le code source de Cooja Simulator. Vous pouvez exécuter Cooja dans ce répertoire.
- **/ contiki / examples / hello-world / :** Ceci a le code source de l'exemple Hello World.

## 5. Lancement du simulateur Cooja

La manière simple d'exécuter Cooja est de l'exécuter dans son propre répertoire.

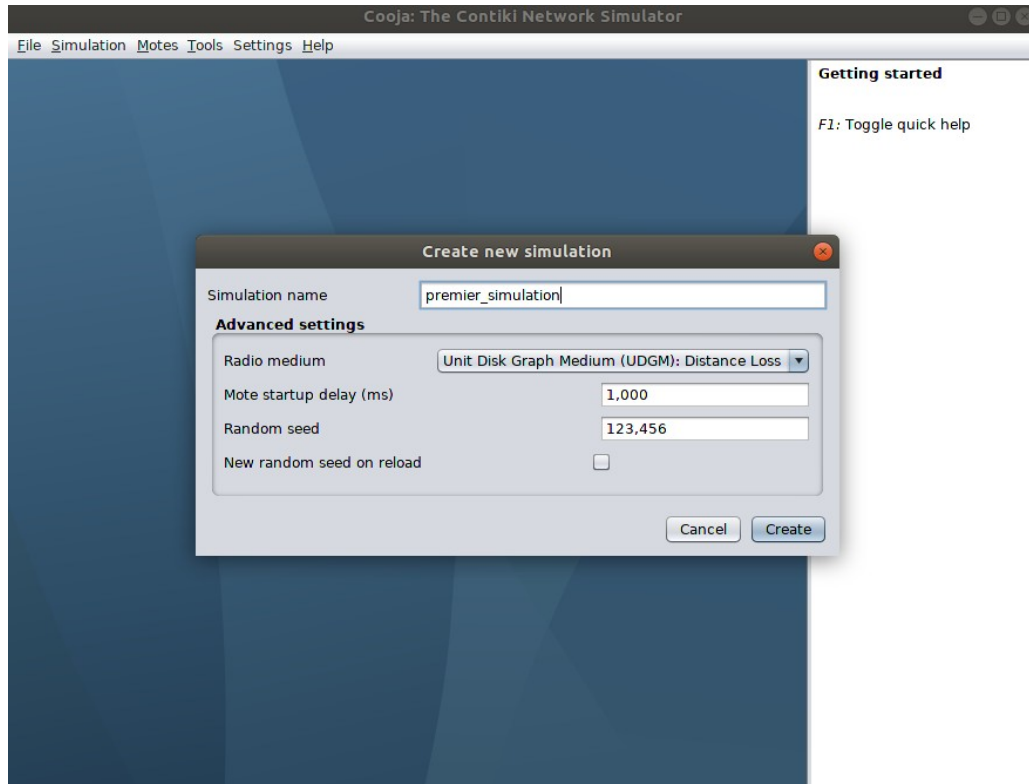
```
cd contiki/tools/cooja  
ant run
```

Lorsque vous exécutez Cooja, la fenêtre suivante s'ouvre.



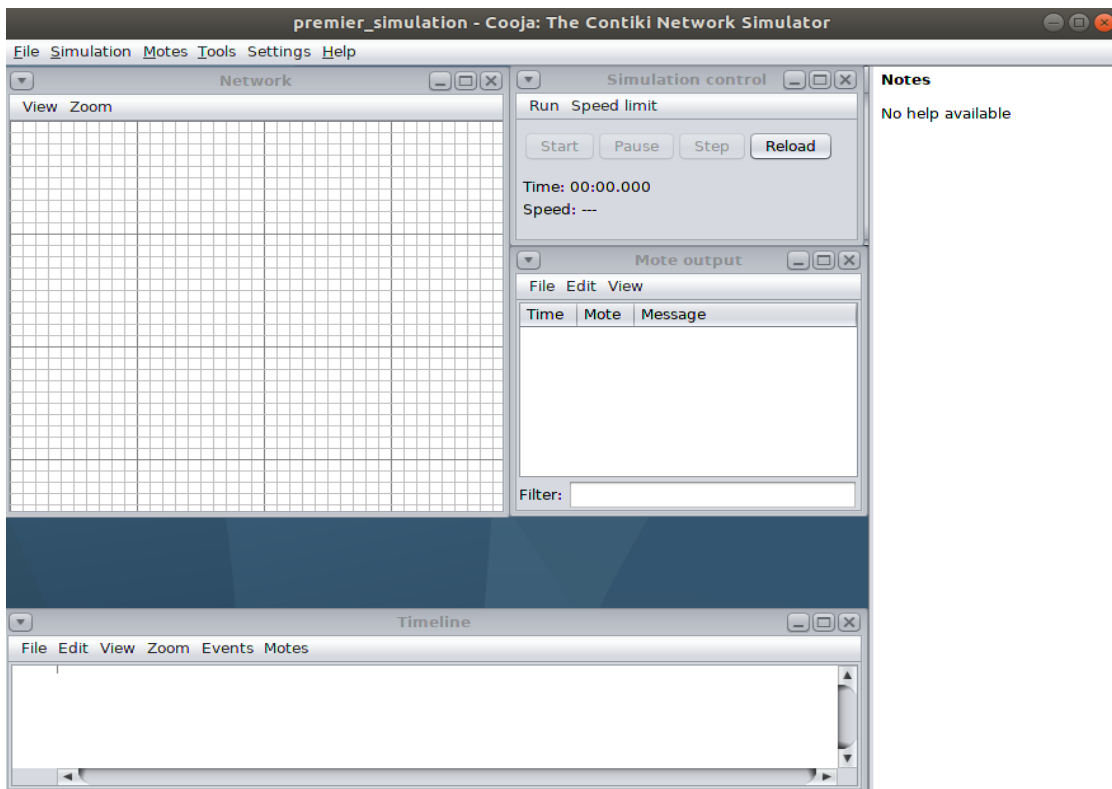
## 6. Création d'une nouvelle simulation :

Dans le menu Fichier, vous pouvez démarrer une nouvelle simulation ou ouvrir une simulation existante. En ce moment, nous allons en commencer un nouveau. Vous devriez choisir : **File > New simulation....** La fenêtre suivante devrait apparaître.



Dans la zone **Simulation name** (Nom de la simulation), vous devez entrer un identificateur pour la nouvelle simulation et dans les réglages avancés, vous pouvez choisir les paramètres de la simulation tels que le support radio, le délai de démarrage des nœuds et la génération aléatoire de semences. Nous allons créer une simulation appelée "premier\_simulation", comme montré ci-dessus.

Après avoir créé une nouvelle simulation, la fenêtre de Cooja est remplie des principaux outils de simulation, comme le montre l'image suivante.



## 7. Fenêtre de simulation :

Dans une simulation nous avons plusieurs fenêtres. Nous décrivons ici brièvement les fonctionnalités de chaque fenêtre :

### ➤ **Network (Réseau) :**

En haut à gauche de l'écran, affiche l'emplacement de chaque nœud du réseau. Permet de visualiser l'état de chaque nœud, y compris les LEDs, les identificateurs de mote, les adresses, les sorties lof, etc. Au départ, cette fenêtre est vide et nous devons la remplir avec nos capteurs.

### ➤ **Simulation Control (Contrôle de simulation) :**

Sur le côté droit de l'écran, 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 rapidement qu'en temps réel.

### ➤ **Notes :**

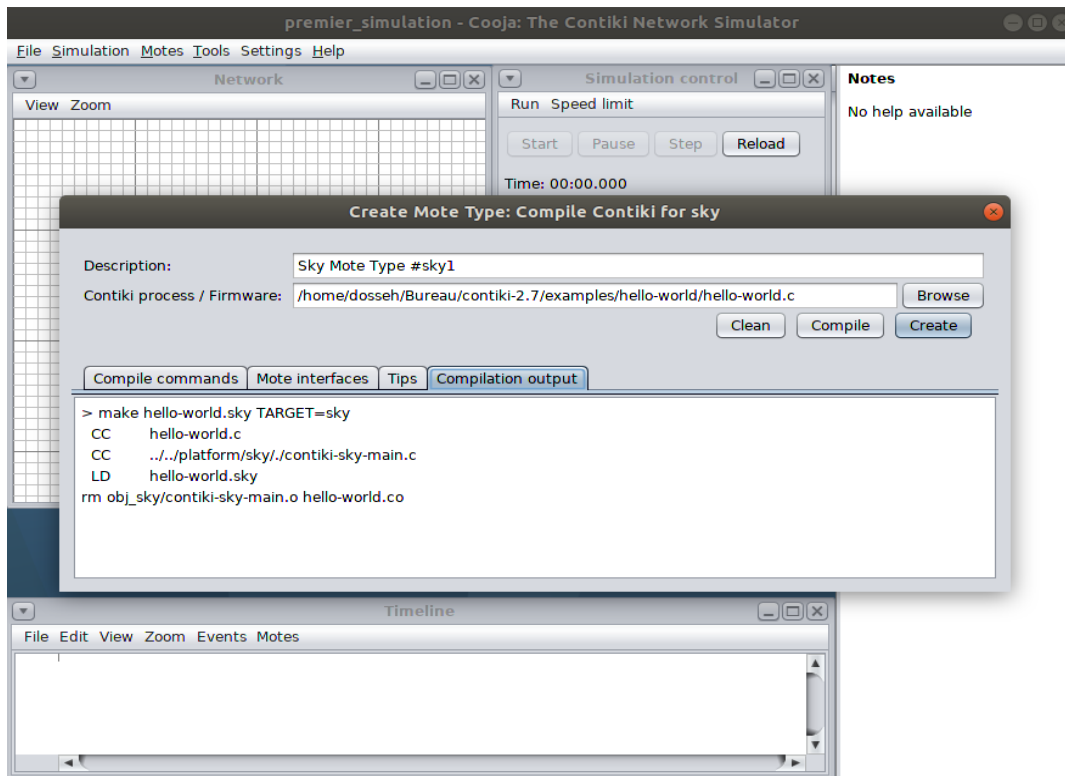
En haut à droite Il s'agit d'un simple bloc-notes pour prendre des notes sur la simulation.

### ➤ **Mote output (Sortie Mote):**

Sur le côté droit de l'écran, affiche toutes les sorties de l'interface série des nœuds. Il est possible d'activer une fenêtre de sortie Mote pour chaque nœud de la simulation.

### ➤ **Timeline (Ligne de temps de simulation) :**

En bas de l'écran, où les messages et les événements tels que le changement de canal, les changements de LED, les sorties de journal, etc. sont affichés.



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 tampon et le duty cycle Mote, qui peuvent être activés dans le menu Tools.

## 8. Création d'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 notes > Create new notes type**. Sélectionnons **Sky mote** afin de créer une mote du même type que le **Tmote Sky** utilisé.

La fenêtre qui apparaît (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 firmware qui sera utilisé pendant la simulation en utilisant le bouton Parcourir. Après avoir sélectionné le firmware souhaité, vous pouvez tester la compilation en cliquant sur le bouton Compile.



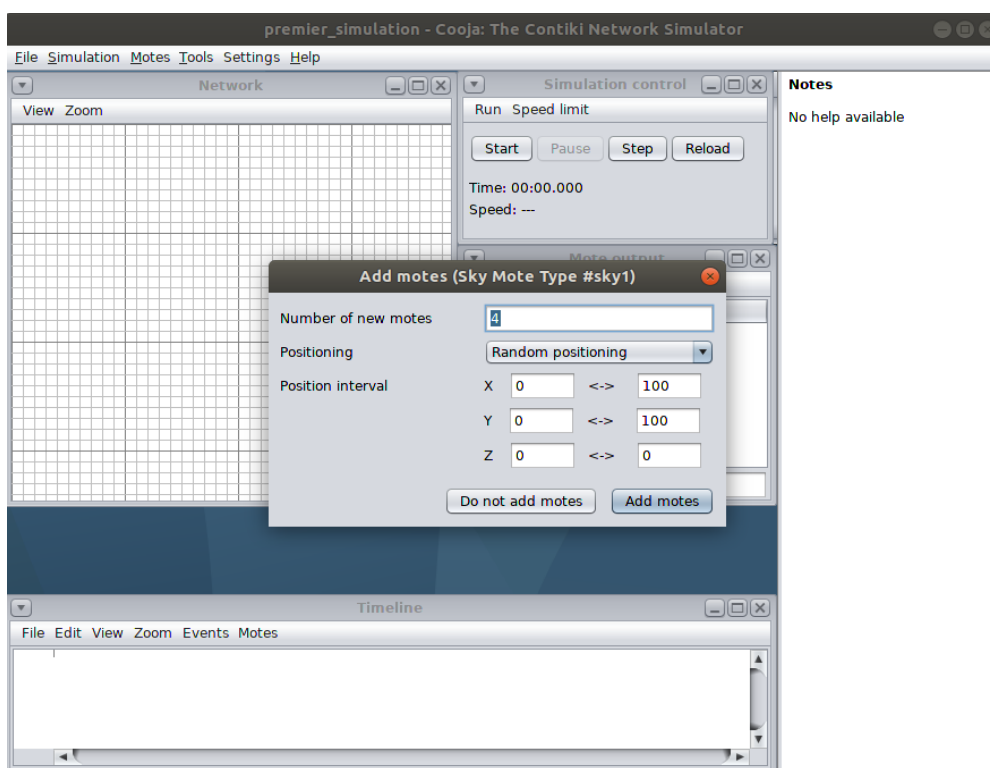
## 9. Description de la simulation de base effectuée avec Cooja (Hello World).

Dans cet exemple, nous utiliserons le firmware Hello World, généralement situé à **/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.

### a. Ajout de nœuds et exécution de la simulation :

Après avoir compilé avec succès le firmware à la dernière étape, vous devez cliquer sur le bouton **created** (voir figure ci-dessus). Un nouveau type de nœud sera créé et vous pourrez ajouter quelques nœuds du même type dans votre simulation. Vous verrez la fenêtre ci-dessous, dans laquelle vous pourrez définir le nombre de nœuds qui seront créés et spécifier leur position. Dans cet exemple, nous allons créer 4 nouveaux nœuds et ils auront un positionnement aléatoire.

Ensuite, on peut démarrer la simulation.



### b. Enregistrement du fichier de simulation :

La configuration et les paramètres de la simulation tels que le nombre de nœuds, le type de nœuds, le firmware utilisé, l'emplacement des nœuds, etc. peuvent être

stockés dans un fichier pour des simulations futures. Vous pouvez enregistrer votre configuration de simulation dans **File > Save simulation as....**

### c. Code du hello world

```
//----- Debut Hello World -----
#include "contiki.h"
#include <stdio.h>
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();
}
//----- Fin Hello World -----
```

### d. Modification du hello world :

On va modifier le code comme suit afin de faire clignoter les 4 LED.

```
//----- Debut Hello World Modifier -----
#include <stdio.h> /* For printf() */
#include "contiki.h"
#include "dev/leds.h"
#include "lib/random.h"
#include "sys/ctimer.h"
#include "sys/etimer.h"

static struct etimer et_hello;
static uint16_t count;
static uint8_t blinks;
PROCESS(hello_world_process, "[Exemple] Hello world avec affichage des LED");
AUTOSTART_PROCESSES(&hello_world_process);

PROCESS_THREAD(hello_world_process, ev, data)
{
    PROCESS_BEGIN();
    count = 1;
    blinks = 1;
    while(1) {
        etimer_set(&et_hello, CLOCK_SECOND*30); // Start the timer
        PROCESS_WAIT_EVENT_UNTIL(ev == PROCESS_EVENT_TIMER);
        printf("[Exemple] Cycle : #%u\n", count);
        count++;
        leds_off(LEDS_ALL);
        if (blinks==1){
            leds_on(LEDS_BLUE);
            printf("[Exemple] Passage au bleu\n");
        }
        else if (blinks==2){
```

```

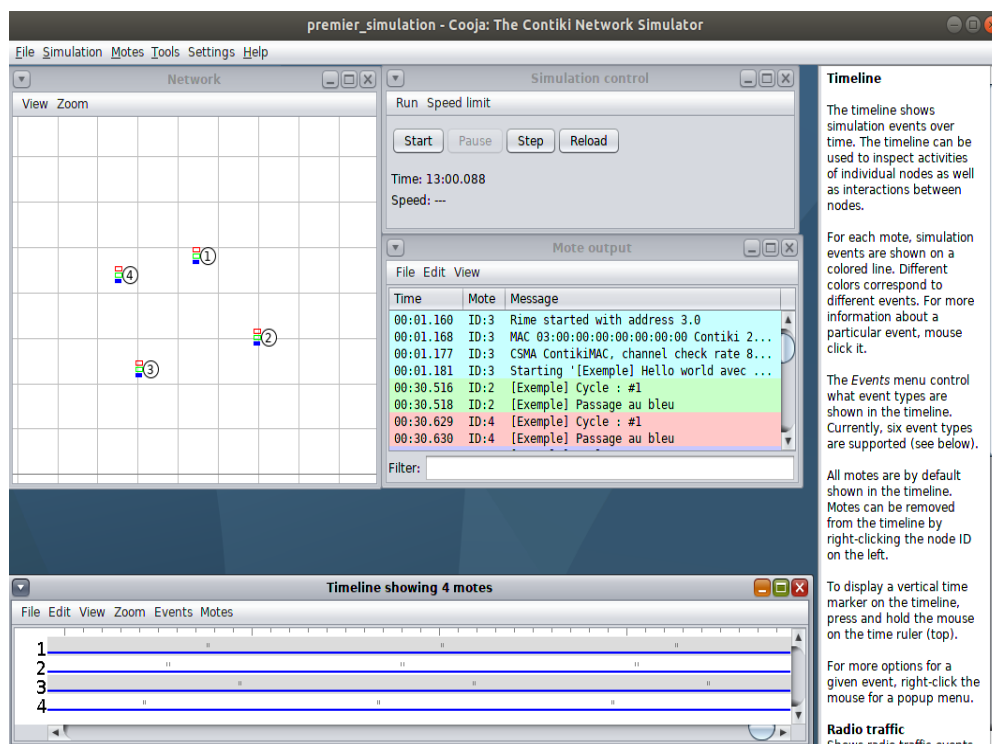
leds_on(LEDS_RED);
printf("[Exemple] Passage au rouge\n");
}
else if (blinks==3){
leds_on(LEDS_GREEN);
printf("[Exemple] Passage au vert\n");
blinks=0;
}
blinks++;
}
PROCESS_END();
}

//----- Fin Hello World Modifier -----

```

### e. Aperçu de la simulation :

Voici l’affichage de la simulation des 4 nœuds en activant le mode de vue LED afin d’observer le changement de couleur des LEDs.

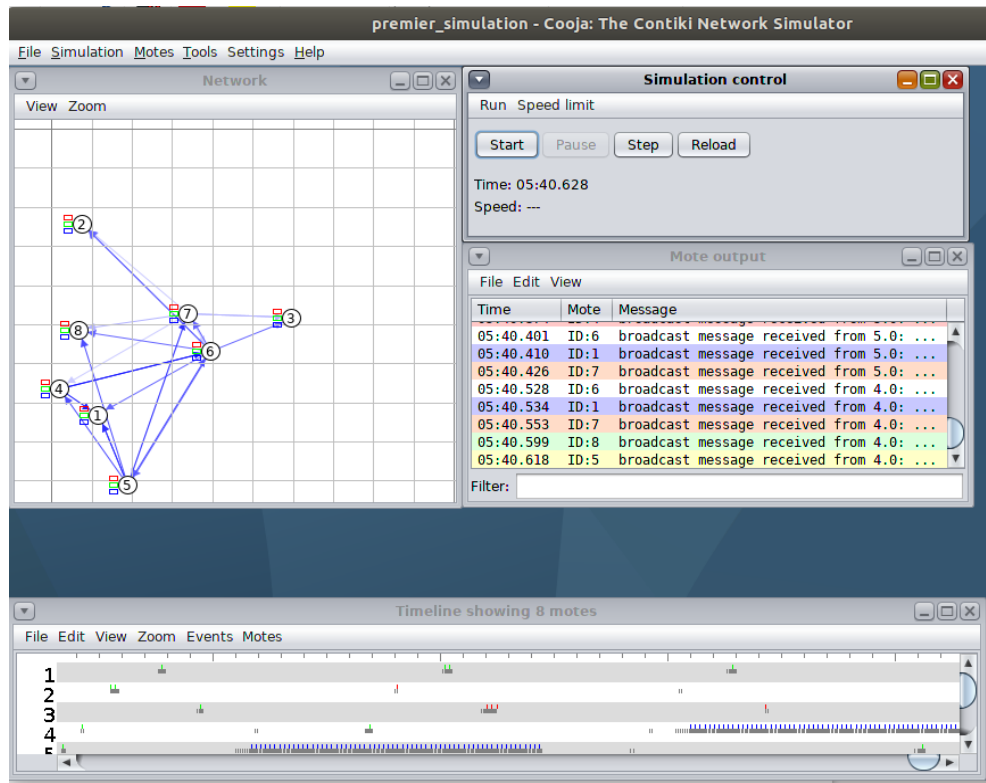


## 10. Diffusion d’un message à travers le réseau.

On va apprendre à faire la diffusion de message dans un réseau.  
Prenons le code d’un exemple de diffusion d’un message en mode "**broadcast**".

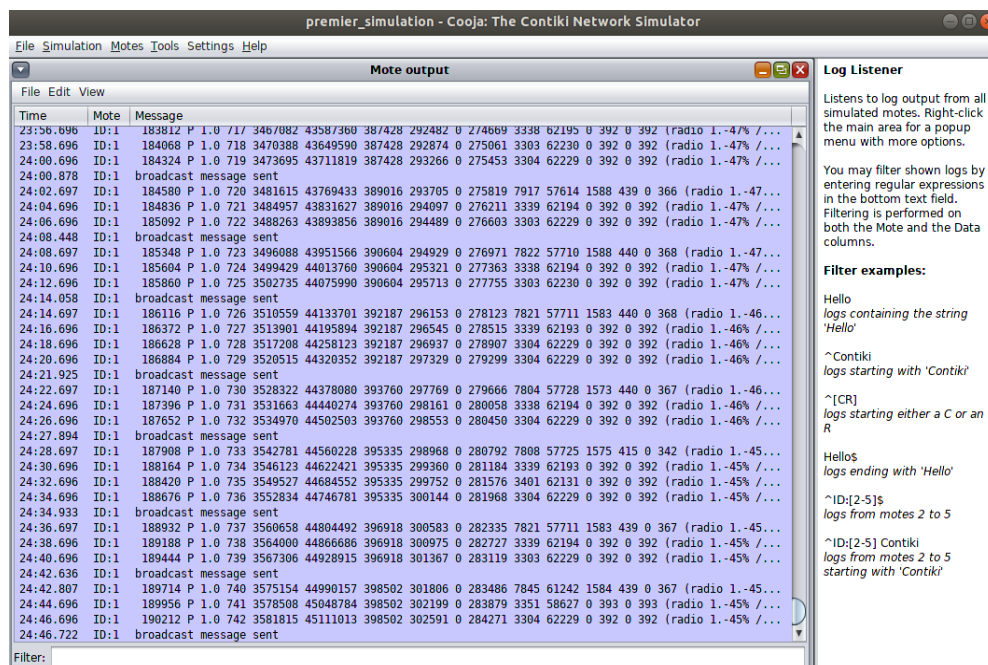
Il s'agit du code de l'exemple "**rime**" et du fichier "**example-broadcast.c**" situé à **/contiki/examples/rime/example-broadcast.c**.

Dans cet exemple, nous allons créer 8 nouveaux nœuds et ils auront un positionnement aléatoire. Ensuite, on démarre la simulation.



## 11. Test de la consommation d'énergie avec powertrace:

On va apprendre à faire le test sur la consommation d'énergie pour les nœuds grâce à la librairie PowerTrace.



Prenons l'exemple "**powertrace**" et du fichier "**example-powertrace.c**" situé à **/contiki/examples/powertrace/example-powertrace.c**.

Dans cet exemple, nous allons créer 1 nouveau nœud.

Ensuite, on démarre la simulation.

Powertrace nous affiche la consommation d'énergie pour les nœuds chaque 2 second dans ce cas.

Il est bien à savoir qu'on peut tester la consommation d'énergie sur n'importe quel programme de simulation. Pour se faire, il suffit d'inclure la bibliothèque powerTrace comme suit : **#include "powertrace.h"** et de faire un appel de la commande **powertrace\_start(CLOCK\_SECOND \* X);** avec **X** le temps en seconde, qui permet d'afficher la consommation d'énergie. Cette commande est implémenter dans le code juste après la commande **PROCESS\_BEGIN()**.

Il est nécessaire d'ajouter dans le fichier **Makefile** la commande suivante :**APPS+=powertrace**.



```
CONTIKI_PROJECT = example-powertrace
APPS+=powertrace
all: $(CONTIKI_PROJECT)

CONTIKI = ../..
include $(CONTIKI)/Makefile.include
```

The screenshot shows a text editor window titled "Makefile" with the path "~/Bureau/contiki-2.7/examples/powertrace". The editor contains the following Makefile content: `CONTIKI_PROJECT = example-powertrace`, `APPS+=powertrace`, `all: $(CONTIKI_PROJECT)`, `CONTIKI = ../..`, and `include $(CONTIKI)/Makefile.include`. The status bar at the bottom indicates "Makefile", "Largeur des tabulations : 8", "Lig 6, Col 36", and "INS".

## 12. Mise en pratique :

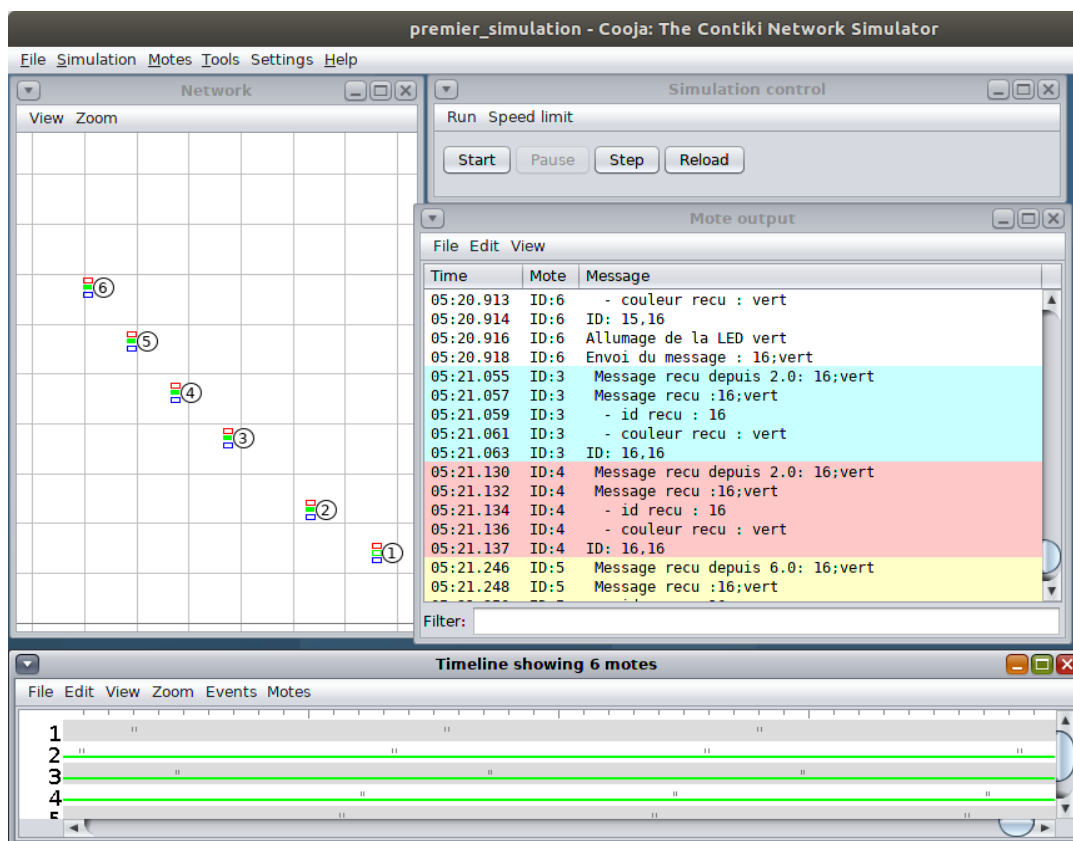
### a. Exercice :

En combinant les trois exemples précédents, nous allons essayer de créer une application basée sur deux types de capteurs.

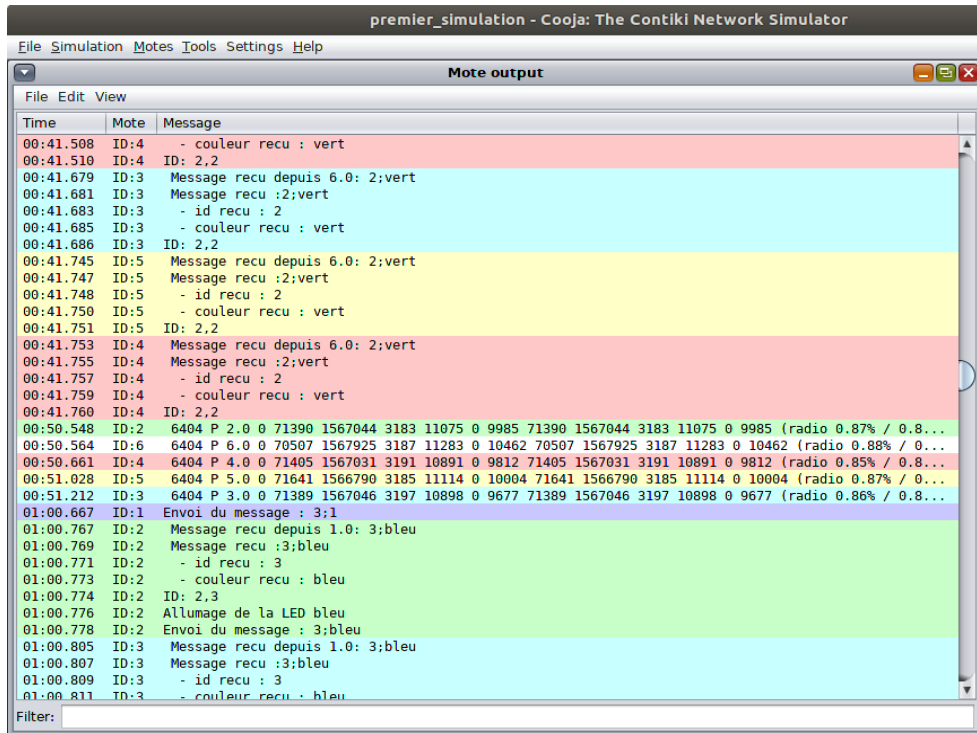
### Description de la simulation :

- Le premier type de capteur diffusera un message de changement de couleur de LED en mode diffusion. Ce message devra comporter un identifiant.
  - Il faudra attendre 20 secondes entre chaque envoi de messages pour laisser du temps pour la propagation.
  - L'identifiant pourra se baser sur un numéro d'envoi.
- Le second type de capteur :
  - changera de couleur lorsqu'il recevra un message de changement de couleur,
  - propagera à son tour le message de changement de couleur,
  - contrôlera que le message n'a pas déjà été reçu (d'après son identifiant).
- mesurer les différentes consommations d'énergie pour les nœuds grâce aux bibliothèques PowerTrace.

### b. Exemple de mise en œuvre :



### c. Affichage de Consommation la d'énergie :



### d. Réalisation de l'application.

Tout d'abord créer un dossier nommé «TP\_cooja» dans le répertoire /contiki/examples/ .

Créer dans ce répertoires ces fichiers suivants :

- capteur\_difusion.c
- capteur\_de\_reception.c
- Makefile

#### ➤ Code de capteur\_difusion.c avec commentaires :

```
//----- Debut capteur de difusion -----
#include "powertrace.h"
#include "contiki.h"
#include "net/rime.h"
#include "random.h"
#include "dev/button-sensor.h"
#include <stdlib.h>
#include "dev/leds.h"
#include <string.h>
#include <stdio.h>
static struct etimer et_hello;
```

```

/*
Ci-dessous, vous avez la definition de la structure "Broadcast_type" qui comporte 2 entier (id_diffusion et
couleur)
qui vont nous permettre d'envoyer un message de diffusion contenant un identifiant et une couleur.
*/
struct Broadcast_type
{
int id_diffusion;
int couleur;
};
typedef struct Broadcast_type broadcast_type;

static struct broadcast_conn broadcast;
static void envoie(int id, int couleur);
static int generer_nombre_aleatoire(int min, int max);

PROCESS(example_broadcast_process, "[Exemple2] Exemple de diffusion");
AUTOSTART_PROCESSES(&example_broadcast_process);

static void broadcast_rcv(struct broadcast_conn *c, const rimeaddr_t *from)
{
// L'implementation de broadcast_rcv pour l'interpretation du message se trouve dans le fichier
"capteur_de_reception.c"
}

static const struct broadcast_callbacks broadcast_call = {broadcast_rcv};

PROCESS_THREAD(example_broadcast_process, ev, data)
{
static struct etimer et;
static int cpt = 0;
PROCESS_EXITHANDLER(broadcast_close(&broadcast);)

PROCESS_BEGIN();
/* Commencez powertracing, une fois toutes les 100 secondes. */
powertrace_start(CLOCK_SECOND*100);
broadcast_open(&broadcast, 129, &broadcast_call);

while (1)
{
cpt++;
etimer_set(&et_hello, CLOCK_SECOND * 20); // attendre 20 secondes entre chaque envoi de messages
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
envoie(cpt, generer_nombre_aleatoire(1, 3)); // On genere un entier aleatoire entre 1 et 3 (avec bleu = 1 ,
vert =2, rouge = 3) a chaque iteration
}
PROCESS_END();
}

/*
Ci-dessous, vous avez la fonction "envoie" qui permet d'envoyer un message de diffusion.
Cette fonction comporte 2 parametre :
- id : identifiant ;

```



- couleur : code de couleur.

le message qui sera envoyer sera de type broadcast\_type (id;couleur)

\*/

```
static void envoie(int id, int couleur)
```

```
{
```

```
  broadcast_type message_envoyer;
```

```
  message_envoyer.id_diffusion = id;
```

```
  message_envoyer.couleur = couleur; // La couleur coder en entier sera decrypter au moment de la reception du message
```

```
  packetbuf_copyfrom(&message_envoyer, 7);
```

```
  broadcast_send(&broadcast);
```

```
  printf("Envoi du message : %d;%d\n", message_envoyer.id_diffusion, message_envoyer.couleur);
```

```
}
```

/\*

Ci-dessous, vous avez la fonction "generer\_nombre\_aleatoire" qui permet de generer un nombre aleatoire entre 1 et 3.

Ces nombres servirons de couleur a la reception du message comme suit : bleu = 1 , vert =2, rouge = 3.

\*/

```
static int generer_nombre_aleatoire(int min, int max)
```

```
{
```

```
  return random_rand() % (max - min + 1) + min; // resultat de l'entier aléatoire
```

```
}
```

//-----Fin capteur de difusion -----

## ➤ Code de capteur\_de\_reception.c avec commentaires :

//----- Debut capteur de reception -----

```
#include "powertrace.h"
```

```
#include "contiki.h"
```

```
#include "net/rime.h"
```

```
#include "random.h"
```

```
#include "dev/button-sensor.h"
```

```
#include <stdlib.h>
```

```
#include "dev/leds.h"
```

```
#include <string.h>
```

```
#include <stdio.h>
```

```
static struct etimer et_hello;
```

/\*

Ci-dessous, vous avez la definition de la structure "Broadcast\_type" qui comporte 2 entier (id\_diffusion et couleur)

qui vont nous permettre de de recuperer un message de difusion contenant un identifiant et une couleur.

\*/

```
struct Broadcast_type
```

```
{
```

```

int id_diffusion;
int couleur;
};
typedef struct Broadcast_type broadcast_type;

static struct broadcast_conn broadcast;
static void generer_couleur_de_blink(int code_couleur);
static char *decripter_code_couleur(int code_couleur);
static int id_dernier_message = 0;

PROCESS(example_broadcast_process, "[Exemple2] Exemple de diffusion");
AUTOSTART_PROCESSES(&example_broadcast_process);

/*
Ci-dessous, vous avez la definition de la fonction broadcast_rcv;
Cette fonction analyse un paquet entrant et affiche le message et l'adresse de l'expéditeur.
En la définissant comme fonction de rappel désignée de la diffusion, broadcast_rcv est
automatiquement appelée lors de la réception d'un paquet.
Cette fonction a 2 paramètre :
- broadcast_conn *: cette structure qui a 2 structures: abc_conn, broadcast_callbacks *.
Abc_conn est le type de connexion de base sur lequel la connexion de diffusion est développée.
De plus, broadcast_callbacks pointe sur rcv et les fonctions envoyées (dans cet exemple, rcv
seulement).
- rimeaddr_t *: Ceci est une union qui a un tableau de caractères u8 [RIMEADDR_SIZE].
*/
static void broadcast_rcv(struct broadcast_conn *c, const rimeaddr_t *from)
{

broadcast_type *message_recu; // declaration de pointeur de type broadcast_type sur message reçu
message_recu = packetbuf_dataptr(); // affectation du message reçu a "message_recu"

printf(" Message reçu depuis %d.%d: '%d;%s'\n", from->u8[0], from->u8[1], message_recu->id_diffusion,
decripter_code_couleur(message_recu->couleur));
printf(" Message reçu : '%d;%s'\n", message_recu->id_diffusion, decripter_code_couleur(message_recu-
>couleur));
printf(" - id reçu : '%d'\n", message_recu->id_diffusion);
printf(" - couleur reçu : '%s'\n", decripter_code_couleur(message_recu->couleur));
printf("ID: '%d,%d'\n", id_dernier_message, message_recu->id_diffusion);

if (message_recu->id_diffusion != id_dernier_message)
{
id_dernier_message = message_recu->id_diffusion;
generer_couleur_de_blink(message_recu->couleur);
printf("Allumage de la LED %s\n", decripter_code_couleur(message_recu->couleur));

packetbuf_copyfrom(message_recu, 7);
broadcast_send(&broadcast);
printf("Envoi du message : %d;%s\n", message_recu->id_diffusion,
decripter_code_couleur(message_recu->couleur));
}

}

```

```
static const struct broadcast_callbacks broadcast_call = {broadcast_rcv};

PROCESS_THREAD(example_broadcast_process, ev, data)
{
    static struct etimer et;
    PROCESS_EXITHANDLER(broadcast_close(&broadcast);)

    PROCESS_BEGIN();
    /* Commencez powertracing, une fois toutes les 50 secondes. */
    powertrace_start(CLOCK_SECOND * 50);

    broadcast_open(&broadcast, 129, &broadcast_call);

    while (1)
    {
        etimer_set(&et_hello, CLOCK_SECOND * 20); // attendre 20 secondes entre chaque reception de
        messages
        PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));
    }
    PROCESS_END();
}

/*
Ci-dessous, vous avez la fonction "generer_couleur_de_blink" qui permet
de generer des couleurs de leds en fonction de son code comme suit 1 = bleu , 2 = vert, 3 = rouge.
cette fonction prend en parametre un entier qui est le code de couleur et allume la LED correspondante.
*/
static void generer_couleur_de_blink(int code_couleur)
{
    leds_off(LEDS_ALL);
    switch (code_couleur)
    {
        case 1:
            leds_on(LEDS_BLUE);
            break;
        case 2:
            leds_on(LEDS_GREEN);
            break;
        case 3:
            leds_on(LEDS_RED);
        default:
            break;
    }
}

/*
Ci-dessous, vous avez la fonction "decripter_code_couleur" qui permet
de decripter un code de couleur passer en parametre puis retourne sa couleur comme suit 1 = "bleu" , 2
= "vert", 3 = "rouge".
*/
static char *decripter_code_couleur(int code_couleur)
{
    switch (code_couleur)

```

```

{
case 1:
return "bleu";
break;
case 2:
return "vert";
break;
case 3:
return "rouge";
}
return "vert";
}

//-----Fin capteur de reception -----

```

### ➤ Code de Makefile :

```

CONTIKI = ../..
APPS = serial-shell powertrace collect-view
all: example-abc example-mesh example-collect example-trickle example-polite \
example-rudolph0 example-rudolph1 example-rudolph2 example-rucb \
example-runicast example-unicast example-neighbors

include $(CONTIKI)/Makefile.include

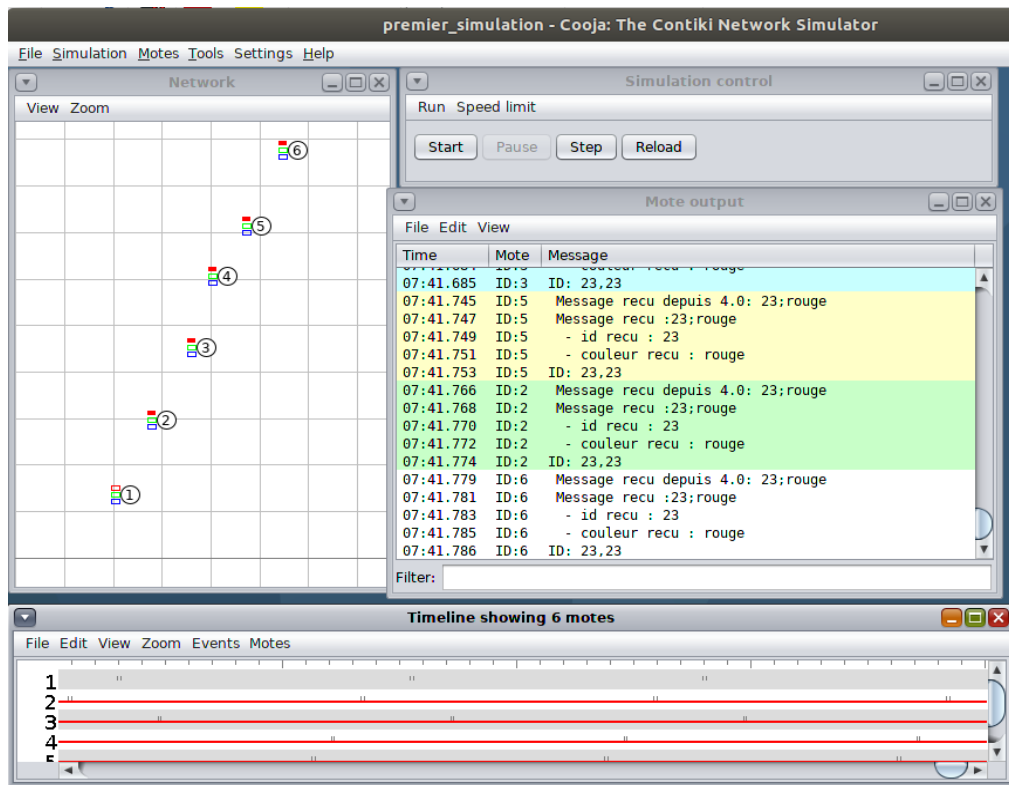
```

Il est bon à savoir qu'on peut sauvegarder les messages de sorties de **Mote Output** en cliquant sur **file>save to file** puis donner un nom au fichier de sauvegarde. Dans notre cas on va le nommer **affichage\_sortie.txt** et on va le sauvegarder dans le dossier **/contiki/examples/TP\_cooja**.

### ➤ Exécution du programme :

Une fois que vous avez implémenté les codes dans les différents fichiers mentionnés récemment, créer une nouvelle simulation. Cliquez sur le menu **Motes > Add notes > Create new motes type**. Sélectionnons **Sky mote** afin de créer une mote du même type que le **Tmote Sky** utilisé, ajoutez le fichier **capteur\_difusion.c** contenant son code, compiler puis ajouter 1 nœuds. Faite de la même manière pour ajouter le fichier **capteur\_de\_reception.c**, compiler puis ajouter 5 nœuds.

Dans la fenetre **Simulation Control**, cliquez sur **start** pour lancer l'exection du programme.



## ➤ Affichage de la consommation d'énergie :

L'affichage de la consommation d'énergie s'effectue chaque 20 secondes dans ce cas. Vous pouvez modifier ce temps comme vous voulez comme suit `powertrace_start(CLOCK_SECOND * X);` avec X le temps en seconde.

Time ms	Mote	Message
20545	ID:2	2564 P 2.0 0 23648 631763 0 3877 0 3877 23648 631763 0 3877 0 3877 (radio 0.59% / 0.59% tx 0.00% / 0.00% listen 0.59% / 0.59%)
20560	ID:6	2564 P 6.0 0 23016 632395 0 3895 0 3895 23016 632395 0 3895 0 3895 (radio 0.59% / 0.59% tx 0.00% / 0.00% listen 0.59% / 0.59%)
20657	ID:4	2564 P 4.0 0 23648 631763 0 3877 0 3877 23648 631763 0 3877 0 3877 (radio 0.59% / 0.59% tx 0.00% / 0.00% listen 0.59% / 0.59%)
21118	ID:5	2575 P 5.0 0 28315 630038 1589 4128 0 3803 28315 630038 1589 4128 0 3803 (radio 0.86% / 0.86% tx 0.24% / 0.24% listen 0.62% / 0.62%)
21211	ID:3	2564 P 3.0 0 28231 627181 1592 4178 0 3816 28231 627181 1592 4178 0 3816 (radio 0.88% / 0.88% tx 0.24% / 0.24% listen 0.63% / 0.63%)
40547	ID:2	5124 P 2.0 1 52986 125773 1661 8356 0 7912 29335 626010 1601 4479 0 4035 (radio 0.75% / 0.92% tx 0.12% / 0.24% listen 0.63% / 0.68%)
40562	ID:6	5124 P 6.0 1 52342 1258416 1599 8425 0 7735 29323 626021 1599 4530 0 3840 (radio 0.76% / 0.93% tx 0.12% / 0.24% listen 0.64% / 0.69%)
40660	ID:4	5124 P 4.0 1 53383 1257374 1593 8415 0 7721 29732 625611 1593 4538 0 3844 (radio 0.76% / 0.93% tx 0.12% / 0.24% listen 0.64% / 0.69%)
41025	ID:5	5124 P 5.0 1 53257 1257503 1589 8610 0 7839 24939 627465 0 4482 0 4036 (radio 0.77% / 0.68% tx 0.12% / 0.00% listen 0.65% / 0.68%)
41211	ID:3	5124 P 3.0 1 57344 1253417 3192 8576 0 7637 29110 626236 1600 4398 0 3821 (radio 0.89% / 0.91% tx 0.24% / 0.24% listen 0.65% / 0.67%)
60548	ID:2	7684 P 2.0 2 82433 1883677 3192 12929 0 11886 29444 625904 1592 4573 0 3974 (radio 0.81% / 0.94% tx 0.16% / 0.24% listen 0.65% / 0.69%)
60563	ID:6	7684 P 6.0 2 81745 1884369 3192 12786 0 11576 29400 625953 1593 4361 0 3841 (radio 0.81% / 0.90% tx 0.16% / 0.24% listen 0.65% / 0.66%)
60660	ID:4	7684 P 4.0 2 82665 1883076 3186 13444 0 11468 29279 626074 1601 4434 0 3747 (radio 0.81% / 0.92% tx 0.16% / 0.24% listen 0.65% / 0.67%)
61027	ID:5	7684 P 5.0 2 83936 1883076 3186 13444 0 12088 29776 625573 1597 4834 0 4249 (radio 0.84% / 0.98% tx 0.16% / 0.24% listen 0.68% / 0.73%)
61211	ID:3	7684 P 3.0 2 86580 1879536 4788 13166 0 11410 29233 626119 1596 4590 0 3773 (radio 0.91% / 0.94% tx 0.24% / 0.24% listen 0.66% / 0.70%)
80549	ID:2	10244 P 2.0 3 111538 2509931 4776 17367 0 15452 29790 625562 1584 4581 0 3876 (radio 0.84% / 0.94% tx 0.18% / 0.24% listen 0.66% / 0.69%)
80563	ID:6	10244 P 6.0 3 111538 2509931 4776 17367 0 15452 29790 625562 1584 4581 0 3876 (radio 0.84% / 0.94% tx 0.18% / 0.24% listen 0.66% / 0.69%)
80661	ID:4	10244 P 4.0 3 112331 2509139 4780 17382 0 15264 29663 625691 1586 4533 0 3796 (radio 0.84% / 0.93% tx 0.18% / 0.24% listen 0.66% / 0.69%)
81028	ID:5	10244 P 5.0 3 112448 2509018 4781 17888 0 15848 29409 625942 1595 4444 0 3760 (radio 0.86% / 0.92% tx 0.18% / 0.24% listen 0.68% / 0.67%)
81212	ID:3	10244 P 3.0 3 115727 2505743 6384 17523 0 15218 29144 626207 1596 4357 0 3808 (radio 0.91% / 0.90% tx 0.24% / 0.24% listen 0.66% / 0.66%)
100549	ID:2	12804 P 2.0 4 141188 3135633 6380 21703 0 19520 29494 625857 1598 4407 0 3823 (radio 0.85% / 0.91% tx 0.19% / 0.24% listen 0.66% / 0.67%)
100563	ID:6	12804 P 6.0 4 140864 3135962 6376 21839 0 19257 29323 626031 1600 4472 0 3805 (radio 0.86% / 0.92% tx 0.19% / 0.24% listen 0.66% / 0.68%)
100661	ID:4	12804 P 4.0 4 141772 3135053 6373 21870 0 19061 29438 625914 1593 4488 0 3797 (radio 0.86% / 0.92% tx 0.19% / 0.24% listen 0.66% / 0.68%)
100698	ID:1	12804 P 1.0 0 133511 3143361 6397 21718 0 19226 133511 3143361 6397 21718 0 19226 (radio 0.85% / 0.85% tx 0.19% / 0.19% listen 0.66% / 0.66%)
101028	ID:5	12804 P 5.0 4 141975 3134844 6372 22340 0 19906 29524 625826 1591 4452 0 4058 (radio 0.87% / 0.92% tx 0.19% / 0.24% listen 0.68% / 0.67%)
101211	ID:3	12804 P 3.0 4 140874 3135953 6384 21966 0 19111 25144 630210 0 4443 0 3893 (radio 0.86% / 0.67% tx 0.19% / 0.00% listen 0.67% / 0.67%)
120549	ID:2	15364 P 2.0 5 170586 3761594 7976 26159 0 23518 29395 625961 1596 4456 0 3998 (radio 0.86% / 0.92% tx 0.20% / 0.24% listen 0.66% / 0.67%)
120563	ID:6	15364 P 6.0 5 170696 3761484 7973 26364 0 23217 29829 625522 1597 4525 0 3960 (radio 0.87% / 0.93% tx 0.20% / 0.24% listen 0.67% / 0.69%)
120661	ID:4	15364 P 4.0 5 171042 3761141 7967 26355 0 23012 29267 626088 1594 4485 0 3951 (radio 0.87% / 0.92% tx 0.20% / 0.24% listen 0.67% / 0.68%)
121110	ID:5	15374 P 5.0 5 175721 3759150 9557 26768 0 23941 33743 624306 1385 4428 0 4035 (radio 0.92% / 1.15% tx 0.24% / 0.48% listen 0.68% / 0.67%)
121212	ID:3	15364 P 3.0 5 174706 3757475 9559 26492 0 22932 33829 621522 1375 4526 0 3821 (radio 0.91% / 1.17% tx 0.24% / 0.48% listen 0.67% / 0.69%)
140549	ID:2	17924 P 2.0 6 199424 4388113 9098 30620 0 27529 28835 626519 1122 4461 0 4011 (radio 0.86% / 0.85% tx 0.19% / 0.17% listen 0.66% / 0.68%)
140563	ID:6	17924 P 6.0 6 199072 4387864 9108 30809 0 27014 28973 626380 1135 4445 0 3797 (radio 0.87% / 0.85% tx 0.19% / 0.17% listen 0.67% / 0.67%)
140661	ID:4	17924 P 4.0 6 200420 4387120 9556 30727 0 26809 29375 625979 1589 4372 0 3797 (radio 0.87% / 0.90% tx 0.20% / 0.24% listen 0.66% / 0.66%)
141119	ID:5	17935 P 5.0 6 205068 4385411 11146 31242 0 27953 29344 626261 1589 4474 0 4012 (radio 0.92% / 0.92% tx 0.24% / 0.24% listen 0.68% / 0.68%)
141212	ID:3	17924 P 3.0 6 203870 4383667 11155 30864 0 26766 29161 626192 1596 4372 0 3834 (radio 0.91% / 0.91% tx 0.24% / 0.24% listen 0.67% / 0.66%)
160549	ID:2	20484 P 2.0 7 229239 5013647 10691 35333 0 31750 29812 625534 1593 4713 0 4221 (radio 0.87% / 0.96% tx 0.20% / 0.24% listen 0.67% / 0.71%)
160564	ID:6	20484 P 6.0 7 229019 5013874 10701 35232 0 30831 29344 626010 1593 4423 0 3817 (radio 0.87% / 0.91% tx 0.20% / 0.24% listen 0.67% / 0.67%)
160661	ID:4	20484 P 4.0 7 230314 5013701 11155 35371 0 30637 30201 625661 1599 4444 0 3860 (radio 0.88% / 0.91% tx 0.20% / 0.24% listen 0.67% / 0.68%)

### ➤ Explication de l'affichage de l'énergie :

En se référant à la figure si déçu, nous allons tracer le diagramme des CPU (processeur), LPM (mode de puissance inférieure), TX (Transmission de données), RX (Réception des données) sur une période donnée en fonction des itérations sachant qu'une itération s'exécute dans une période de 20 secondes.

Pour connaître l'énergie accumuler entre  $t$  et  $t+1$  du CPU, on soustrait de la valeur du cpu à l'instant  $t+1$ , la valeur du cpu à l'instant  $t$ . On fait de la même manière pour le LPM, RX et TX.

On a plusieurs méthodes pour tracer les diagrammes d'énergie. On peut utiliser [Gnuplot](#) ou [microsoft excel](#) en suivant cette [methodologie](#).

Nous allons adopter une autre méthode plus simple grâce à la bibliothèque **collect-view.h** disponible en cooja.

N'oubliez pas d'ajouter les lignes de codes suivants juste après **powertrace\_start(CLOCK\_SECOND\*20)**, si vous choisissez cette option :

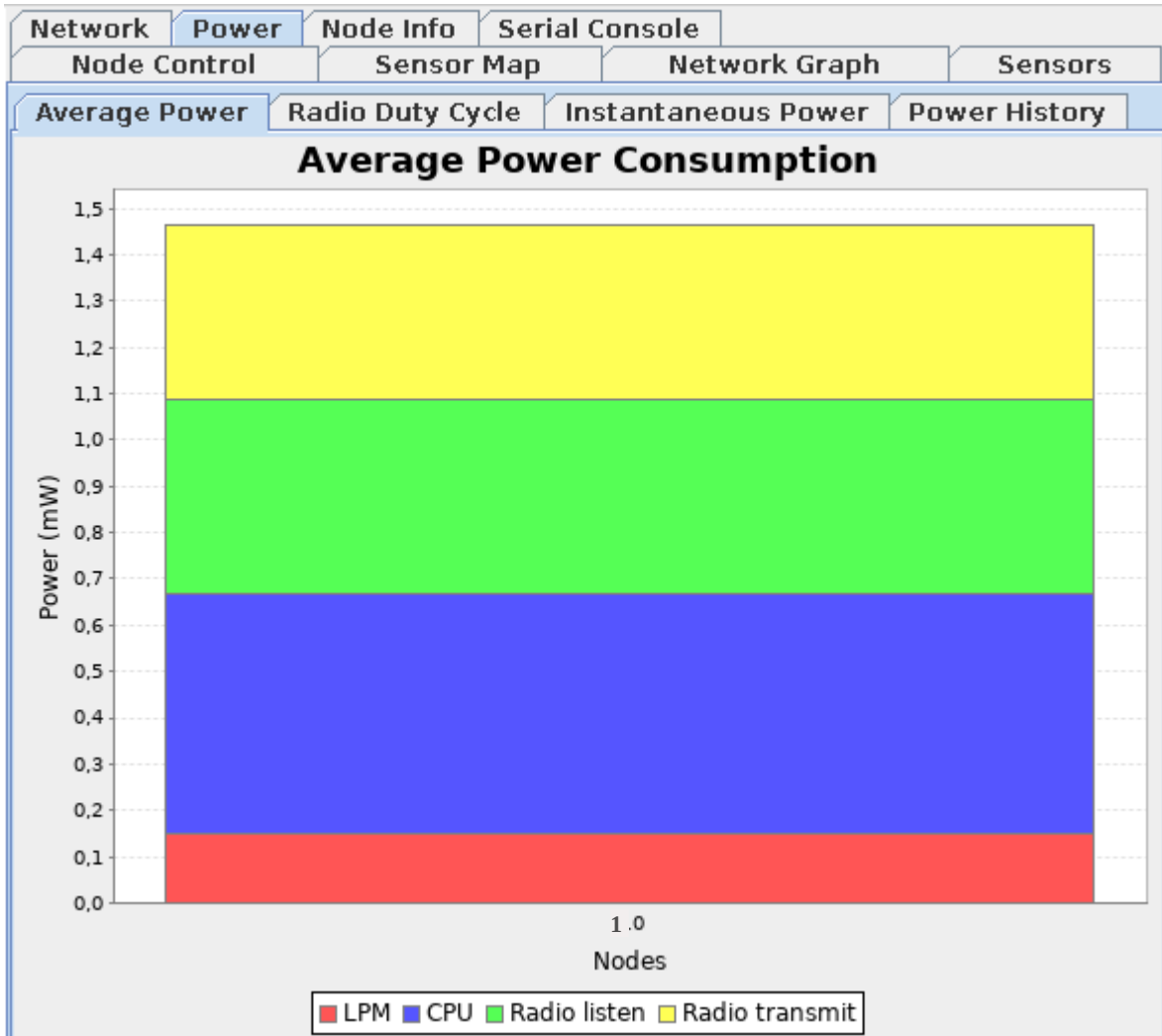
```
serial_shell_init();
shell_reboot_init();
shell_rime_init();
shell_rime_netcmd_init();
shell_powertrace_init();
shell_text_init();
shell_time_init();
shell_collect_view_init();
```

Pour générer le diagramme des énergies avec dans ce cas, cliquez sur le menu **tools/collect view** puis choisissez le mote sur lequel vous souhaitez mesurer l'énergie. Un menu s'ouvrira, cliquez sur **send command to nodes** puis sur **start Collect** et enfin cliquez sur l'onglet **Power**.

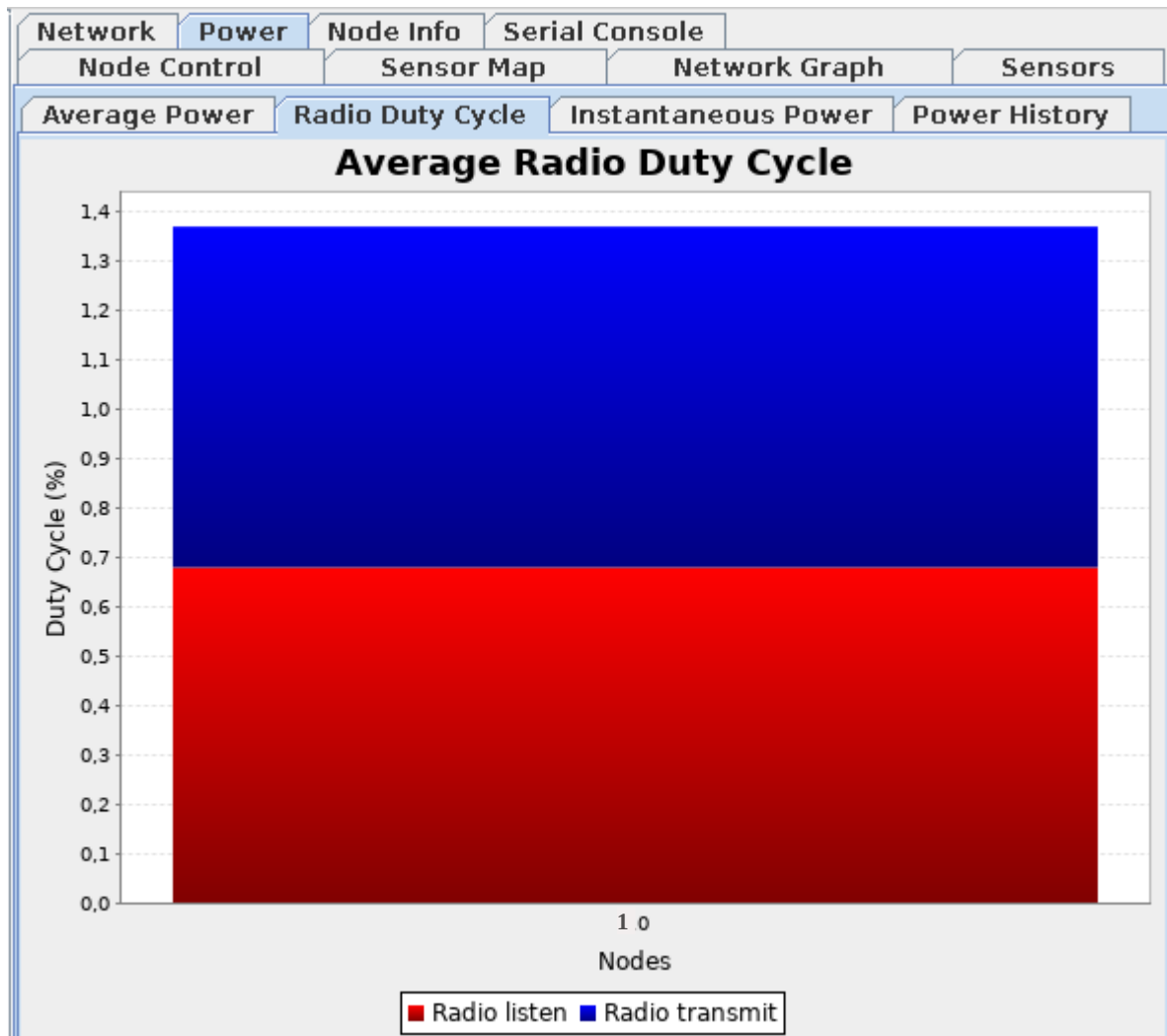
➤ **Diagramme des énergies :**

- **Emetteur (Mote 1) :**

**Consommation moyenne d'énergie :**



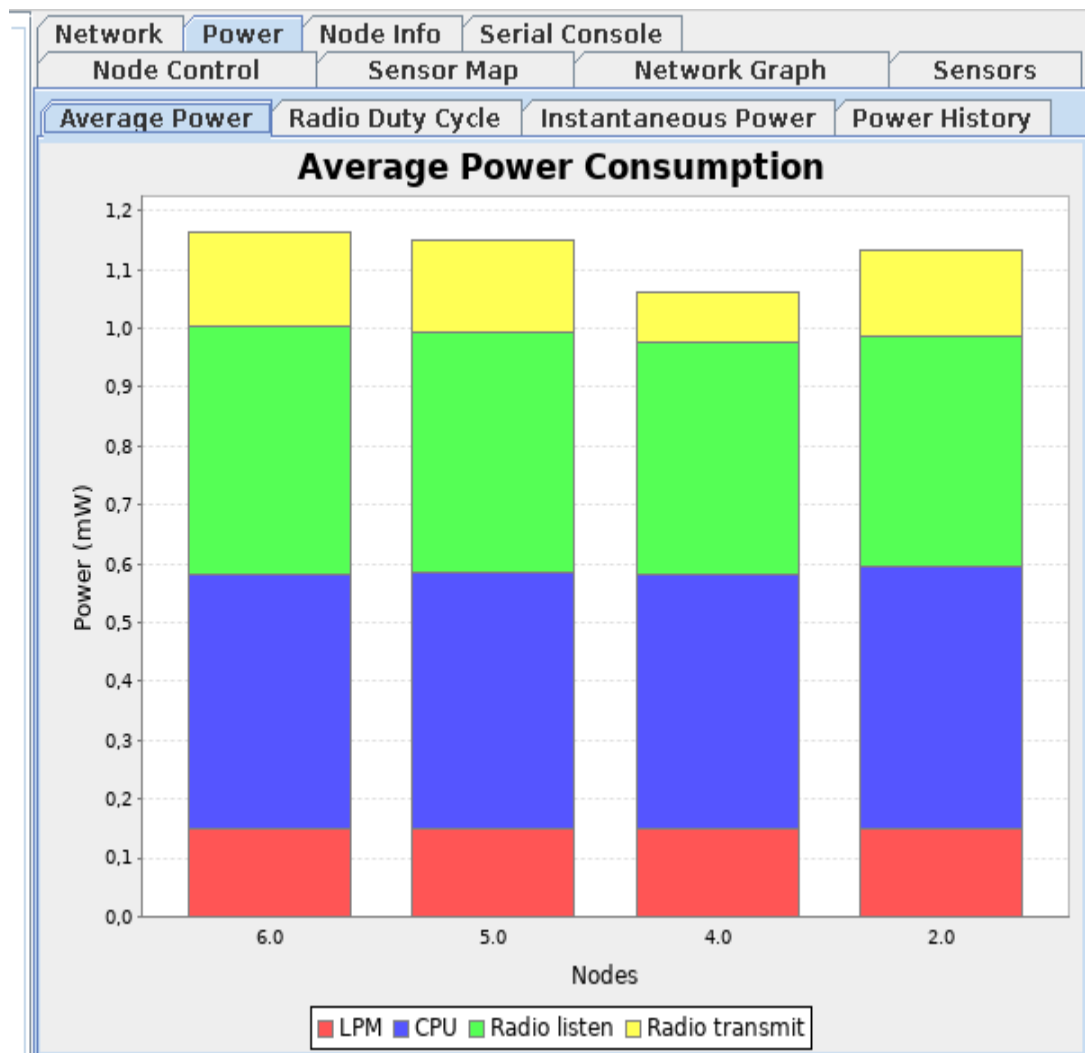
## Cycle moyen de service radio (RX et TX)



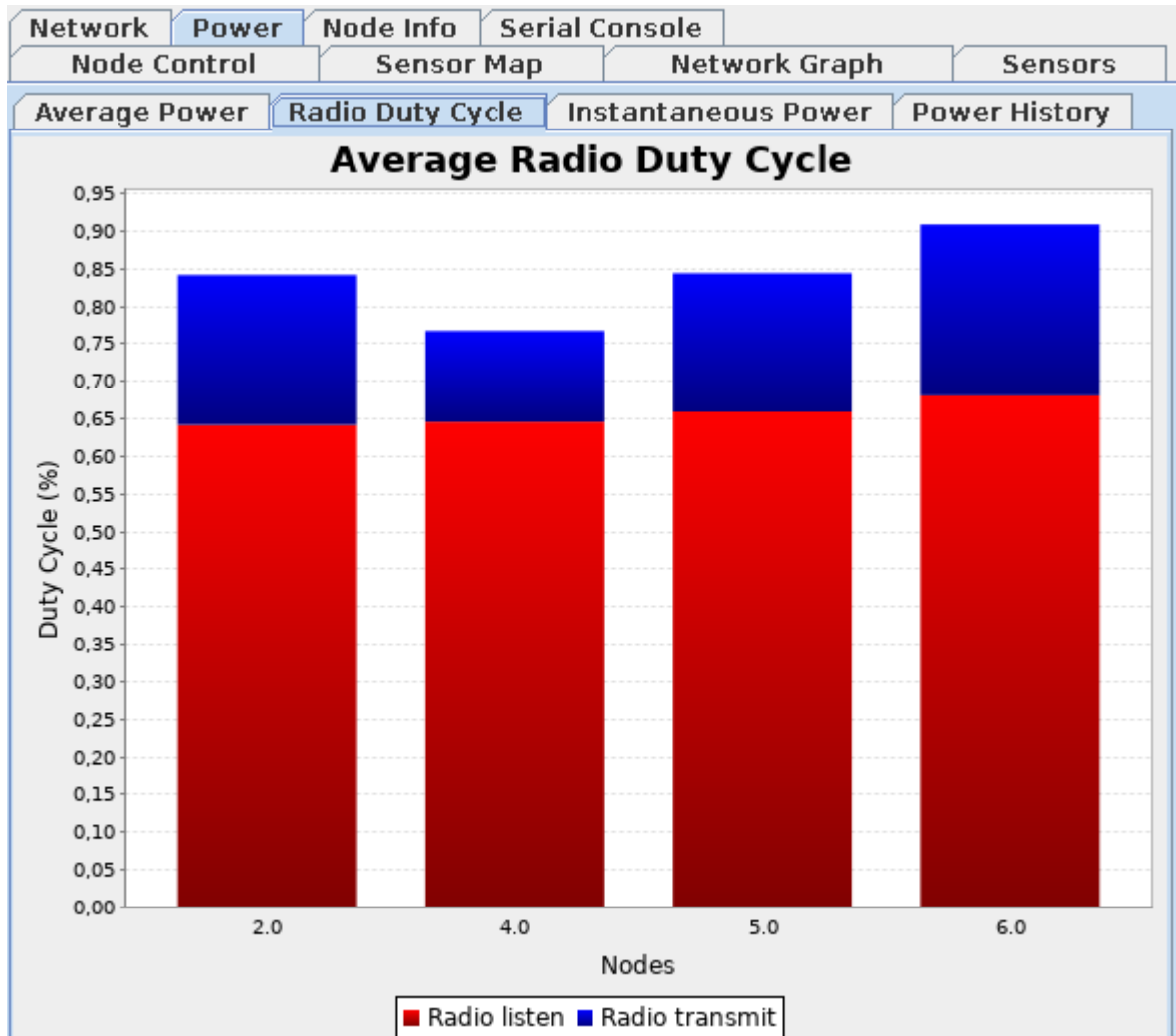


- **Recepteurs (Mote 2,3,4,5,6) :**

**Consommation moyenne d'énergie :**



## Cycle moyen de service radio (RX et TX)



### 13. Conclusion :

Ce tutoriel, permet de découvrir une plateforme de programmation adéquate qui est Contiki (programmé en C) avec le simulateur Cooja. Nous avons guider le lecteur dans un exercice de simulation des capteurs sans fil.

Nous avons travaillé dans un premier temps avec l'exemple de Hello World pour nous familiariser avec son code source, Broadcast pour la Diffusion d'un message à travers le réseau et powertrace pour tester la consommation d'énergie des nœuds.

Le code source et les fichiers complémentaires se trouvent sur : [la forge](#).

### 14. Références :

[1] G. De Sousa, « Etude en vue de la réalisation de logiciels bas niveau dédiés aux réseaux de capteurs sans fil : microsystème de fichiers », Thèse de Doctorat, Département informatique, Université Blaise Pascal-Clermont II, Octobre 2008.

[2] FEKIH Kawther et GORINE Asma « [Mise en place d'une applicatiion d'agregation des donnees dans un réseau de capteurs sans fil sous la platform Contiki](#)», Mémoire de fin d'étude pour l'obtention du diplôme de master en Informatique, Département informatique, Université Abou Bakr Belkaid-Tlemcen.

[3] T S Pradeepkumar « Powertrace in Contik OS-Cooja », [video](#) sur youtube.

[4] Laden « Cours sur Gnuplot » , [tutoriel](#) sur wiki ubuntu-fr

[5] Claude Duvallet, « [Simuation d'un réseau de capteur sans fils avec Cooja](#) » , Cours IoT , UFR Sciences et Techniques, Université du Havre.