Simulation d'un réseau de capteurs sans fils avec Cooja

Sfax, 24-26 novembre 2017

Workshop Internet des Objets

Claude Duvallet

Université du Havre
UFR Sciences et Techniques
25 rue Philippe Lebon - BP 540
76058 LE HAVRE CEDEX
Claude.Duvallet@gmail.com
http://litis.univ-lehavre.fr/~duvallet/

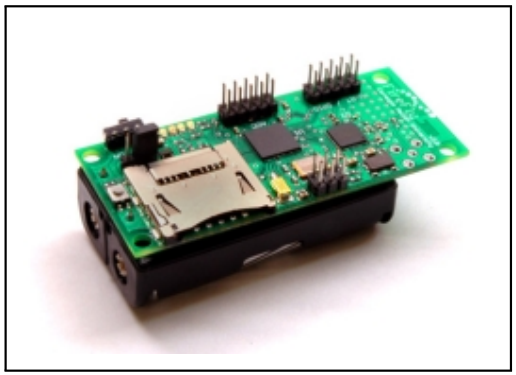
Plan de la présentation

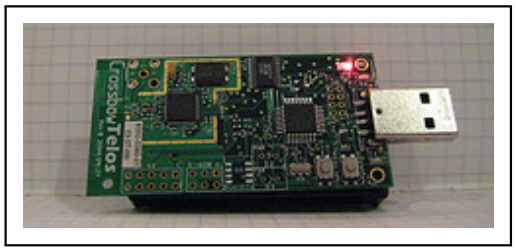
- Introduction et contexte
- Présentation des outils
- Installation et démarrage du simulateur
- Simulation d'un réseau de capteurs
- Création d'une simulation
- Modification d'une simulation

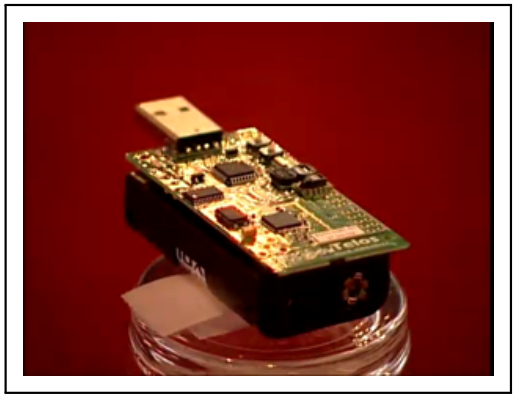
Les réseaux de capteurs sans fils

- Dans le domaine de l'Internet des Objets, il existe de nombreux objets connectés.
- Parmi ces objets connectés, nous allons considérer les capteurs intelligents.
- On peut considérer que les réseaux de capteurs sans fils constituent un sous-ensemble du concept plus large qu'est l'Internet des Objets.
- Nous allons considérer ici les réseaux de capteurs sans fils.

Les capteurs : caractéristiques

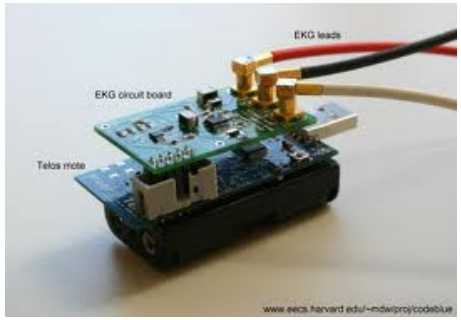


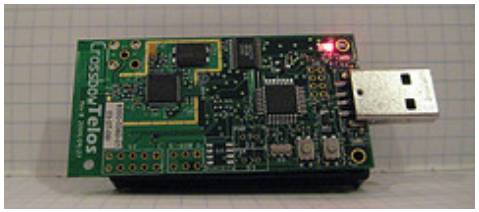


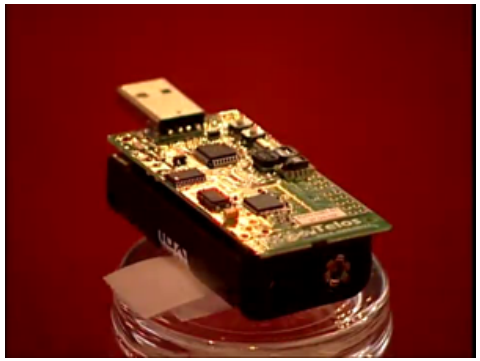


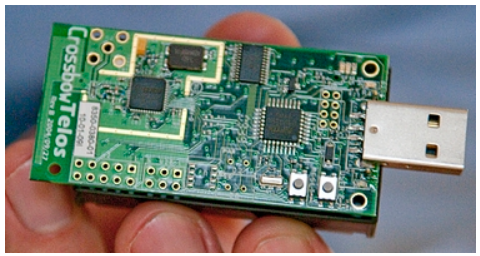
- Des dispositifs de taille réduite en général.
- Une très faible consommation énergétique mais des capacités énergétiques limités dans le temps.
- La possibilité de lire différents type de mesures physiques telles que :
  - la lumière, le son, l'humidité, la pression, la température, la composition du sol, de l'air ou de l'eau, etc.
- Géo-positionnement (données GPS).
- Dotés de faibles capacités de calcul et de stockage.
- Des capacités de communication sans fil : ZigBee (802.15.4), WiFi (802.11), etc.
- Différents types de capteurs (Fixes, Agiles ou Mobiles).
- Une grande facilité de déploiement.

Exemples de capteurs









Les capteurs : classification et architecture

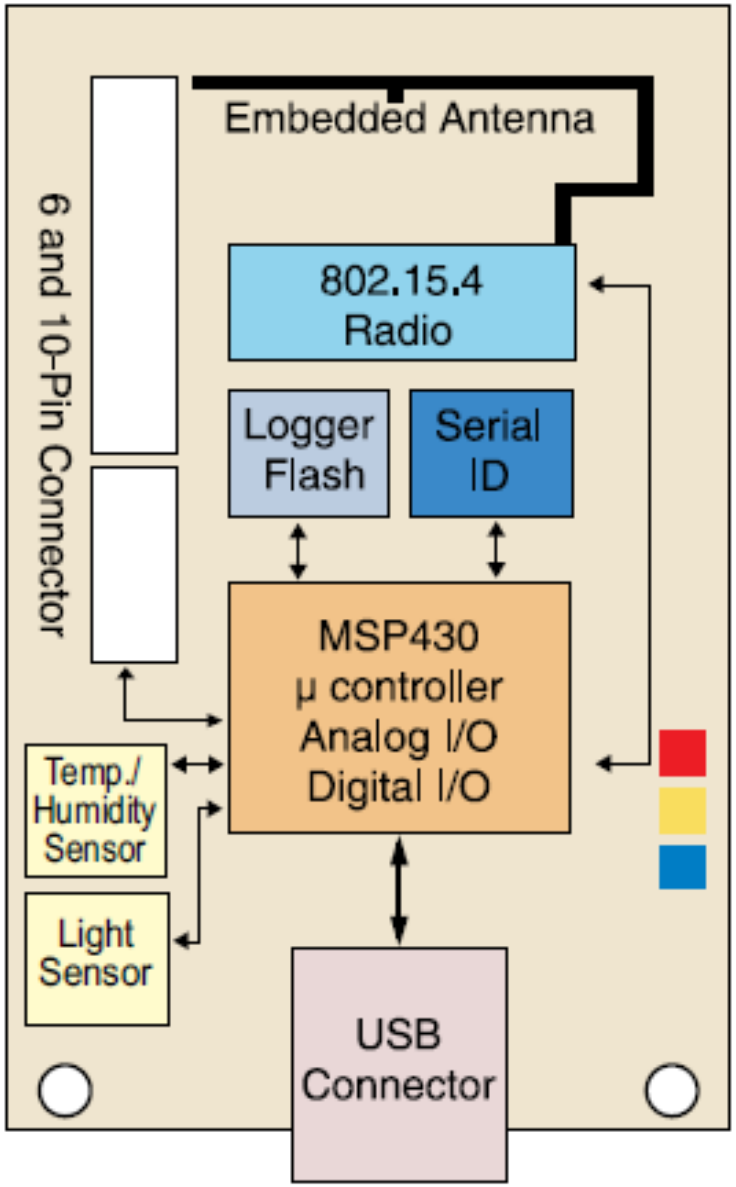
Classifications des capteurs :

- Des capteurs fixes :
  - Implantés définitivement dans un endroit.
  - Effectuant des mesures pour un endroit unique.
- Des capteurs agiles :
  - Variation discrète de la position.
- Des capteurs mobiles :
  - Positions variant dans le temps et dans l'espace.

Architecture d'un capteur sans fil

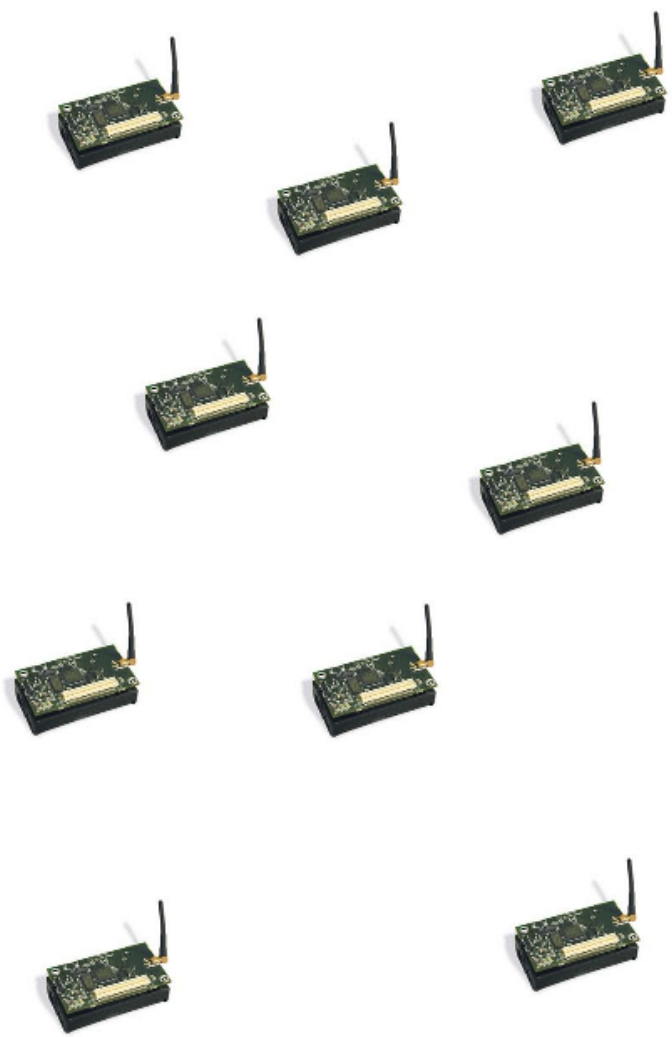
- Unité de traitement : processeur.
- Unité de stockage : mémoire "flash" ou mémoire vive (RAM).
- Unité d'énergie : batterie.
- Unité de détection : convertisseur analogique/numérique.
- Unité de transmission : antenne et récepteur.

Exemple de capteur : le capteur TelosB



- Processeur :
  - TI MSP430
  - 8 MHz
  - 10 ko RAM
- Transmission :
  - IEEE 802.15.4 (ZigBee)
  - Débit (250 kbps)
  - Bande de fréquence (2.4 à 2.4835 GHz)
  - Antenne intégrée
- Mémoire : 1 Mo
- Capteurs :
  - Température, Humidité, Lumière.
- Système d'exploitation : TinyOS 1.1.11 ou plus

Les capteurs : organisation en réseau



- Réseau Ad Hoc composés d'un ensemble de dispositifs (capteurs).
- Dispersés dans une zone géographique pour détecter les caractéristiques de l'environnement (température, pression, mouvement, ...).
- Capables de communiquer entre eux et avec une station de base.
- Implique des fonctionnalités de routage, parfois d'agrégation des données et de clustering...
- De nombreux domaines d'application.
- De nombreux problèmes de recherche.

Caractéristiques des réseaux de capteurs sans fil

- Passage à l'échelle :
  - De quelques centaines à plusieurs milliers de capteurs.
  - Avec ZigBee : environ 65 000 nœuds.
- Accès sans fil :
  - Sensibilité aux interférences radio.
  - Problème de portée et de débit .
- Ressources limitées :
  - Calcul (4 MHz).
  - Énergie (Piles électriques).
  - Mémoire (512 Ko - 1 Mo)
- Gestion de l'énergie :
  - Alimentation par batterie.
  - Impossibilité de changer la batterie.

  $\Longrightarrow$ Durée de vie limitée dans le temps.

Présentation du simulateur Cooja

- Cooja est un simulateur de réseaux de capteurs développé en Java.
- Il est fait pour émuler le fonctionnement de capteurs sous le système d'exploitation Cooja.
- Il s'agit d'un simulateur flexible et extensible permettant de modifier ou remplacer tous les niveaux du système.
- Il interagit avec Code Contiki au travers de JNI (Java Native Interface).
- L'interface du simulateur est composée de plusieurs plugins présent sous formes de fenêtres.

L'interface du simulateur Cooja (1/2)

Il existe 5 zones dans cette interface :

- La première zone porte le nom de "Network" :
  - Cette zone permet de visualiser chaque nœud du réseau et de visualiser leur état (identifiant, adresse, LED, etc.).
  - A l'initialisation de la simulation, cette zone est vide et il faut lui ajouter des nœuds.

- La seconde zone porte le nom de "Timeline" :
  - Il s'agit d'une frise chronologique dans laquelle chaque les messages et les événements sont affichés.
  - Elle permet par exemple de visualiser les envois de données entre les nœuds.
  - Les messages à afficher doit être indiqué dans le code du nœud.

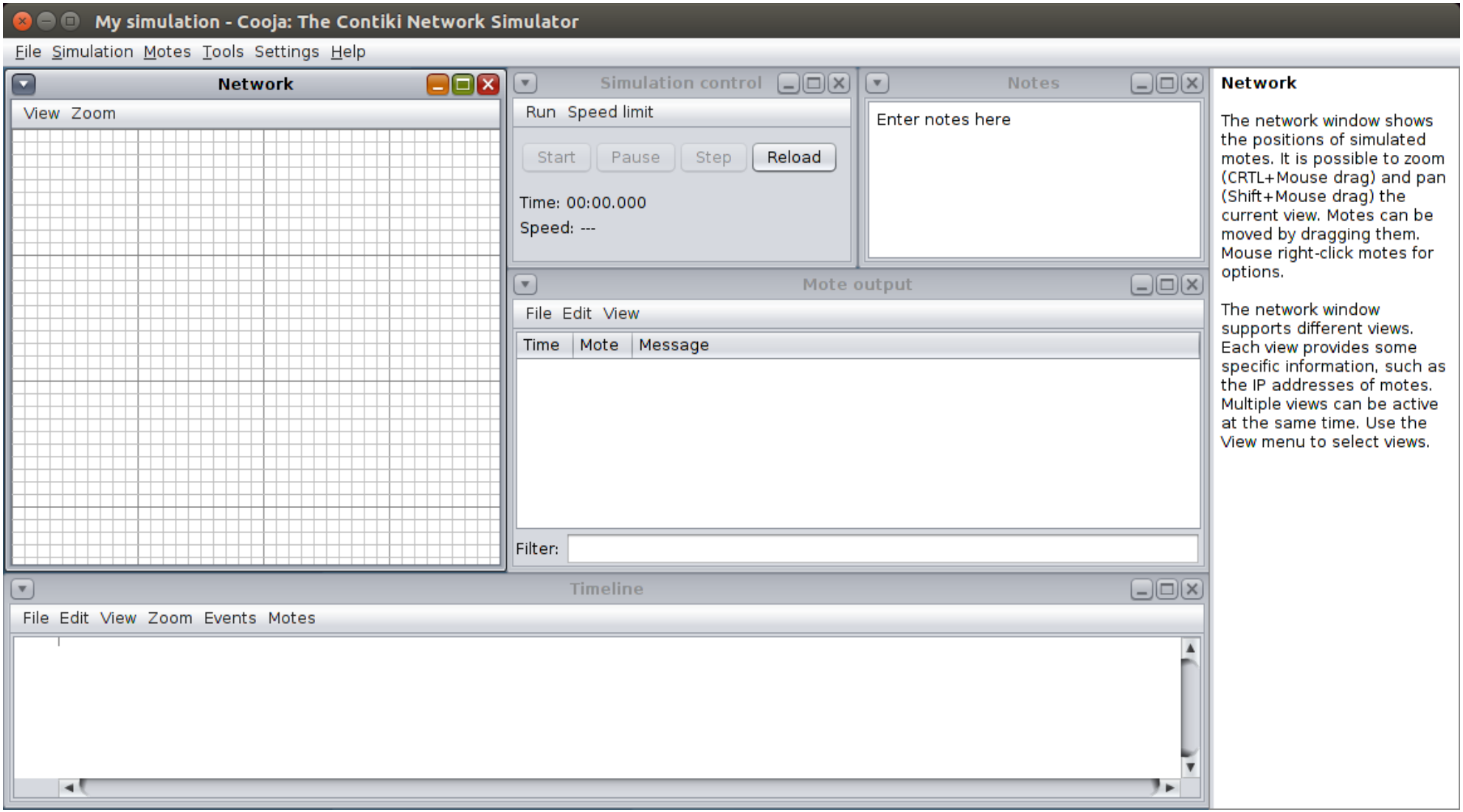
L'interface du simulateur Cooja (2/2)

- La troisième zone porte le nom de "Mote Output" :
  - Cette zone permet d'afficher toutes les sorties des différentes interfaces des nœuds.
  - On peut disposer d'une fenêtre "Mode Output" différente pour chaque nœud.

- La quatrième zone porte le nom de "Control" :
  - Cette zone est utilisée pour contrôler la simulation : démarrer, recharger, ou exécuter pas à pas.
  - Le temps d'exécution ainsi que la vitesse y sont affichés.

- La cinquième zone porte le nom de "Notes" :
  - Il s'agit d'un espace permettant de prendre des notes sur la simulation en cours, dans l'optique de l'enregistrement de la simulation.

CAPTURE D'ÉCRAN DE L'INTERFACE DU SIMULATEUR



Les différents outils nécessaires

Nous allons travailler sous Ubuntu.

- 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
    ```

Les différents outils nécessaires

- 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 Developpement Kit) pour pouvoir l'utiliser.

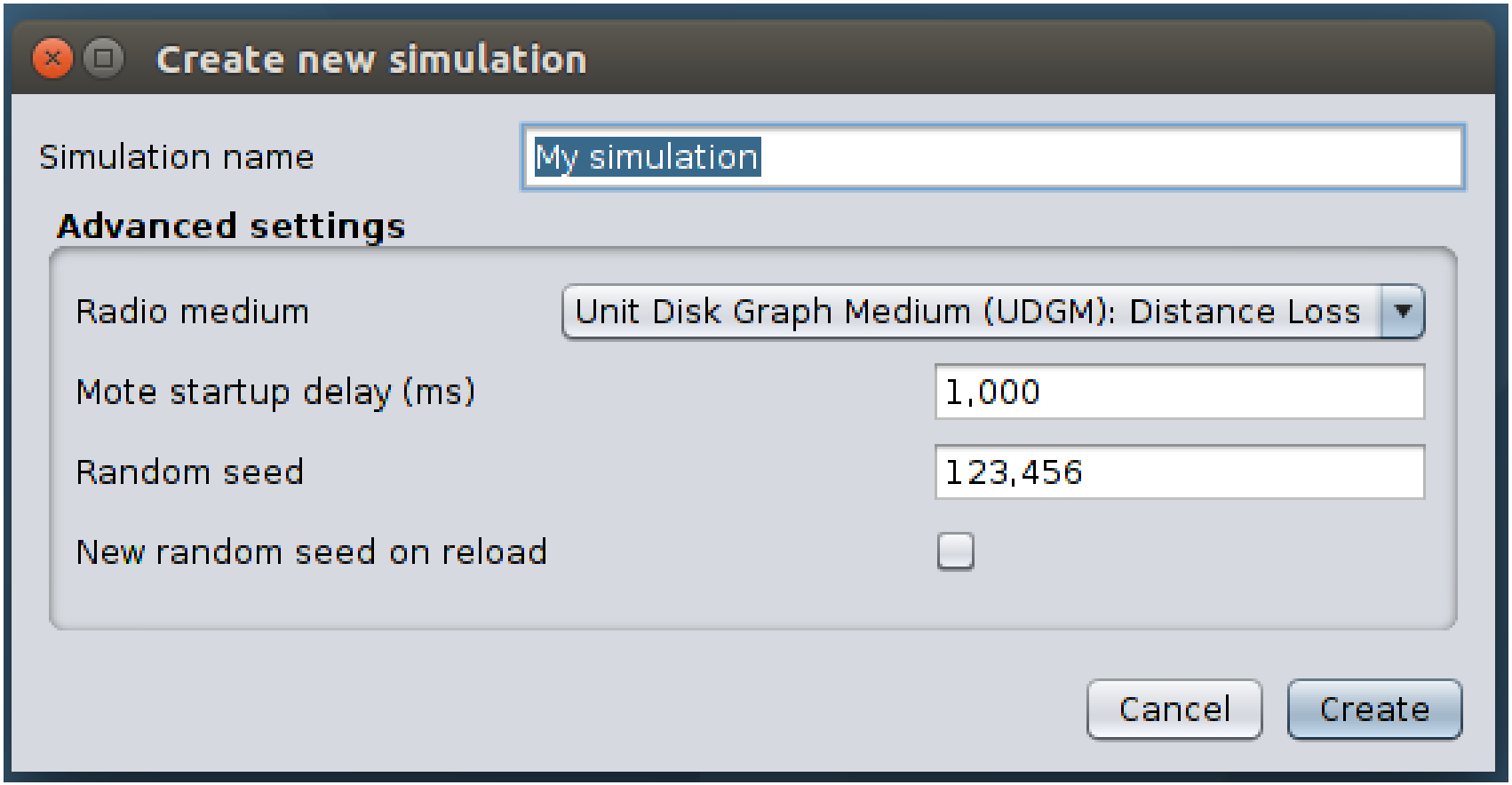
    ```
    sudo apt-get install openjdk-8-jdk-headless
    ```

- 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
    ```

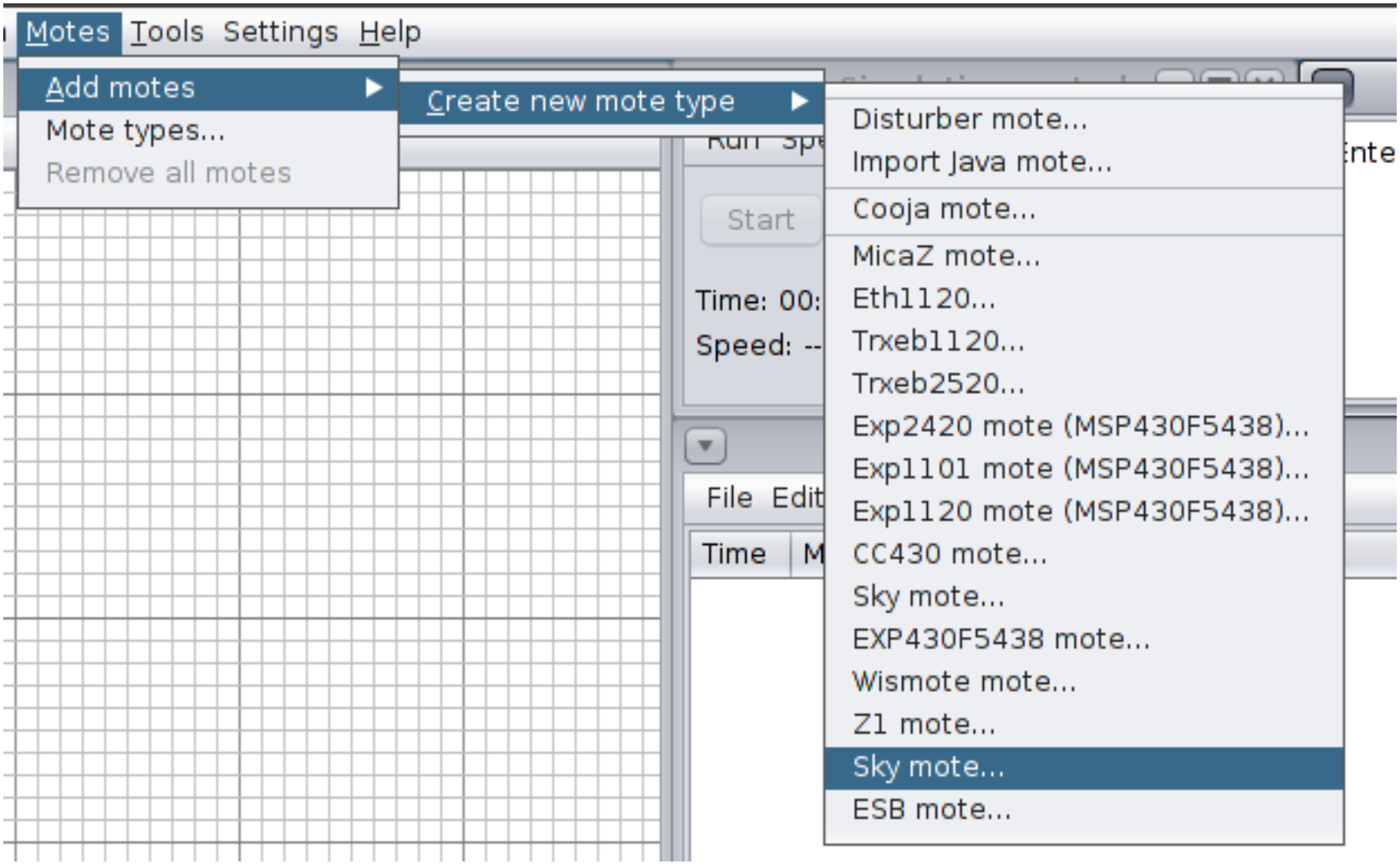
Démarrage de l'application

- Pour démarrer l'application, il suffit de lancer la commande suivante : `ant run`
- Puis, on va créer une nouvelle simulation à partir du menu `File` / `New simulation`



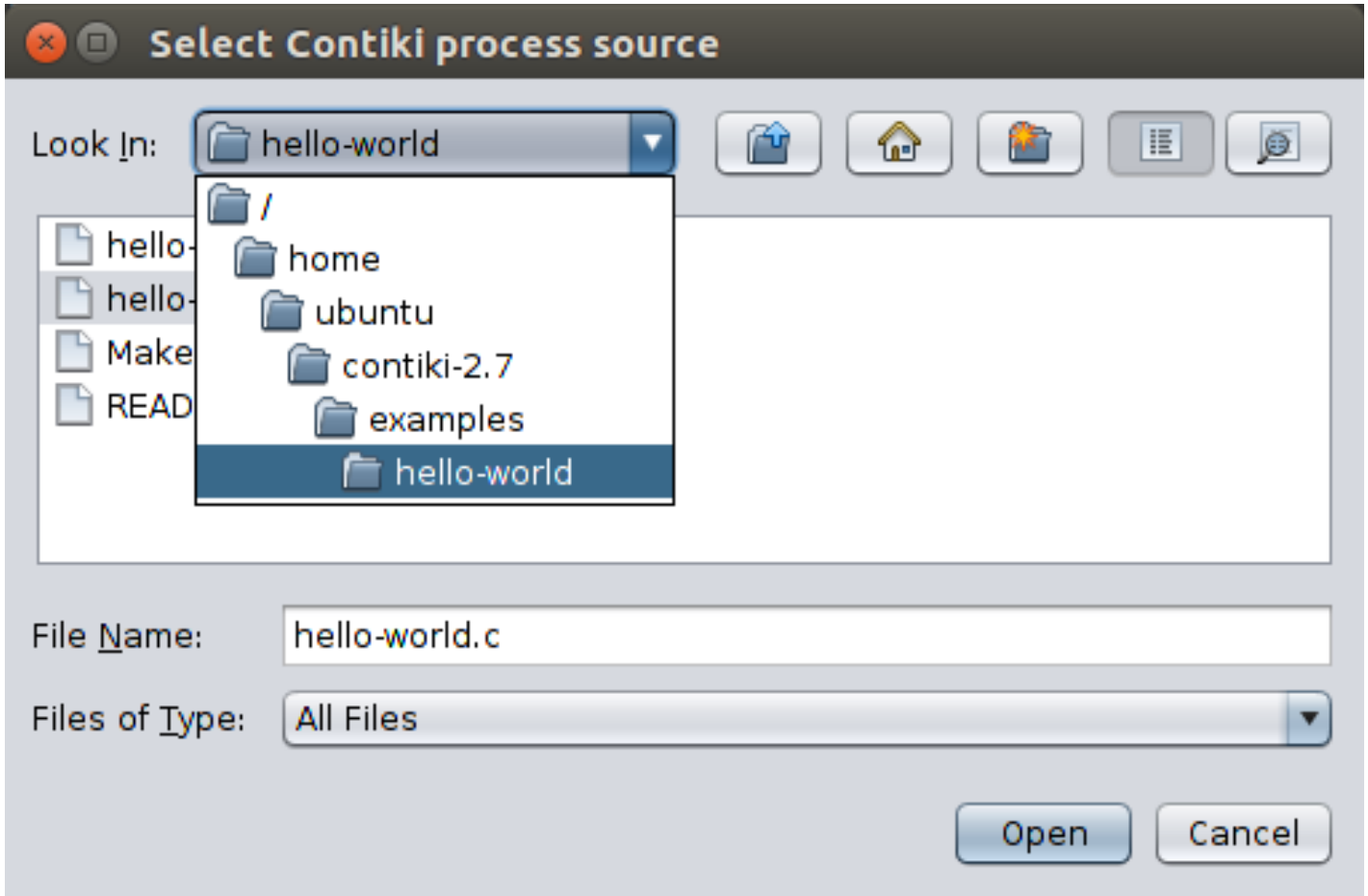
Ajout de nœuds

- Il faut ajouter des nœuds pour créer un réseau de capteurs.
- Nous choisissons le type de nœud "Sky mote" :



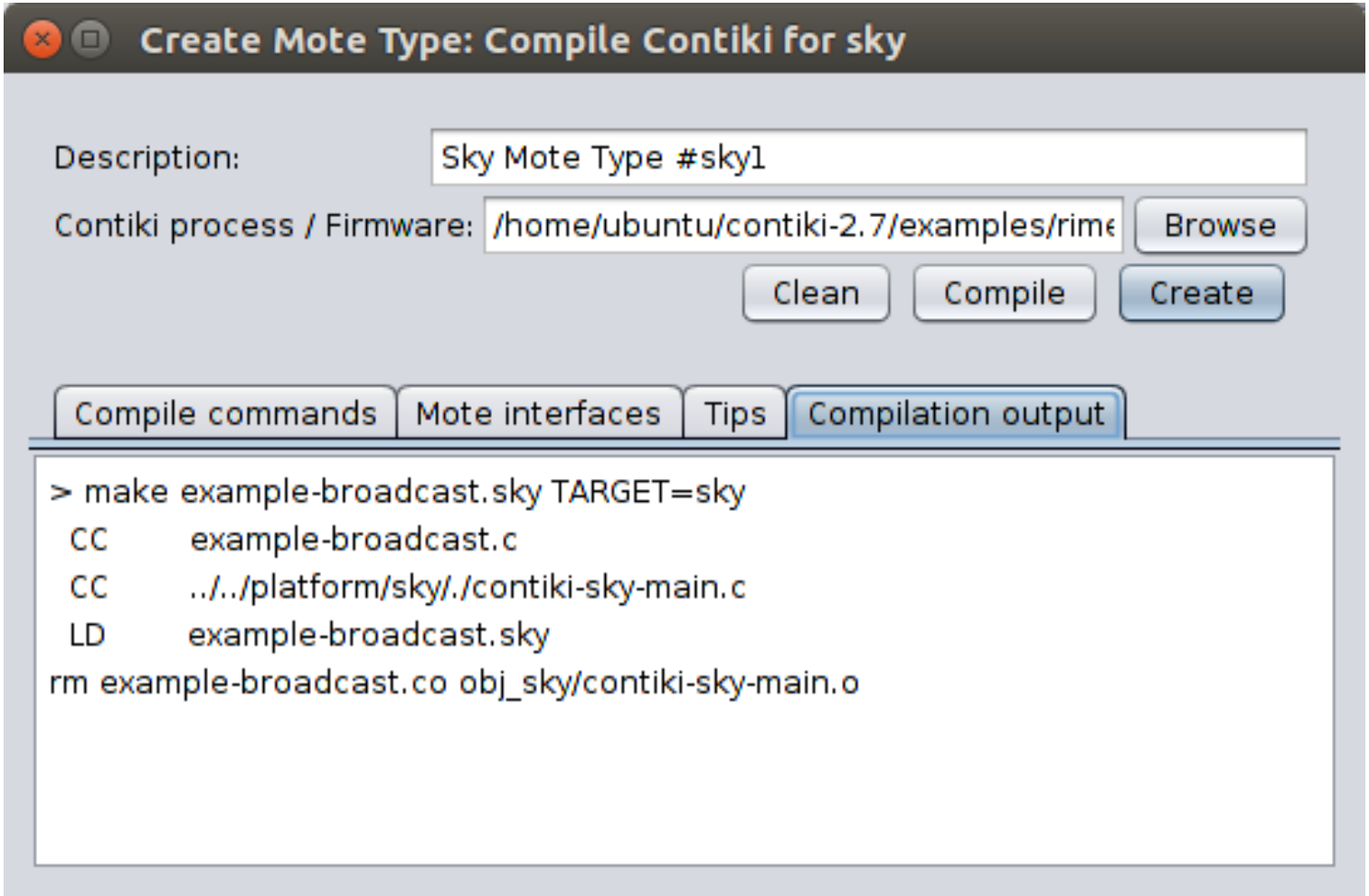
Ajout de nœuds

- Nous allons commencer par un simple "Hello World".
- Nous chargeons donc le fichier `hello-world.c` qui se trouve dans le répertoire `examples\hello-world`



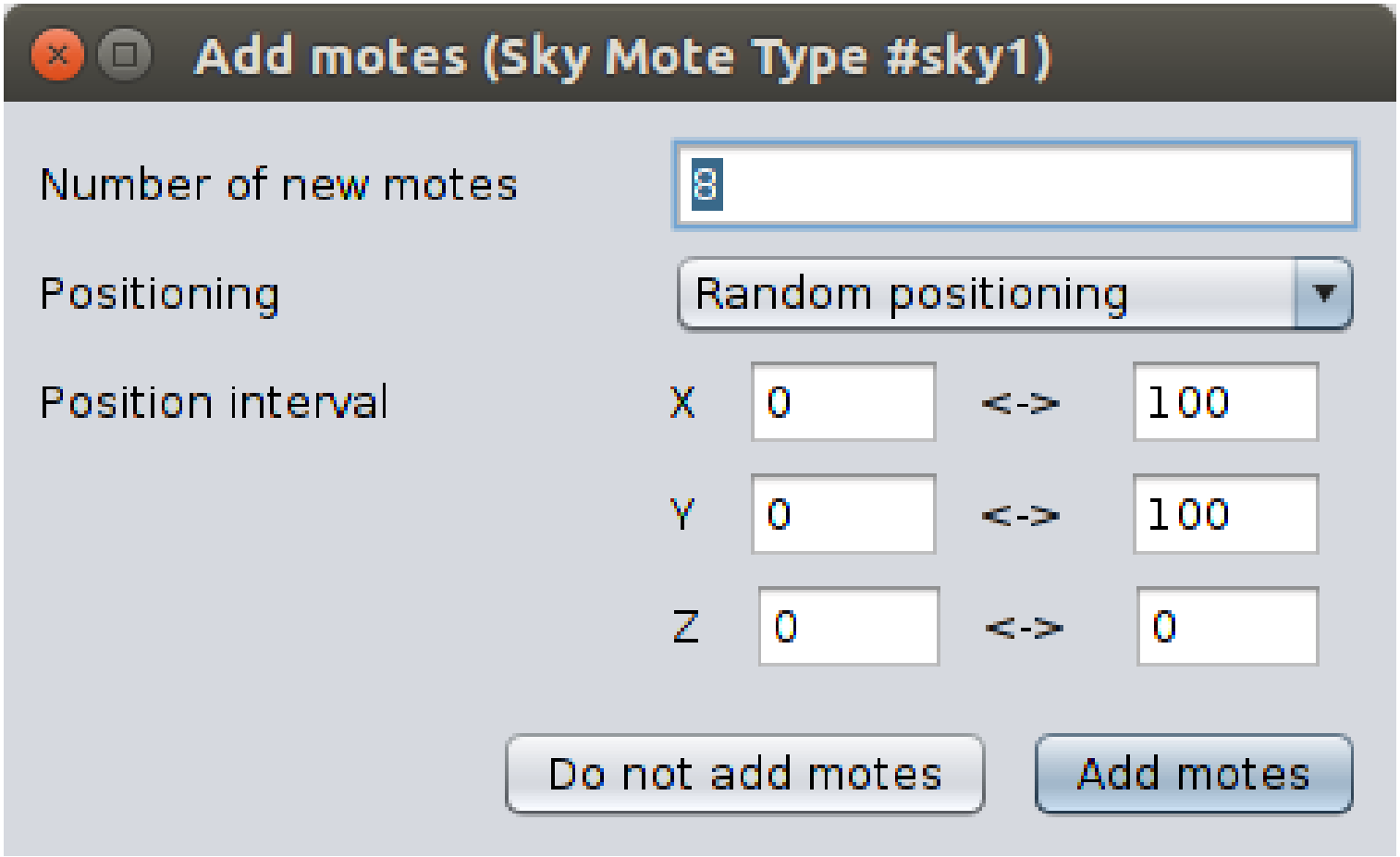
Ajout de nœuds

- Il faut ensuite compiler le code.
- Lorsque le code est compilé, on peut créer des nœuds.



Ajout de nœuds

- On doit ensuite indiquer le nombre de nœuds avec un placement aléatoire.



- Ensuite, on peut démarrer la simulation.

Code du Hello World

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

Modification du Hello World

- On va modifier le code afin de faire clignoter les LED.

```
/*---------------------------------------------------------------
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);//, &blink_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++;
```

Modification du Hello World

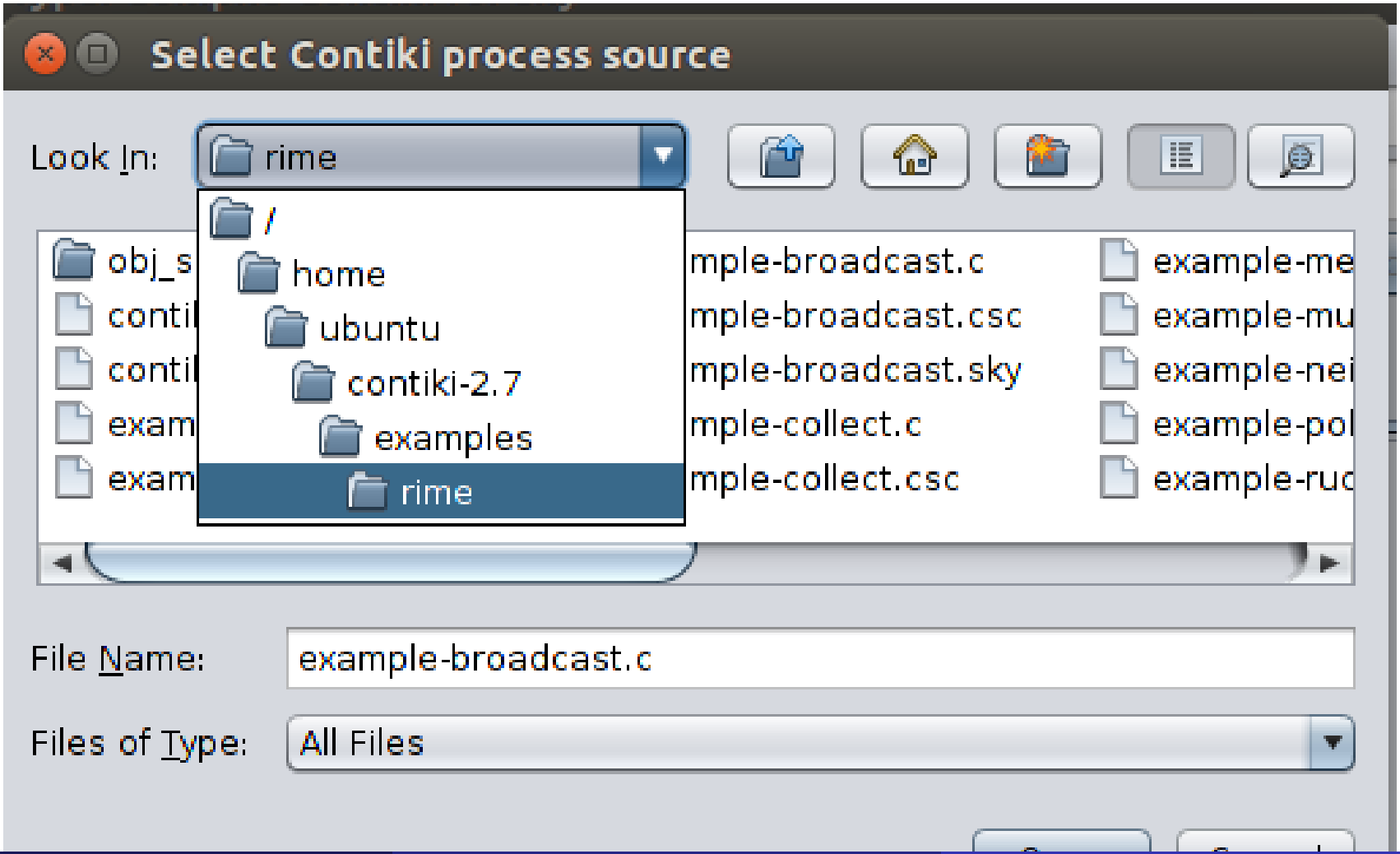
- Suite des modifications :

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

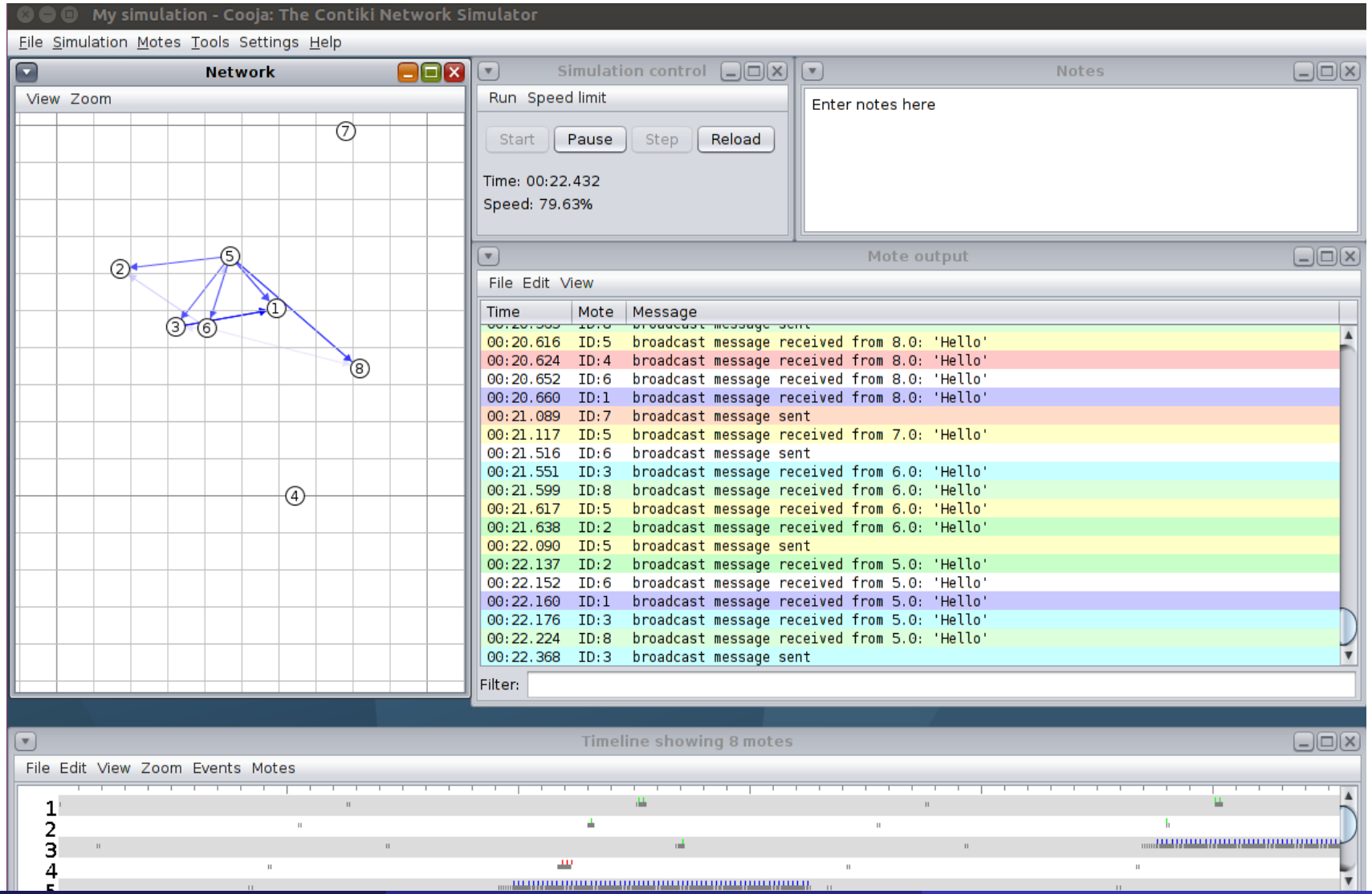
DIFFUSION D'UN MESSAGE À TRAVERS LE RÉSEAU

- On va prendre 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"



DIFFUSION D'UN MESSAGE À TRAVERS LE RÉSEAU

- Après le démarrage de la simulation, on va pouvoir constater les échanges de messages.



MODIFICATION DU CODE DE example-broadcast

- On va juste passer les messages en français.

```
/*---------------------------------------------------------------------
PROCESS(example_broadcast_process, "[Exemple2] Exemple de diffusion");
AUTOSTART_PROCESSES(&example_broadcast_process);

/*---------------------------------------------------------------------
static void broadcast_recv(struct broadcast_conn *c, const rimeaddr_t *from)
{
  printf("[Exemple2] Message recu depuis %d.%d: '%s'\n",
         from->u8[0], from->u8[1], (char *)packetbuf_dataptr());
}
static const struct broadcast_callbacks broadcast_call = {broadcast_recv};
static struct broadcast_conn broadcast;

/*---------------------------------------------------------------------
PROCESS_THREAD(example_broadcast_process, ev, data)
{
  static struct etimer et;
  static int cpt=0;

  PROCESS_EXITHANDLER(broadcast_close(&broadcast);)
  PROCESS_BEGIN();
```

MODIFICATION DU CODE DE example-broadcast.c

- Suite des modifications :

```
  broadcast_open(&broadcast, 129, &broadcast_call);

  while(1) {
    cpt++;
    /* Delay 2-4 seconds */
    etimer_set(&et, CLOCK_SECOND * 8 + random_rand() % (CLOCK_SECOND * 8));

    PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&et));

    packetbuf_copyfrom("Bonjour", 7);
    broadcast_send(&broadcast);
    printf("[Exemple2] Message envoye en diffusion %d\n",cpt);
  }

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

Mise en pratique

- Exercice :
  - En combinant les deux exemples précédents, vous allez essayer de créer une application basée sur deux types de capteurs.
  - Le premier type de capteur diffusera un message de changement de couleur de LED en mode diffusion. Ce message devra comporté 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).

Mise en pratique

- Exemple de mise en œuvre :