IT-SDN(v0.4.1): Developer Guide (for Linux 64 bits – March, 2019)

Renan C. A. Alves[1], **Doriedson A. G. Oliveira**[1], **Gustavo N. Segura**[1], **Cintia B. Margi**[1] *

[1]Escola Politécnica – Universidade de São Paulo – São Paulo, Brazil

Contents

*C. B. Margi is supported by CNPq research fellowship #307304/2015-9.

1. Introduction

IT-SDN is a Software Defined Wireless Sensor Network (SDWSN) tool that is completely open and freely available, designed to be independent of the operating system and its functions. Although it is inspired by TinySDN [de Oliveira et al. 2014, de Oliveira and Margi 2016], we improved the architecture, protocols and implementation.

The IT-SDN framework version presented here comprises SDN enabled-nodes developed under Contiki OS and a controller developed in C and C++ with Qt.

This manual assumes that you have configured IT-SDN development environment and you can compile the enabled-node, controller-node and controller-pc. If not, you should read the installation guide before[1].

2. Getting Started

Let's start developing a simple program, known as "Hello World!", which a node sends the message "Hello World!" to another node that receives and prints the message. The same code is used for the node that generates message and the node that receives message, using the node address to sort the difference.

2.1. Creating a new program

To create the program, copy the file `enabled-node.c` to `enabled-node-hello-world.c` in the folder `it-sdn/applications` and open in your favorite editor.

Now, replace the function:

```
PROCESS_THREAD(sdn_test_process, ev, data) {
...
}
```

with the following clean function:

```
PROCESS_THREAD(sdn_test_process, ev, data) {

  PROCESS_BEGIN();

  sdn_init(receiver);

  while(1) {

    PROCESS_WAIT_EVENT();
  }

  PROCESS_END();
}
```

This function is equivalent to `int main()` function when programming in C. The call to `sdn_init(receiver)` function is necessary to inform IT-SDN a callback function to process incoming messages on the node.

[1]Source code and installation manual are available at `http://www.larc.usp.br/~cbmargi/it-sdn`

2.2. Programming node to receive messages

This piece of code informs the controller that the node wants to receive data. IT-SDN sends and receives data through a flow identified by one id (in this case, id 2018). Since the same code is used for transmitter and receiver, we use an `if` statement to distinguish which node informs the controller that it owns the flow.

Type the following code before `while(1)`:

```
if (sdn_node_addr.u8[0] == 2) {

    sdn_register_flowid(2018);
}
```

The variable `sdn_node_addr` maintains the SDN node address. It can have different sizes according to the implementation. You can observe the current SDN address size by reading the constant `SDNADDR_SIZE`, whose default value is the same of `LINKADDR_SIZE` implemented by CONTIKI.

As you can see, the `sdn_register_flowid(2018)` function configures the flow id 2018 with target to the node address with last byte 2.

Function `receive()` already has one `printf()` statement to print the incoming message, therefore the node with final address 2 is ready to receive messages from flow id 2018.

2.3. Programming node to send messages

In this section we program a node with final address 3 to send data to flow id 2018.

In the same file `enabled-node-hello-world.c` type the following code before the `while(1)`:

```
static struct etimer periodic_timer;

etimer_set(&periodic_timer, 30 * CLOCK_SECOND);
```

And, replace the line `PROCESS_WAIT_EVENT();` with the following code:

```
PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));

etimer_restart(&periodic_timer);

if (sdn_node_addr.u8[0] == 3) {

  char *data = "Hello World!";

  sdn_send((uint8_t*) data, strlen(data), 2018);
}
```

Here we implemented a timer to send a message every 30 seconds and use `sdn_send((uint8_t*) data, data_len, 2018)` to send a message `Hello World!` to flow id 2018. Below you can check the send function signature:

```
void sdn_send(uint8_t *data, uint16_t len, flowid_t flowid);
```

The node with final address 3 is ready to send messages.

3. Complete code

Here you can see the complete code:

```
PROCESS_THREAD(sdn_test_process, ev, data) {

  PROCESS_BEGIN();

  sdn_init(receiver);

  static struct etimer periodic_timer;

  etimer_set(&periodic_timer, 30 * CLOCK_SECOND);

  if (sdn_node_addr.u8[0] == 2) {

       sdn_register_flowid(2018);
  }

  while(1) {

     PROCESS_WAIT_EVENT_UNTIL(etimer_expired(&periodic_timer));

     etimer_restart(&periodic_timer);

     if (sdn_node_addr.u8[0] == 3) {

        char *data = "Hello World!";

        sdn_send((uint8_t*) data, strlen(data), 2018);
     }
  }
  PROCESS_END();
}
```

4. Compiling

4.1. enabled-node

To compile enabled-node copy the `Makefile_enabled_node` to `Makefile_enabled_node_hello_world` and changes the line `all: enabled-node...` to `all: enabled-node-hello-world`.
Compile the enabled-node with the following commands:

```
applications/$ make clean -f Makefile_enabled_node_hello_world
applications/$ make -f Makefile_enabled_node_hello_world
```

4.2. controller-node

To compile the controller-node run the following commands:

```
/applications/$ make clean -f Makefile_controller_node
/applications/$ make -f Makefile_controller_node
```

5. Running

5.1. Creating a new simulation

Let's create a new simulation using Cooja Simulator. To open Cooja simulator execute command `ant run` on the contiki/tools/cooja/ folder). On Cooja simulator,

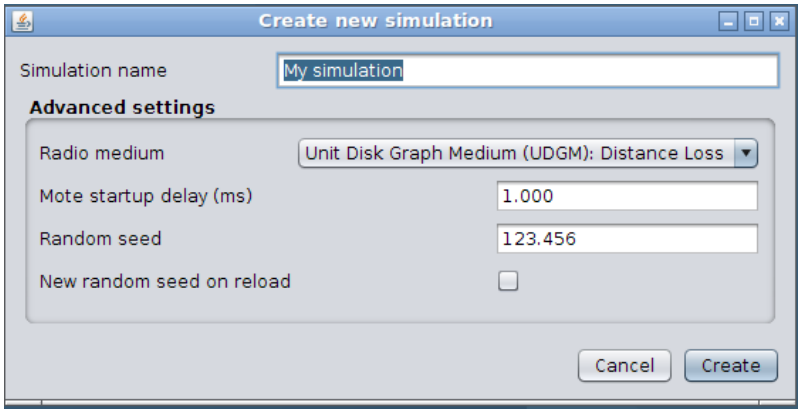

Figure 1. Create new simulation window in Cooja.

click in `File -> New Simulation...` menu and you can see the `Create new simulation` window as shown in Figure 1. Click [create] button.

Let's insert one controller-node to simulation. Click `Motes -> Add motes -> Create new mote type -> Sky mote ...` and you can see the `Create Mote Type: Compile Contiki for sky` window as shown in Figure 2

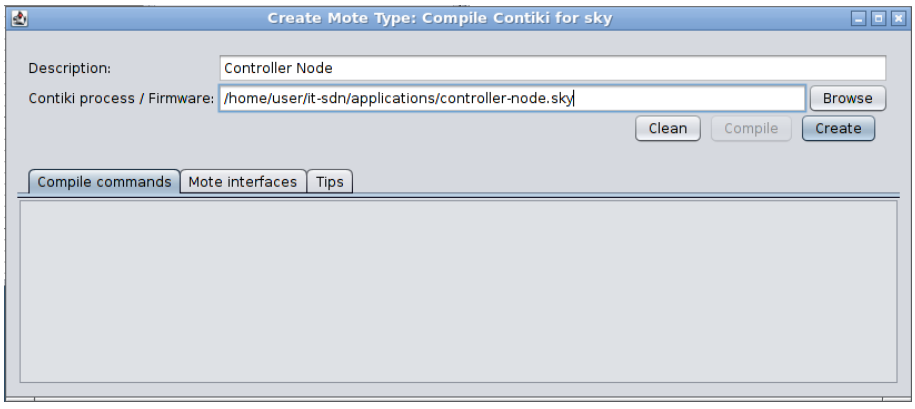

Figure 2. Create Mote Type: Compile Contiki for wismote window in Cooja.

Select `controller-node.sky` firmware by clicking on [Browse] button and navigating to your application folder. After, change description to `Controller Node` as shown in Figure 2. After click [Create] Button and you can see the `Add motes` window as shown in Figure 3.

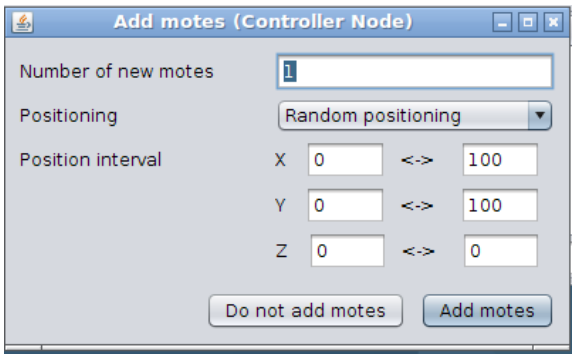

Figure 3. Add motes (Controller Node) window in Cooja.

Click [Add motes] button to add one controller-node and you can see a controller node in `Network` window as shown in Figure 4.

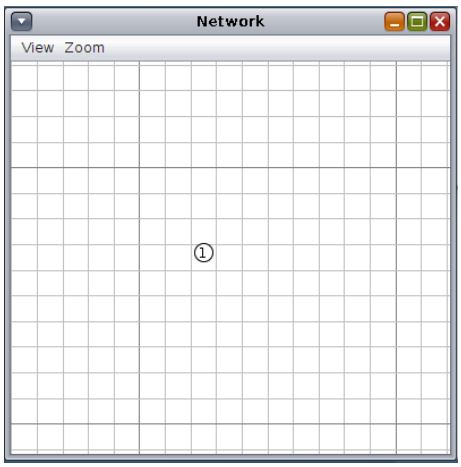

Figure 4. Cooja Simulator with a controller-node.

Now, let's insert two enabled-nodes to simulation, one to send message to flow id 2018 and another two receive messages from flow id 2018. Click `Motes -> Add motes -> Create new mote type -> Sky mote ...` and you can see the `Create Mote Type: Compile Contiki for sky` window as shown in Figure 5

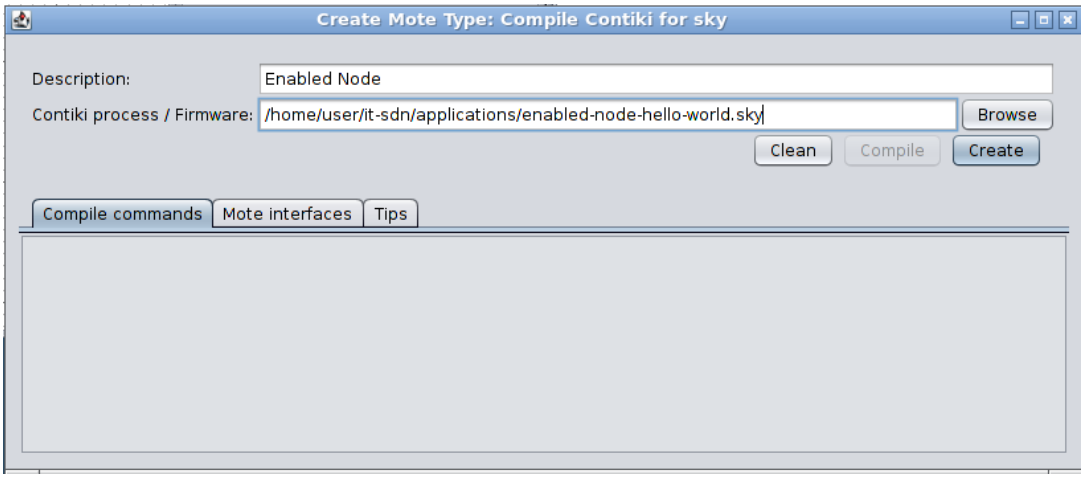

Figure 5. Create Mote Type: Compile Contiki for wismote window in Cooja.

Select `enabled-node-hello-world.sky` firmware by clicking on [Browse] button and navigating to your application folder. After, change description to `Enabled Node` as shown in Figure 5. After click [Create] Button and you can see the `Add motes` window as shown in Figure 6.

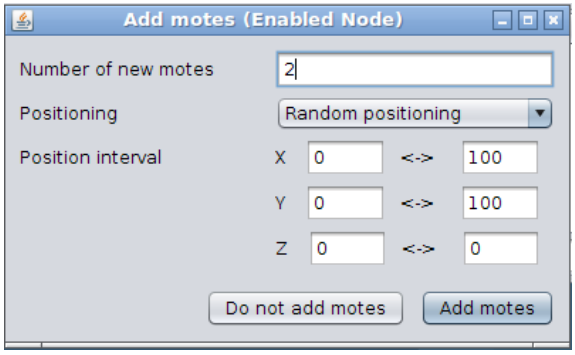

Figure 6. Add motes (Enabled Node) window in Cooja.

Change `Number of new motes` to 2 and click [Add mote] button to add two enabled-nodes and you can see three nodes in `Network` window as shown in Figure 7. The node of number 1 is the controller-node and the nodes of number 2 and 3 are enabled-nodes. Adjust the topology of nodes in line as shown in Figure 7.

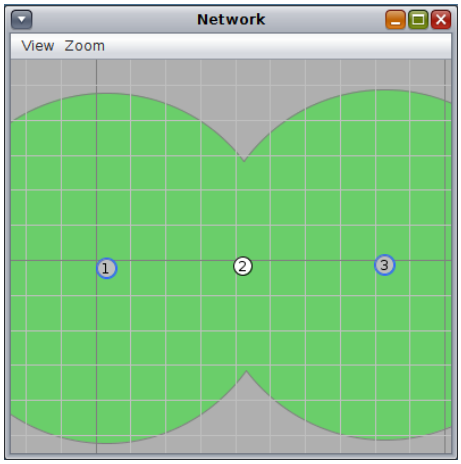

Figure 7. Cooja Simulator with a controller-node.

We need to open the serial socket server for node 1 to enable connection from controller-pc to controller-node. Click with the right mouse button over the node 1 and select the `Mote tools for Sky 1 -> Serial Socket (SERVER) ...` menu. Click [Start] button on `Serial Socket (SERVER) (Sky 1)` window as shown in Figure 8.

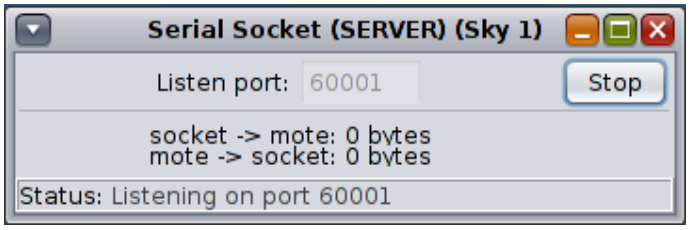

Figure 8. Serial Socket Server window from node 1.

5.2. Running controller-pc

Let's start controller-pc and connect to controller-node in Cooja.

Open the controller-pc project on Qt Creator, click on menu `Build -> Build Project "controller-pc"` or press keys `CTRL+B`. After, click on menu `Build -> Run` or press keys `CTRL+R` and you can see the `Controller` window as shown in Figure 9.

Click [Connect] button to start.

5.3. Running simulation

On Cooja simulator, click [Start] button in `Simulation control` window to start simulation as shown in Figure 10.

Monitoring `Mote output` window on Cooja, you can see node 2 receiving a `"Hello World!"` message from node 3 as shown in Figure 11.

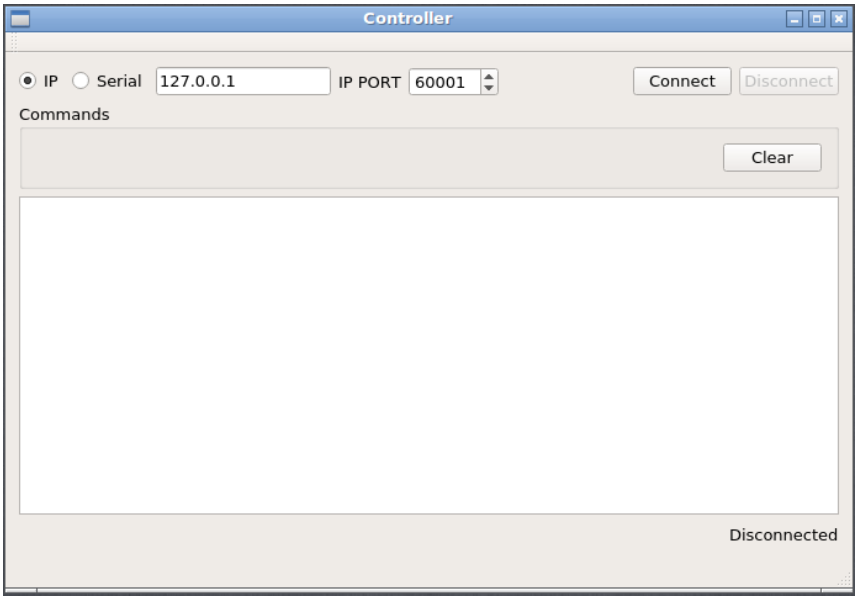

Figure 9. Controller window.

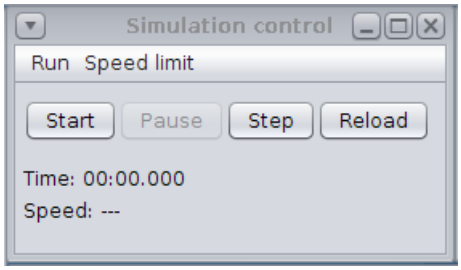

Figure 10. Simulation control window.

```
00:31.721  ID:2  Receiving message from [ 03 00 ] of len 12: Hello World!
```

Figure 11. Node 2 receiving message from node 3.

6. Getting Statistics

You can extract statistics from a COOJA simulation data by running `sdn_get_statistics.py` python script, located in folder `it-sdn/simulation`. For this, you need to install python (tested with python 2.7.6) and python libraries: python-scipy and python-matplotlib.

We created a sample `simulation_data` file from a simulation by using `File -> Append to file` option from `Mote output` tool in Cooja. We used the following command to extract the statistics:

```
~it-sdn/simulation$ python sdn_get_statistics.py simulation_data
```

The script output is shown bellow:

```
Delivery rates by source node:
(04 00) 100.00 [ 8 / 8 ]
(05 00) 100.00 [ 3 / 3 ]
(02 00) 100.00 [ 8 / 8 ]
```

```
(03 00) 100.00 [ 10 / 10 ]
(01 00) 100.00 [ 8 / 8 ]

Delay by source node:
(04 00) 595.25
(05 00) 10062.00
(02 00) 194.88
(03 00) 311.40
(01 00) 249.88

Delivery rates by packet type:
03 100.00 [ 3 / 3 ]  SDN_PACKET_DATA_FLOW_REQUEST
04 100.00 [ 8 / 8 ]  SDN_PACKET_NEIGHBOR_REPORT
05 100.00 [ 9 / 9 ]  SDN_PACKET_DATA
0C 100.00 [ 8 / 8 ]  SDN_PACKET_ACK
0D 100.00 [ 1 / 1 ]  SDN_PACKET_REGISTER_FLOWID
E7 100.00 [ 8 / 8 ]  SDN_PACKET_SRC_ROUTED_DATA_FLOW_SETUP
Fraction of data packets: 24.32%

Delay by packet type:
03 279.33  SDN_PACKET_DATA_FLOW_REQUEST
04 214.88  SDN_PACKET_NEIGHBOR_REPORT
05 504.78  SDN_PACKET_DATA
0C 325.75  SDN_PACKET_ACK
0D 29915.00  SDN_PACKET_REGISTER_FLOWID
E7 249.88  SDN_PACKET_SRC_ROUTED_DATA_FLOW_SETUP

Overall delivery rate:
100.00 [ 37 / 37 ]

Overall delay:
1124.86
```

The `Delivery rates by source node` section shows the ratio between packets delivered and packets sent from each node.

The `Delay by source node` section shows the average delivered packet delay, according to each transmitting node.

The `Delivery rates by packet type` section shows the packet delivery rates according to each packet type.

The `Delay by packet type` section shows the average delivered packet delay, according to each packet type.

The `Overall delivery rate` shows the ratio between all packets delivered and all packets sent in the simulation.

The `Overall delay` shows the average packet delay for all the packets in the simulation.

References

de Oliveira, B. T. and Margi, C. B. (2016). TinySDN: Enabling TinyOS to Software-Defined Wireless Sensor Networks. In *XXXIV Simpósio Brasileiro de Redes de Computadores*, pages 1229–1237.

de Oliveira, B. T., Margi, C. B., and Gabriel, L. B. (2014). TinySDN: Enabling multiple controllers for software-defined wireless sensor networks. In *Communications (LATINCOM), 2014 IEEE Latin-America Conference on*, pages 1–6.