

Mục lục

Danh sách hình vẽ	2
1 GIỚI THIỆU VÀ MỤC TIÊU ĐỀ TÀI	3
1.1 Giới thiệu đề tài	3
1.2 Mục tiêu đề tài	3
2 THIẾT KẾ VÀ THỰC HIỆN	4
2.1 Topo của network	4
2.2 Quy trình thực hiện	5
3 KẾT QUẢ	11
3.1 Kịch bản mô phỏng	11
3.2 Kết quả mô phỏng	11
3.3 Kết luận	13

Danh sách hình vẽ

1	Topo của network	4
2	Tạo 1 block Stream trên Adafruit IO	6
3	Dashboard sau khi tạo 1 block mới	6
4	Adafruit IO Key để MQTT Client có thể được xác thực và lấy dữ liệu từ topic . .	7
5	Cách điền vào các field của trigger dành cho Google Assistant	8
6	Applet được tạo ra sau khi thực hiện quy trình	9
7	Kết quả sau khi nói "turn on 2"	11
8	Kết quả sau khi nói "turn off 2"	12
9	Kết quả sau khi nói "turn on 3"	12
10	Kết quả sau khi nói "turn off 3"	13

1 GIỚI THIỆU VÀ MỤC TIÊU ĐỀ TÀI

1.1 Giới thiệu đề tài

Trong bối cảnh Internet vạn vật (IoT) ngày càng phát triển, người dùng càng mong muốn họ không phải tới gần thiết bị để điều khiển mà họ có thể điều khiển từ xa thông qua smartphone của mình. Trong đề tài này nhóm xin trình bày cách thiết kế một mạng cảm biến không dây được điều khiển thông qua ứng dụng Google Assistant của Google.

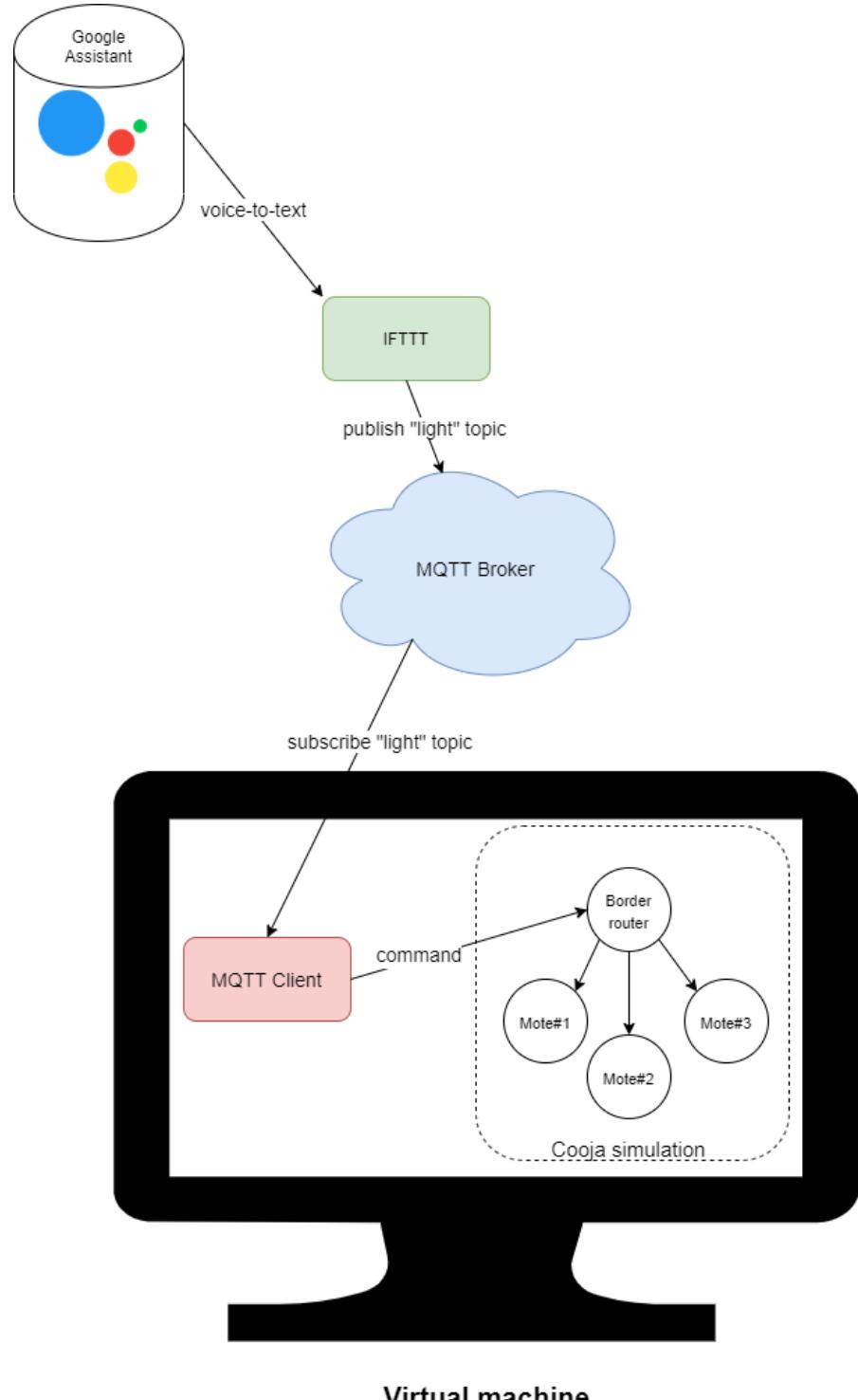
1.2 Mục tiêu đề tài

Mục tiêu đề tài này sẽ tập trung vào thiết kế và mô phỏng bằng Cooja một hệ thống gồm mạng cảm biến không dây và các ứng dụng khác cần thiết trong một máy ảo và hệ thống trên có thể được điều khiển các đèn LED màu xanh bởi Google Assistant.

2 THIẾT KẾ VÀ THỰC HIỆN

2.1 Topo của network

Topo của network, bao gồm mạng cảm biến và các ứng dụng cần thiết được trình bày qua sơ đồ sau



Hình 1: Topo của network

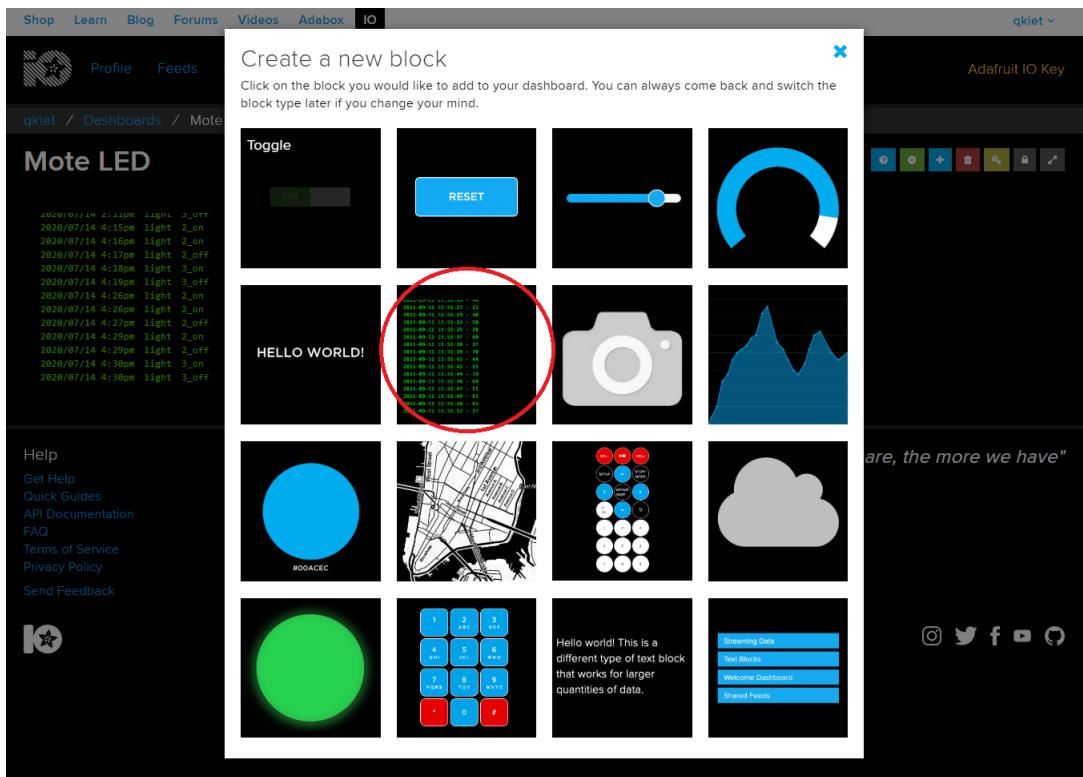
Theo sơ đồ trên thì ta có các phần tử quan trọng trong network như sau

- **Google Assistant:** đây chính là thiết bị để chuyển giọng nói của người thành chữ. Google Assistant còn đảm nhiệm chức năng gửi text chuyển được lên platform IFTTT để xử lý lệnh từ giọng nói
- **IFTTT:** là 1 platform dùng để kết nối thiết bị và ứng dụng với nhau. Cụ thể ở đây IFTTT sẽ kết nối Google Assistant với MQTT Broker. Như vậy, bất cứ khi nào Google Assistant gửi text tới IFTTT thì IFTTT đảm nhiệm vai trò publisher cho MQTT network và publish cho MQTT Broker
- **MQTT Broker** sẽ là nơi chuyển tiếp message được publish cho các client subscribe vào topic tương ứng. Trong topo này, MQTT Broker đóng vai trò trung gian để IFTTT giao tiếp với MQTT Client ở máy ảo
- **MQTT Client:** là 1 ứng dụng được chạy trong máy ảo để chuyển text chuyển từ Google Assistant trở thành command điều khiển các led trong mạng cảm biến được mô phỏng bằng Cooja. Lưu ý rằng để nhận được text từ Google Assistant thì MQTT Client lẫn IFTTT phải cùng sử dụng 1 MQTT Broker và 1 topic (trong topo thì topic tên là “light”)
- **Mạng cảm biến:** chính là end user cuối cùng trong topo mạng này. Mạng cảm biến bao gồm 1 border router đóng vai trò liên kết mạng cảm biến với network bên ngoài, như vậy command từ MQTT Client mới có thể truyền tới các mote trong mạng cảm biến thông qua UDP và IPv6

2.2 Quy trình thực hiện

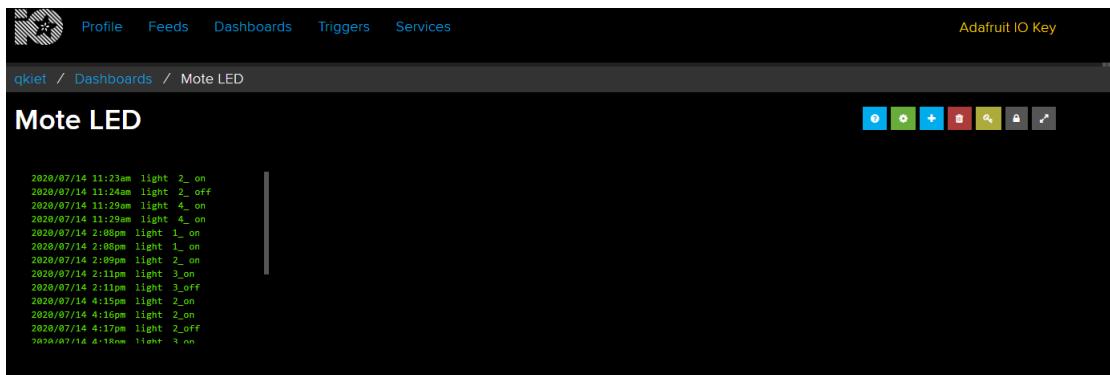
Adafruit IO

Trước hết phải tạo một MQTT Broker để text từ Google Assistant có thể truyền tới MQTT Client và Client sử dụng nó để điều khiển LED. Ta sẽ vào <https://io.adafruit.com> và đăng ký tài khoản. Sau đó tạo topic "light" để IFTTT có thể gửi cho MQTT Client, nhưng topic không tạo trực tiếp trên Adafruit IO mà phải thông qua tạo 1 block trong Dashboard. Block là 1 đơn vị hiển thị thông tin hoặc tương tác với các feed và việc tạo Block đồng thời tạo một feed mới, nghĩa là tạo 1 topic mới. Có rất nhiều loại block khác nhau như button, slider... và nhóm chọn block Stream để có thể theo dõi message nào được gửi lên



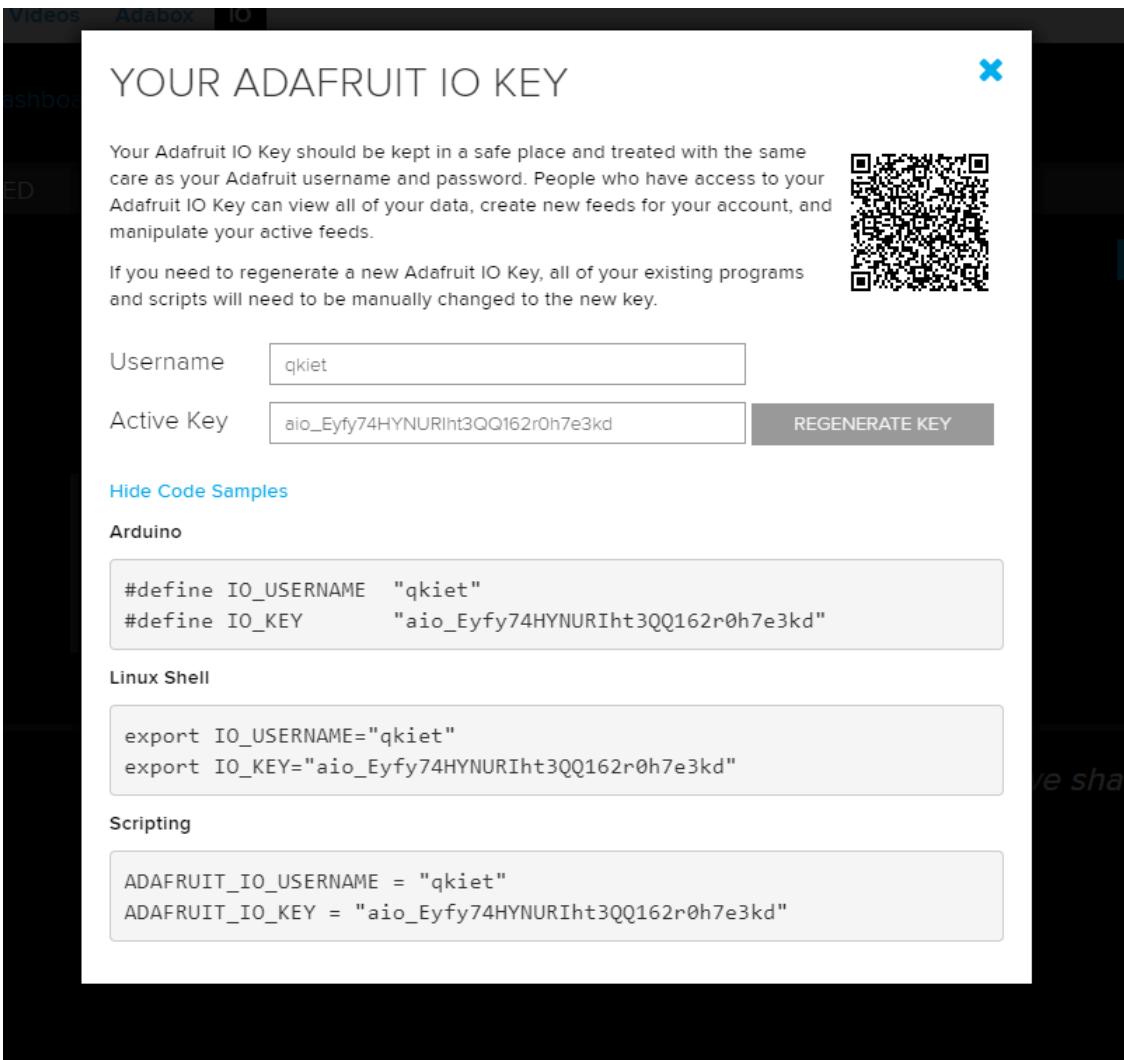
Hình 2: Tạo 1 block Stream trên Adafrui IO

Như vậy ta đã tạo 1 topic mới, và dashboard hiển thị 1 block mới như sau:



Hình 3: Dashboard sau khi tạo 1 block mới

Tuy nhiên ngoài việc tạo topic mới ta cũng cần lưu lại các thông tin quan trọng để MQTT Client có thể connect vào Adafruit IO vì Adafruit IO không phải là 1 public Broker nên nó sẽ có cơ chế bảo mật riêng. Vào Adafruit IO Key ở phía trên bên phải và lưu lại username và active key



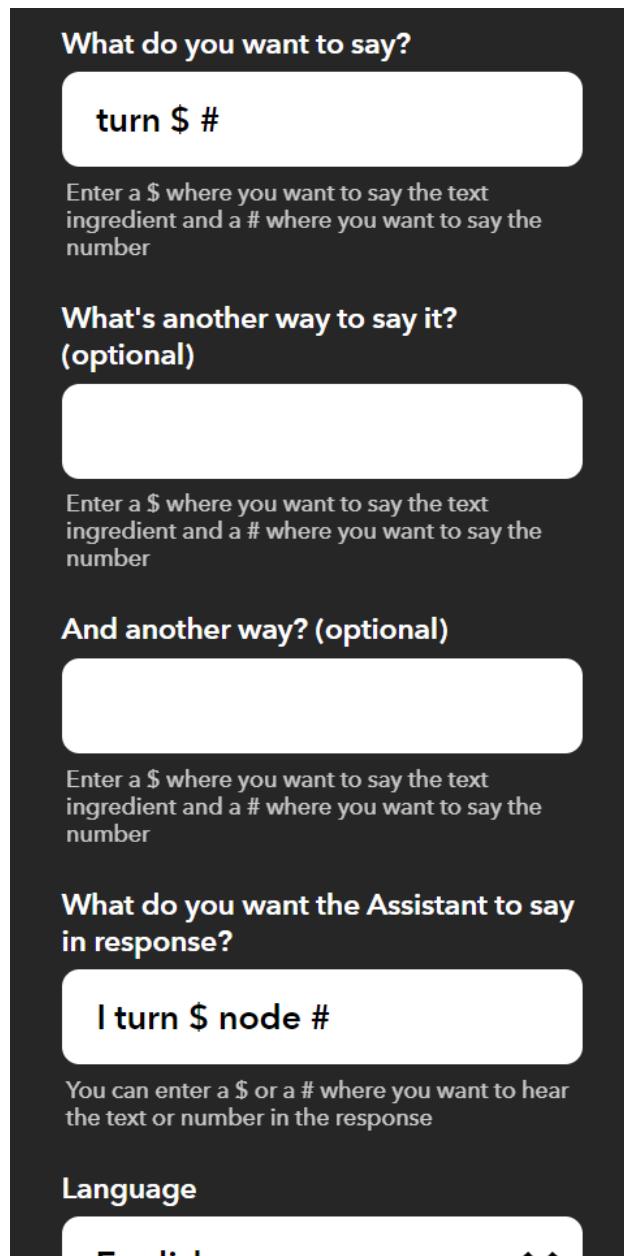
Hình 4: Adafruit IO Key để MQTT Client có thể được xác thực và lấy dữ liệu từ topic

IFTTT

IFTTT hoạt động thông qua việc người dùng tạo ra các applet để kết nối một ứng dụng với một service. Applet gồm một trigger là điều kiện kích hoạt applet này và service là nơi applet sẽ thực hiện. Như vậy đối với đề tài thì trigger sẽ là Google Assistant và service chính là 1 MQTT Broker.

Cách làm applet này như sau:

- Trước hết phải đăng ký một tài khoản với IFTTT để có thể tạo ra applet
- Đăng nhập tài khoản IFTTT, sau đó bắt đầu tạo ra một applet mới bằng cách bấm vào nút "Create" góc trên bên phải
- Khi yêu cầu chọn service cho trigger của applet thì ta chọn "Google Assistant" với kiểu kích hoạt "Say a phrase with both a number and a text ingredient" vì khi điều khiển bằng Google Assistant ta mong muốn có thể nói bật tắt và mote ID
- Điền vào trigger như hình dưới đây:



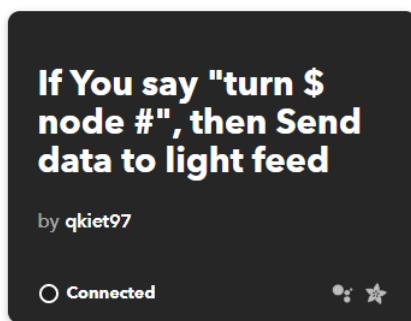
Hình 5: Cách điền vào các field của trigger dành cho Google Assistant

Như vậy cú pháp để tắt mở các đèn led là "turn + (on/off) + mote ID". Ví dụ ta muốn mở đèn LED mote số 2 thì chỉ cần nói "turn on 2"

- Trong phần action service ta sẽ chọn Adafruit và chọn Send data to Adafruit IO. Adafruit IO chính là MQTT Broker mà IFTTT sẽ gửi data khi nhận trigger từ Google Assistant
- Sau đó chọn **Feed name** là "light", feed mà nhóm đã tạo trước 1 feed trên Adafruit và đây cũng là feed mà MQTT client sẽ subscribe. **Data to save** sẽ là {{NumberField}}_{{TextField}}, tương ứng là (mote ID)_on/off. Ví dụ khi tắt đèn led của mode ID 3 thì data gửi lên Adafruit IO sẽ là 3_off

Kết quả ta có 1 applet như sau:

My Applets



Hình 6: Applet được tạo ra sau khi thực hiện quy trình

MQTT Client

MQTT Client nhận data subscribe tại feed "light" và được chạy tại máy ảo. MQTT Client được viết bằng python và sử dụng thư viện Adafruit_io để ứng dụng có thể tạo kết nối với MQTT Broker Adafruit IO và lấy data về. Ngoài ra MQTT client còn xử lý data lấy từ topic "light" và chuyển thành command và địa chỉ IPv6 cần gửi.

Code của MQTT Client được trình bày như sau:

```
import logging
import paho.mqtt.client as mqtt
import time
import sys

import socket

HOST = [ 'aaaa::212:7402:2:202' , 'aaaa::212:7403:3:303' , 'aaaa::212:7404:4:404' ]
# Standard loopback interface address (localhost)

PORT = 3000          # Port to listen on (non-privileged ports are > 1023)

USERNAME    = 'tuannv'
KEY         = 'aio_vCRp50VNZfCzjGLQou8mf9mjBlj'

ADAFRUIT_SERVER      = 'io.adafruit.com'
ADAFRUIT_PORT        = 1883

PATH          = USERNAME + '/feeds/Light'

def on_connect(client, userdata, flags, rc):
    print('Connected!')
    client.subscribe(PATH)
    print('Subscribed to path {}'.format(PATH))

def on_disconnect(client, userdata, rc):
    print('Disconnected!')
```

```

def on_message(client, userdata, msg):
    cmd = msg.payload.decode('utf-8')
    s = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
    print('Received on {0}: {1}'.format(msg.topic, cmd))
    if (int(cmd[0]) == 2): #find out which address
        destination_index = 0
    elif (int(cmd[0]) == 3):
        destination_index = 1
    elif (int(cmd[0]) == 4):
        destination_index = 2
    if (cmd[2:] == "ON"):
        send_data = "led_on"
    elif (cmd[2:] == "OFF"):
        send_data = "led_off"

    s.connect((HOST[destination_index], PORT))
    print(destination_index)
    print(send_data)
    s.sendall(send_data.encode())
    sys.stdout.flush()
    s.close()

client = mqtt.Client()
client.username_pw_set(USERNAME, KEY)
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_message = on_message
client.connect(ADAFRUIT_SERVER, port=ADAFRUIT_PORT)
client.loop_forever()

```

Như vậy bản chất MQTT Client hoạt động là 1 MQTT Subscriber. Client sẽ được cho trước topic, username và active key và sau đó thực hiện kết nối lên MQTT Broker. Sau đó nó đợi nhận message mới và tiến hành xử lý message và gửi cho mote tương ứng.

Mạng cảm biến

Mạng cảm biến sẽ bao gồm 1 sky mote được nạp firmware border router và các mote khác sẽ là các mote được Google Assistant điều khiển và được nạp firmware *udp-echo-Broker* trong folder **contiki/example/cc2538dk/demo_cc2538** vì firmware này đã có sẵn các lệnh để điều khiển LED màu xanh. Để đơn giản hóa việc mô phỏng thì mạng cảm biến chỉ dùng 2 mote *udp-echo-Broker*

3 KẾT QUẢ

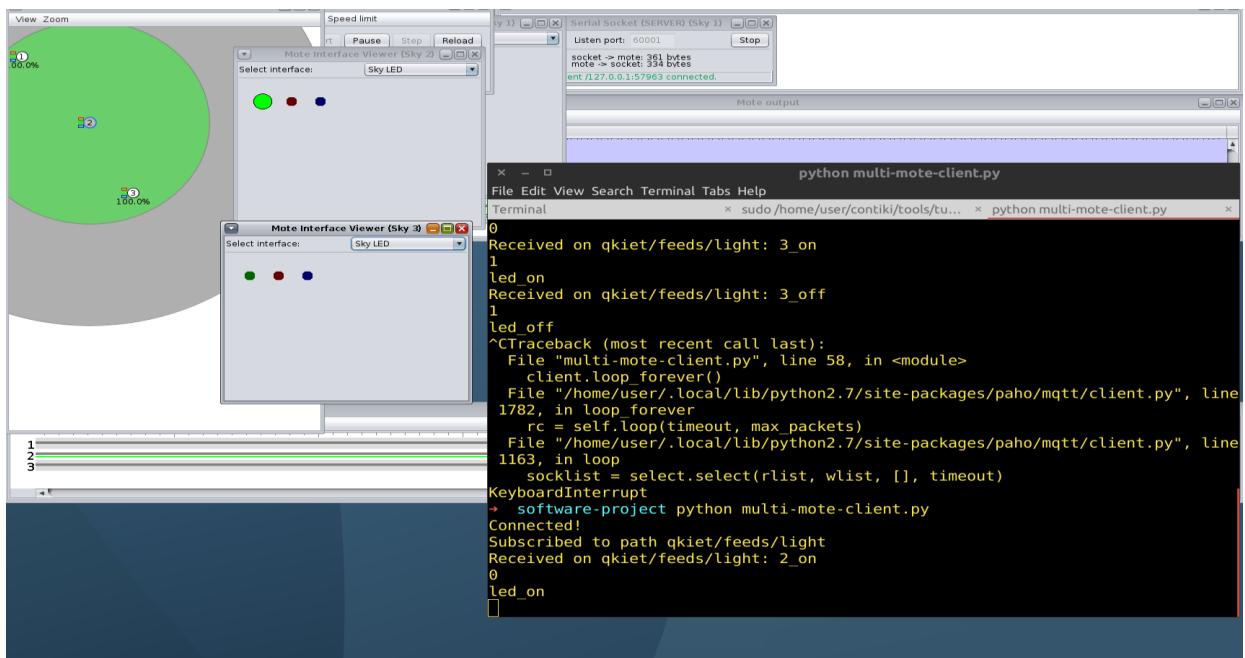
3.1 Kịch bản mô phỏng

Lần lượt nói các cú pháp sau và quan sát kết quả

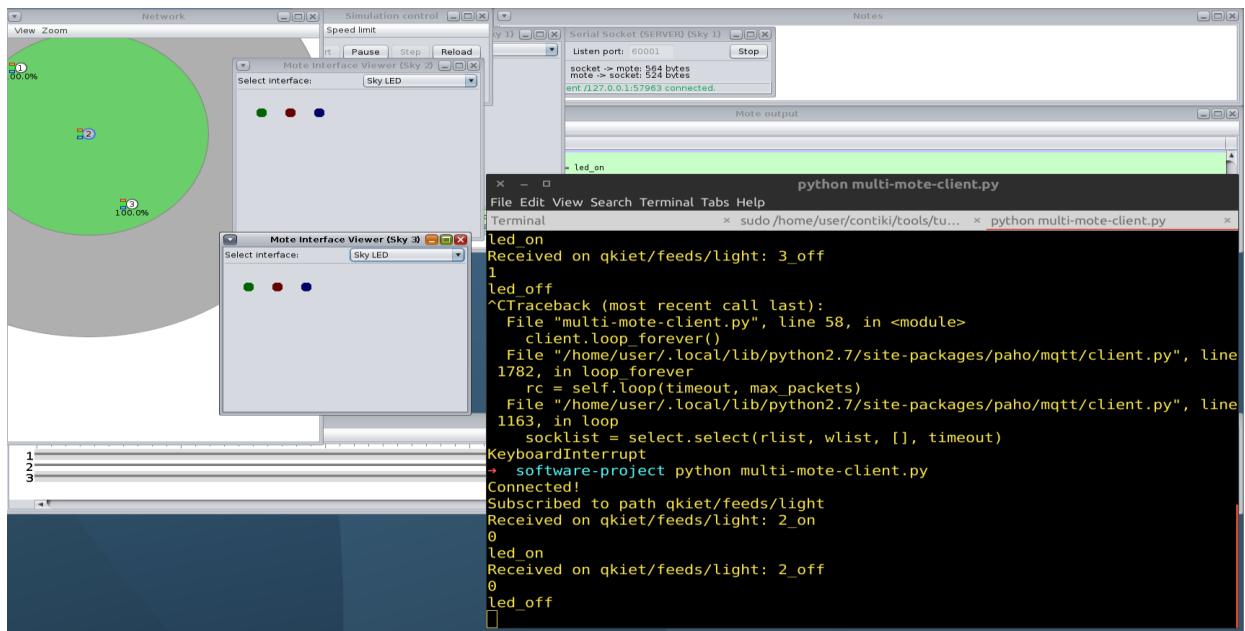
- "turn on 2". Mong đợi kết quả đèn LED xanh lá của mote Id 2 mở
- "turn off 2". Mong đợi kết quả đèn LED xanh lá của mote Id 2 tắt
- "turn on 3". Mong đợi kết quả đèn LED xanh lá của mote Id 3 mở
- "turn off 3". Mong đợi kết quả đèn LED xanh lá của mote Id 3 tắt

3.2 Kết quả mô phỏng

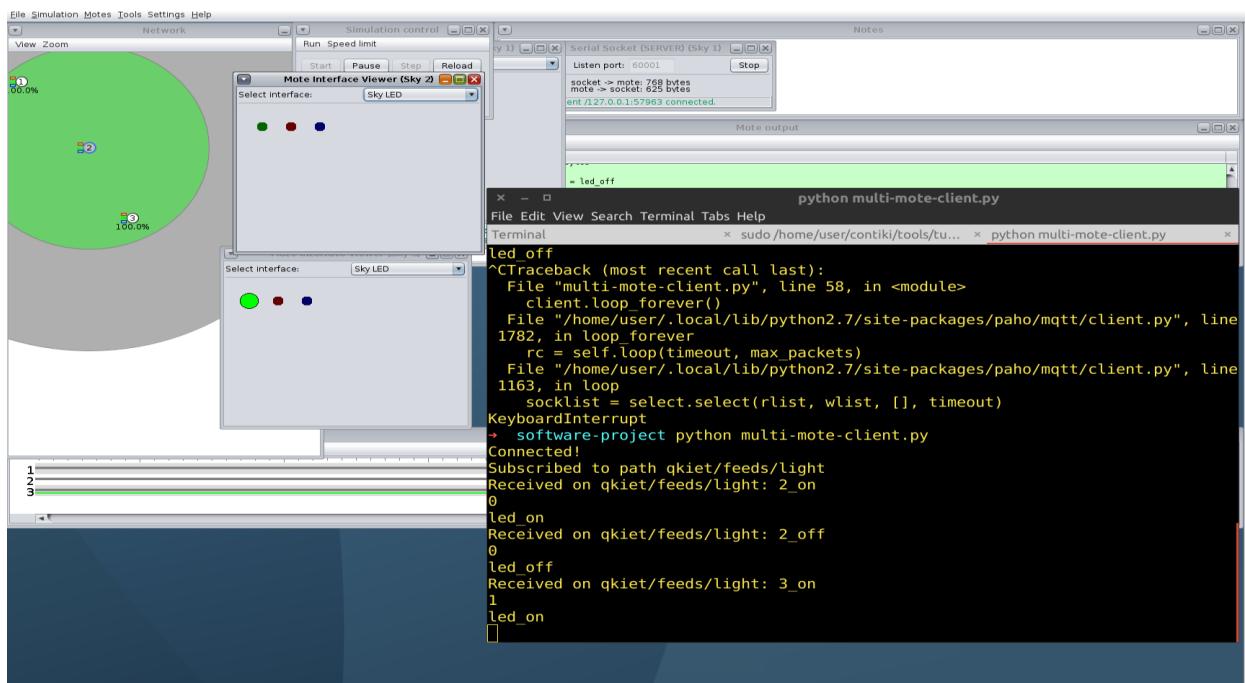
Sau đây là kết quả sau khi nói các cú pháp trong kịch bản. Lưu ý rằng trạng thái các đèn LED ở phía trên của mote 2 và ở phía dưới là của mote 3.



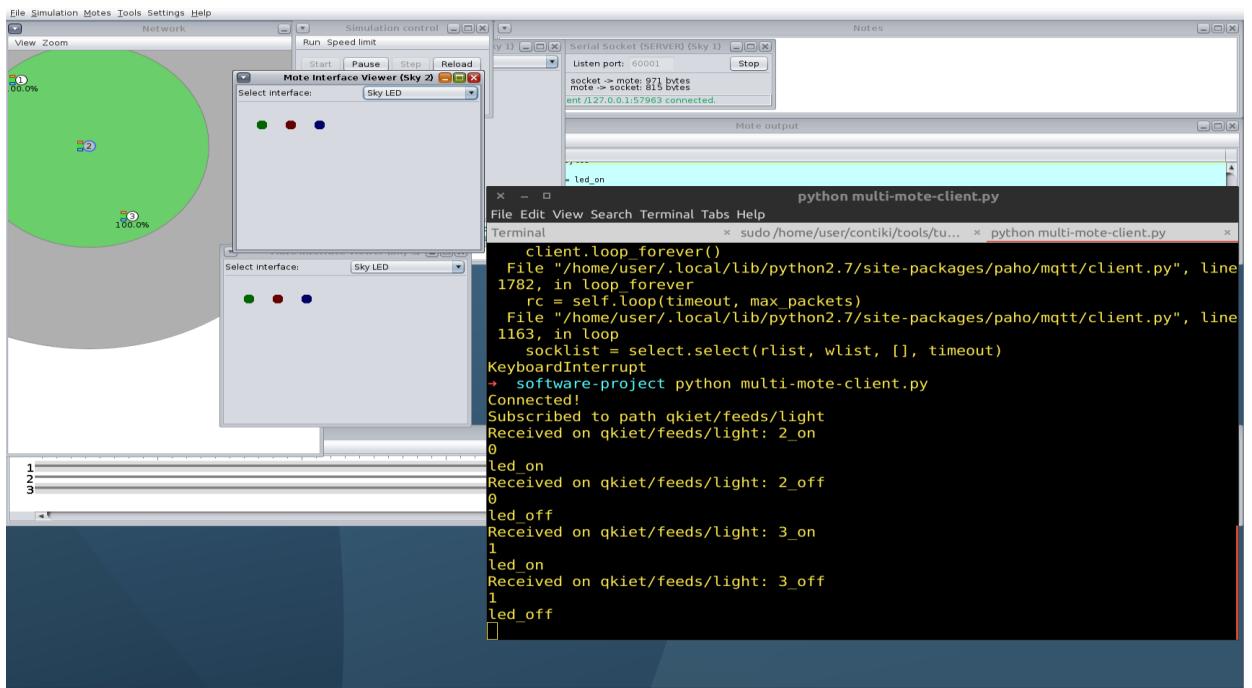
Hình 7: Kết quả sau khi nói "turn on 2"



Hình 8: Kết quả sau khi nói "turn off 2"



Hình 9: Kết quả sau khi nói "turn on 3"



Hình 10: Kết quả sau khi nói "turn off 3"

Tham khảo link youtube sau để có thể quan sát được quá trình Google Assistant điều khiển mạng cảm biến được mô phỏng ra sao: <https://youtu.be/GaGcWuapAZ0>

3.3 Kết luận

Như vậy hệ thống đã hoạt động đúng như những gì kịch bản mô phỏng đề ra và có thể thấy rằng hệ thống hoạt động tốt

ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN VIỄN THÔNG



BÁO CÁO BÀI TẬP NHÓM

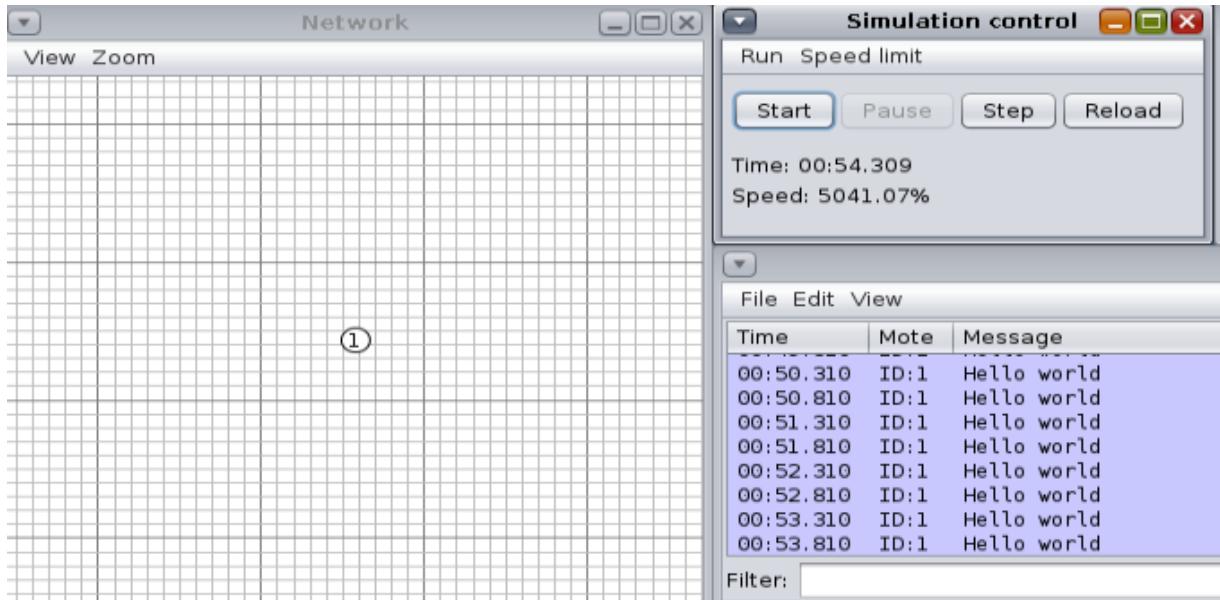
Môn học: Mạng cảm biến không dây

GVHD: TS. Võ Quê Sơn
SVTH: Phạm Quang Kiệt – 1511662
Ngô Văn Tuân – 1613861
Bùi Thanh Phong - 1512430

TP. HỒ CHÍ MINH, THÁNG 07 NĂM 2020

I. Assignment 1: Hello world

Sau khi compile **contiki/example/hello-world/hello-world.c** và mô phỏng trên Cooja ta được kết quả sau:



Có thể thấy rằng trong ví dụ gốc thì cứ mỗi 500ms mote sẽ thực hiện lệnh printf “Hello world” và dẫn đến kết quả trên.

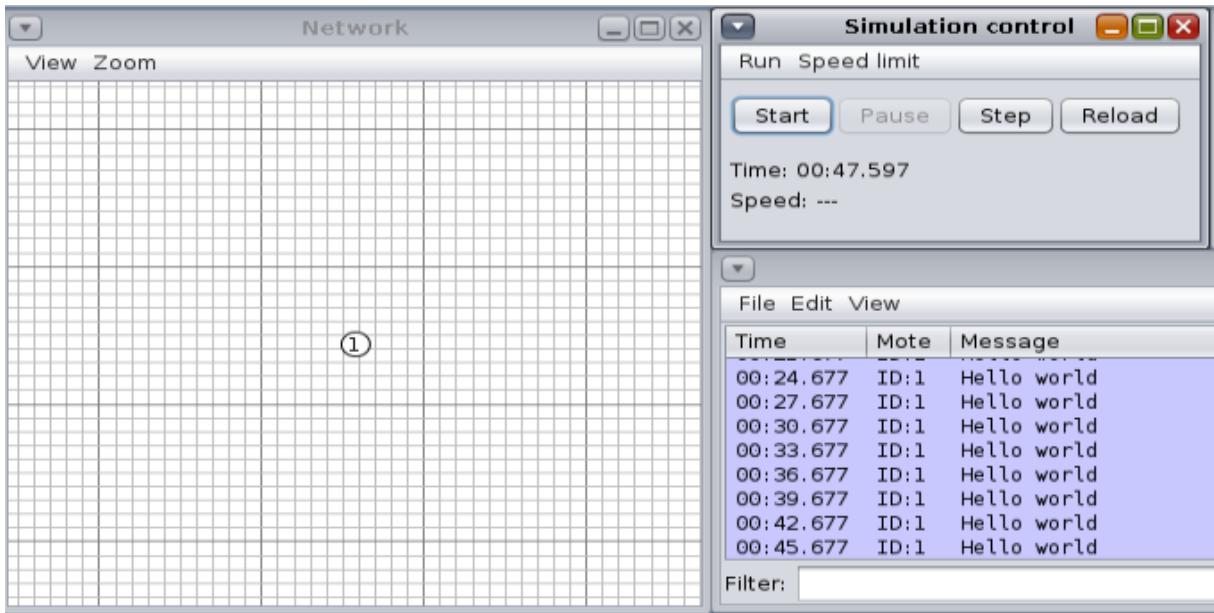
Để in ra dòng chữ “Hello world” sau mỗi 3 giây, ta vào file **contiki/example/hello-world/hello-world.c** và thay đổi dòng lệnh sau đây

```
etimer_set(&et, CLOCK_SECOND *0.5);
```

trở thành:

```
etimer_set(&et, CLOCK_SECOND *3);
```

lệnh trên sẽ tạo ra 1 timer có chu kỳ 0.5 giây, ta thay đổi thành 3 giây. Compile lại file và quan sát kết quả mô phỏng trên Cooja:



Như vậy ta đã in được dòng chữ “Hello world” chu kì 3 giây.

II. Assignment 2: Choose the BNN

Chỉnh sửa code từ ví dụ Broadcast.

Gửi giá trị RSSI cho các Node khác.

```
packetbuf_clear();
packetbuf_set_datalen(sprintf(packetbuf_dataptr(), "%d", cc2420_last_rssi, 6));

broadcast_send(&broadcast);
printf("broadcast message sent\n");
```

Tách dữ liệu RSSI trong gói tin Broadcast được gửi từ các node khác.

ở đây ta dùng thêm một biến mảng rssi_table[] để lưu giá trị RSSI của các Node khác.

Vì dữ liệu nhận từ gói Broadcast là chuỗi kí tự Char nên ta cần chuyển kiểu dữ liệu về Int. Ta sử dụng hàm atoi() thuộc thư viện stdlib.h để thực hiện việc này.

```

static void
broadcast_recv(struct broadcast_conn *c, const linkaddr_t *from)
{
    rss_i_table[from->u8[0]]=atoi((char*)packetbuf_dataptr());
    printf("broadcast message received from %d.%d: RSSI=%d\n",
           from->u8[0] , from->u8[1] ,rss_i_table[from->u8[0]],packetbuf_totlen());
}
static const struct broadcast_callbacks broadcast_call = {broadcast_recv};
static struct broadcast_conn broadcast;
/*-----*/

```

Viết hàm so sánh giá trị RSSI và cho ra 5 giá trị tốt nhất. Chọn 1 Node là BNN.

Lấy từng giá trị trong mảng rss_i_table[] so sánh với các giá trị khác nếu giá trị đó lớn hơn 10 giá trị khác sẽ được đưa vào danh sách có giá trị RSSI tốt.

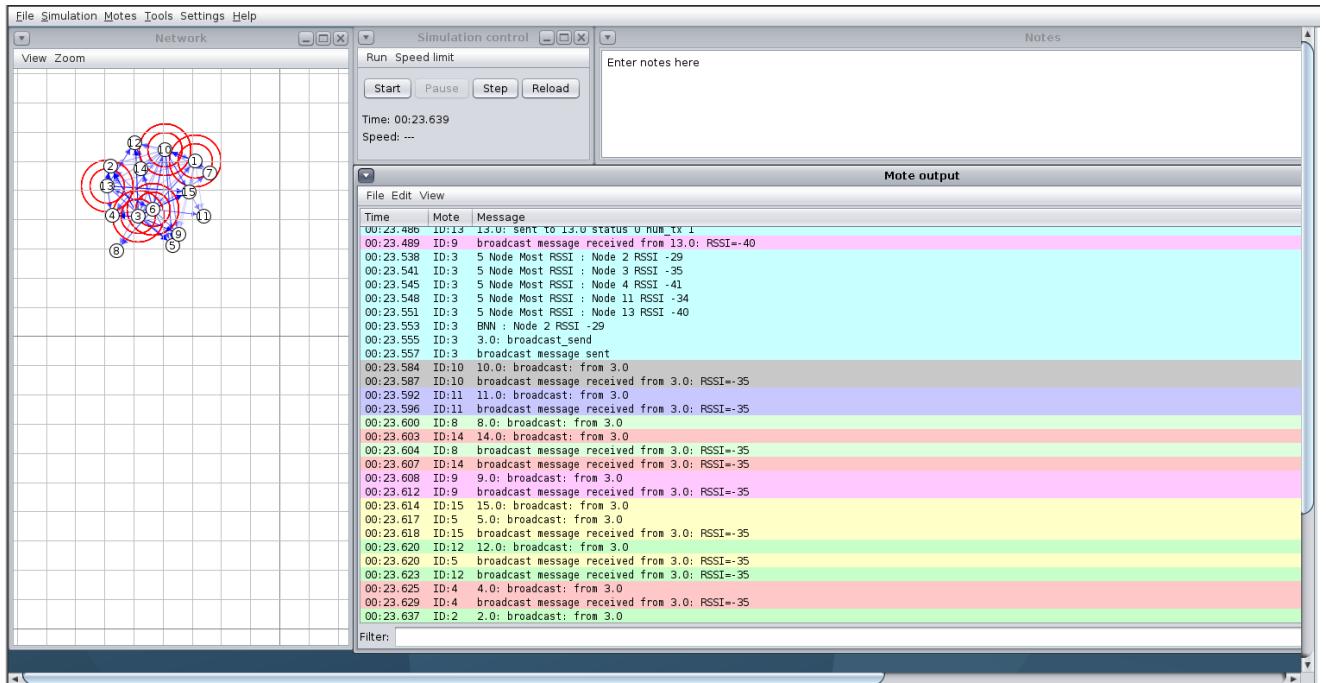
Thực hiện tương tự như vậy để chọn ra BNN.

```

static void most_rssi()
{
    unsigned char i=1;
    while (i<=15)
    {
        int j=1;
        int n=0;
        while (j<=15)
        {
            if( rss_i_table[i]>rss_i_table[j] )
                n++;
            j++;
        }
        if(n>=10)
            printf("5 Node Most RSSI : Node %d RSSI %d \n",i,rss_i_table[i]);
        i++;
    }
    int max=-1000;
    unsigned char k;
    i=1;
    while(i<=15)
    {
        if(rss_i_table[i] > max)
        {
            max = rss_i_table[i];
            k=i;
        }
        i++;
    }
    printf("BNN : Node %d RSSI %d \n",k,max);
}

```

Kết quả mô phỏng.



```

00:23.538 ID:3 5 Node Most RSSI : Node 2 RSSI -29
00:23.541 ID:3 5 Node Most RSSI : Node 3 RSSI -35
00:23.545 ID:3 5 Node Most RSSI : Node 4 RSSI -41
00:23.548 ID:3 5 Node Most RSSI : Node 11 RSSI -34
00:23.551 ID:3 5 Node Most RSSI : Node 13 RSSI -40
00:23.553 ID:3 BNN : Node 2 RSSI -29

00:23.584 ID:10 10.0: broadcast: from 3.0
00:23.587 ID:10 broadcast message received from 3.0: RSSI=-35
00:23.592 ID:11 11.0: broadcast: from 3.0
00:23.596 ID:11 broadcast message received from 3.0: RSSI=-35
00:23.600 ID:8 8.0: broadcast: from 3.0
00:23.603 ID:14 14.0: broadcast: from 3.0
00:23.604 ID:8 broadcast message received from 3.0: RSSI=-35
00:23.607 ID:14 broadcast message received from 3.0: RSSI=-35
00:23.608 ID:9 9.0: broadcast: from 3.0
00:23.612 ID:9 broadcast message received from 3.0: RSSI=-35
00:23.614 ID:15 15.0: broadcast: from 3.0
00:23.617 ID:5 5.0: broadcast: from 3.0
00:23.618 ID:15 broadcast message received from 3.0: RSSI=-35
00:23.620 ID:12 12.0: broadcast: from 3.0
00:23.620 ID:5 broadcast message received from 3.0: RSSI=-35
00:23.623 ID:12 broadcast message received from 3.0: RSSI=-35
00:23.625 ID:4 4.0: broadcast: from 3.0
00:23.629 ID:4 broadcast message received from 3.0: RSSI=-35
00:23.637 ID:2 2.0: broadcast: from 3.0

```

```

00:22.852 ID:8 broadcast message received from 1.0: RSSI=-42
00:22.854 ID:10 broadcast message received from 1.0: RSSI=-42
00:22.855 ID:14 broadcast message received from 1.0: RSSI=-42
00:22.860 ID:9 9.0: broadcast: from 1.0
00:22.863 ID:9 broadcast message received from 1.0: RSSI=-42
00:22.865 ID:5 5.0: broadcast: from 1.0
00:22.866 ID:15 15.0: broadcast: from 1.0
00:22.869 ID:5 broadcast message received from 1.0: RSSI=-42
00:22.869 ID:15 broadcast message received from 1.0: RSSI=-42
00:22.871 ID:12 12.0: broadcast: from 1.0
00:22.874 ID:4 4.0: broadcast: from 1.0
00:22.875 ID:12 broadcast message received from 1.0: RSSI=-42
00:22.877 ID:4 broadcast message received from 1.0: RSSI=-42
00:22.888 ID:2 2.0: broadcast: from 1.0
00:22.891 ID:2 broadcast message received from 1.0: RSSI=-42
00:22.902 ID:6 6.0: broadcast: from 1.0
00:22.905 ID:6 broadcast message received from 1.0: RSSI=-42
00:22.910 ID:13 13.0: broadcast: from 1.0
00:22.914 ID:13 broadcast message received from 1.0: RSSI=-42
00:22.927 ID:3 3.0: broadcast: from 1.0
00:22.929 ID:7 7.0: broadcast: from 1.0
00:22.930 ID:3 broadcast message received from 1.0: RSSI=-42
00:22.933 TD:7 broadcast message received from 1.0: RSSI=-42

```

III. Assignment 3: MAC Protocols

Bỏ define PERIOD trong file .c của client.

Sử dụng file project-conf.h để thay đổi MAC driver và PERIOD (thay đổi traffic)

Contikimac - High traffic:

```

19 #undef PERIOD
20 #define PERIOD 1
21 #define NETSTACK_CONF_MAC      csma_driver
22 #define NETSTACK_CONF_RDC      contikimac_driver
23 #define NETSTACK_CONF_FRAMER   framer_802154

```

Contikimac - Low traffic:

```

25 #undef PERIOD
26 #define PERIOD 10
27 #define NETSTACK_CONF_MAC      csma_driver
28 #define NETSTACK_CONF_RDC      contikimac_driver
29 #define NETSTACK_CONF_FRAMER   framer_802154

```

Xmac - High traffic:

```
7 #undef PERIOD
8 #define PERIOD 1
9 #define NETSTACK_CONF_MAC      csma_driver
10 #define NETSTACK_CONF_RDC      cxmac_driver
11 #define NETSTACK_CONF_FRAMER   framer_802154
```

Xmac - Low traffic:

```
13 #undef PERIOD
14 #define PERIOD 10| You, a few seconds ago .
15 #define NETSTACK_CONF_MAC      csma_driver
16 #define NETSTACK_CONF_RDC      cxmac_driver
17 #define NETSTACK_CONF_FRAMER   framer_802154
```

Thay đổi dense network và loose network bằng cách thay đổi cách bố trí các node trong mạng mô phỏng:

- Dense network: các node được xếp khá gần nhau.
- Loose network: các node được kéo xa ra sao cho vẫn có thể giao tiếp với ít nhất một node khác trong mạng.

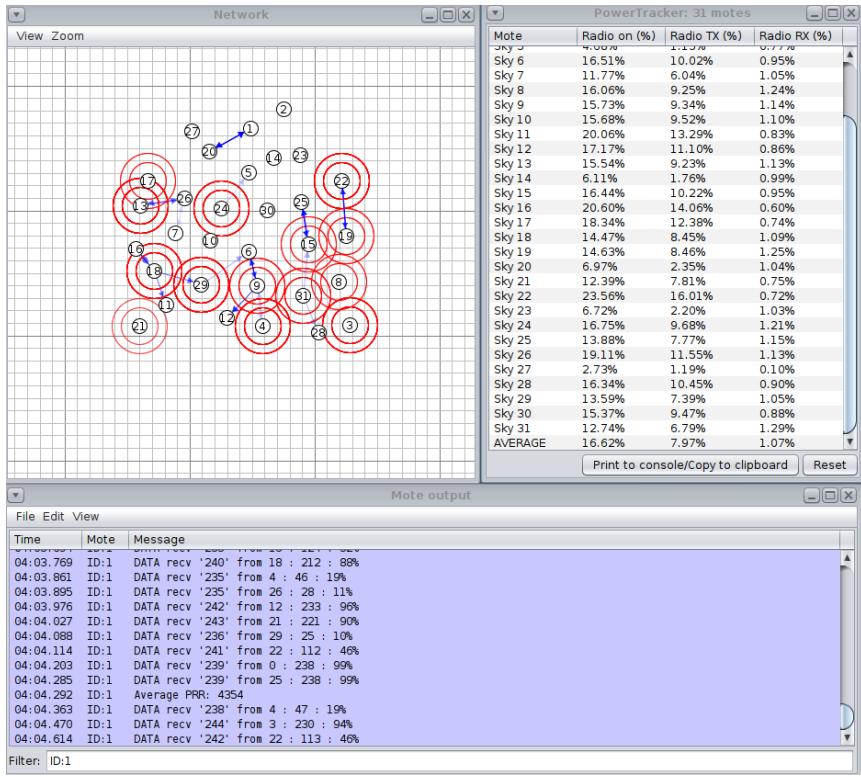


Figure 1. Contikimac - Dense network - High traffic

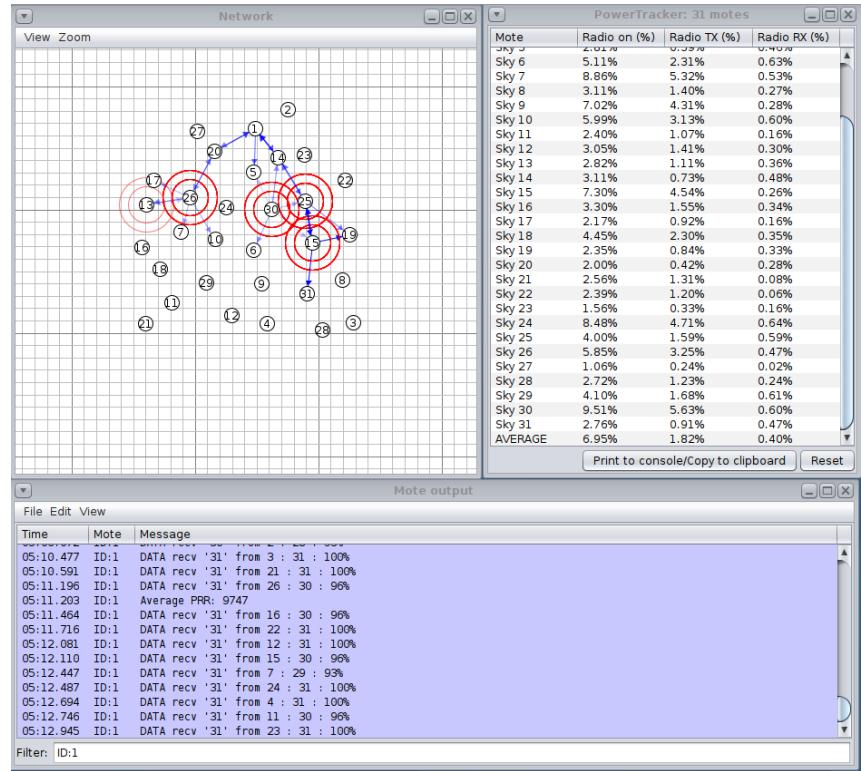


Figure 2. Contikimac - Dense network - Low traffic

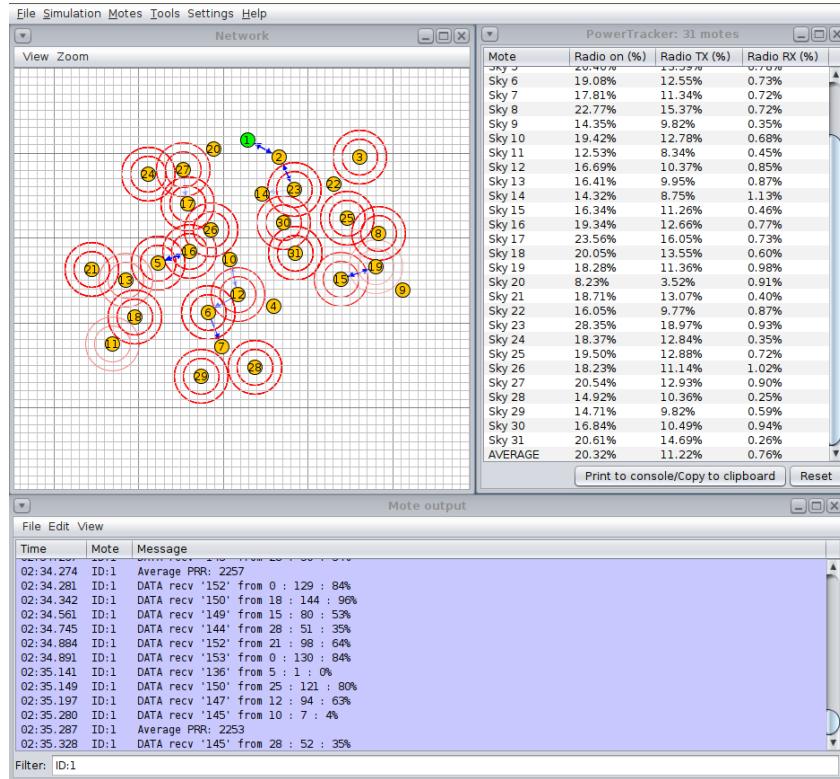


Figure 3. Contikimac - Loose network - High traffic

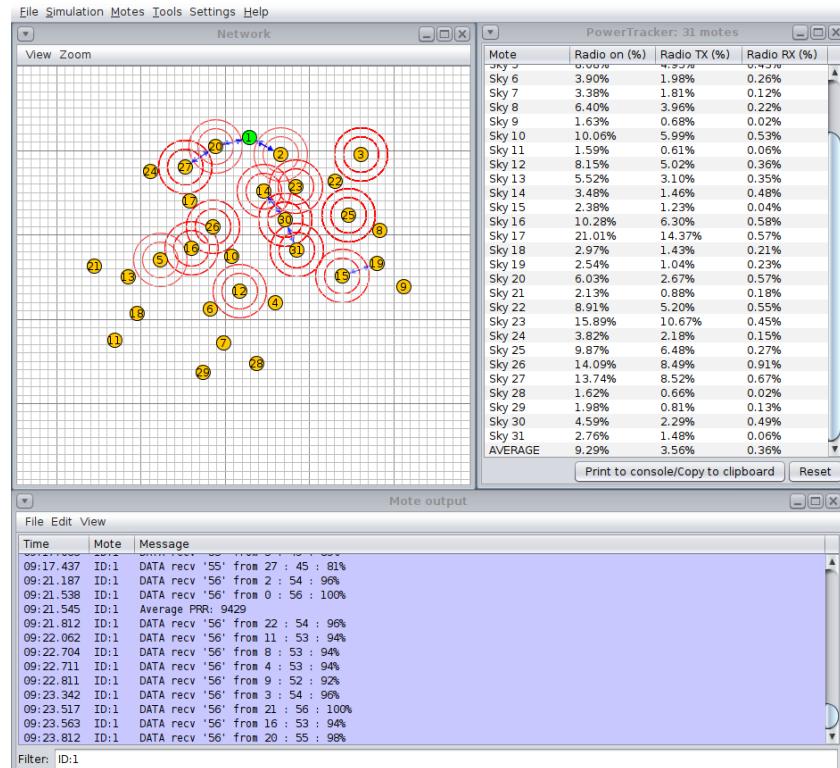


Figure 4. Contikimac - Loose network - Low traffic

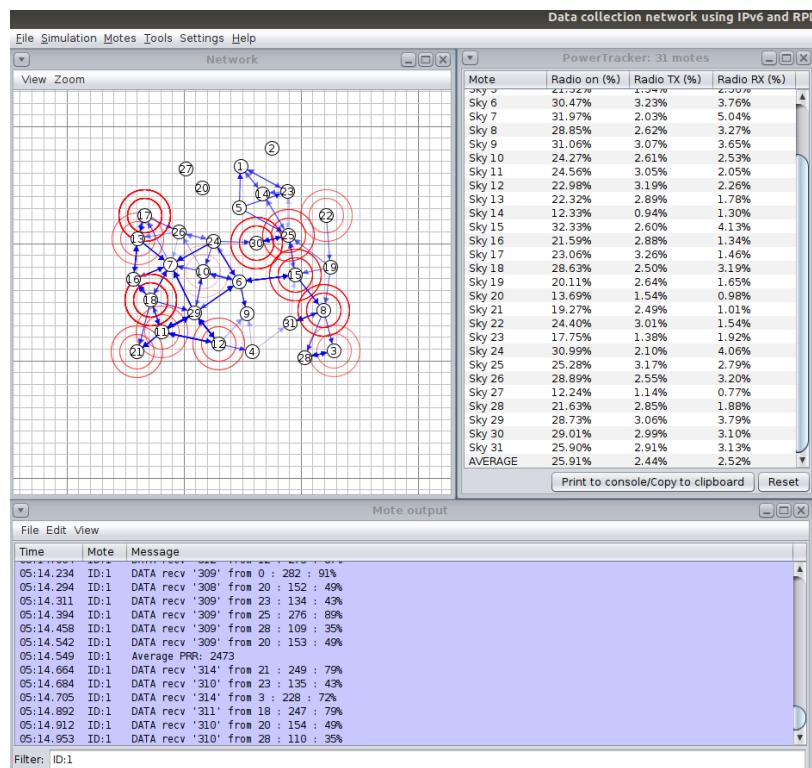


Figure 5. Cxmac - Dense network - High traffic

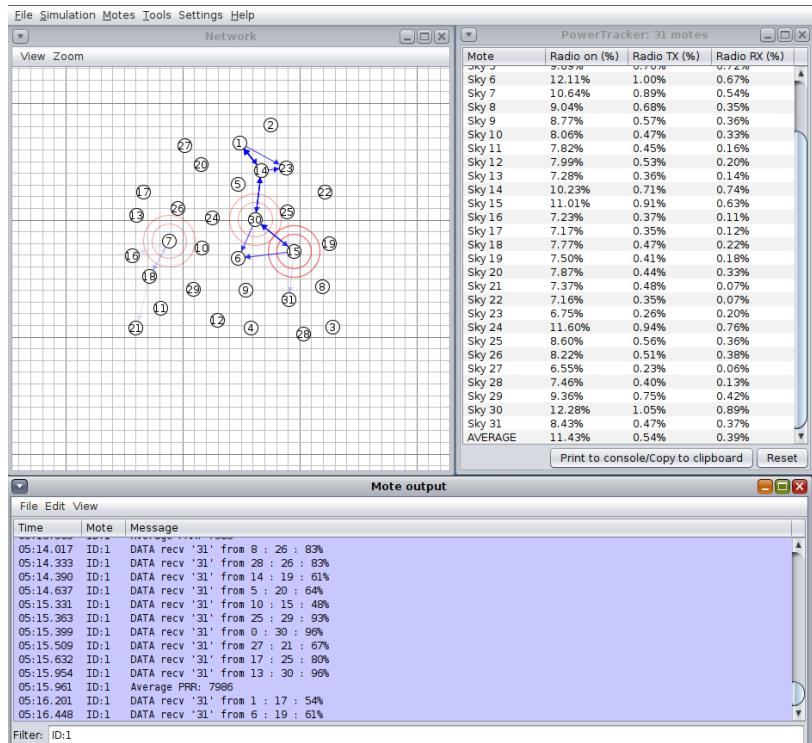


Figure 6. Cxmac - Dense network - Low traffic

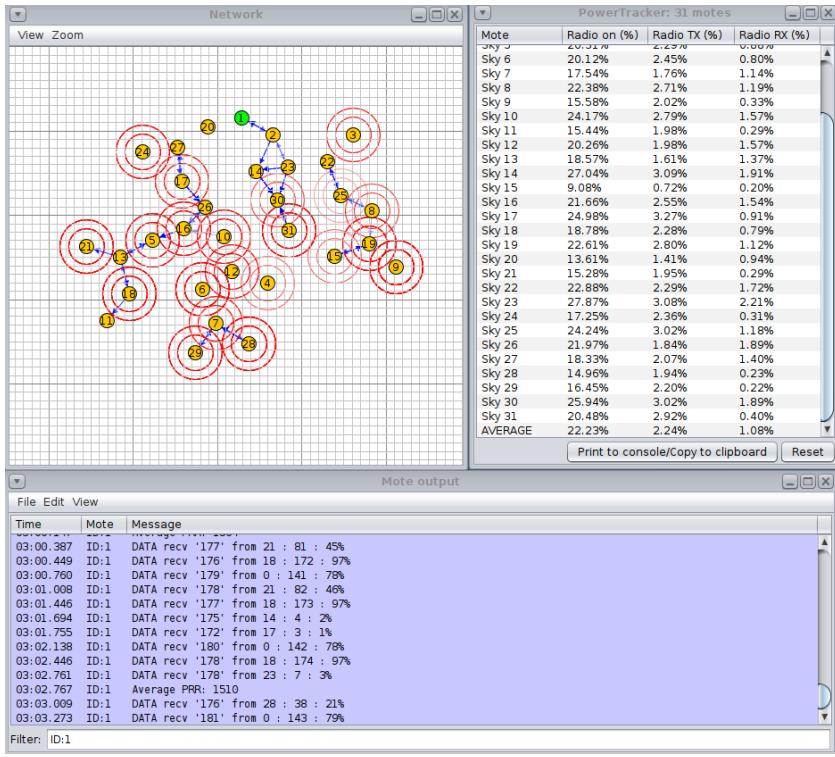


Figure 7. Cxmac - Loose network - High traffic

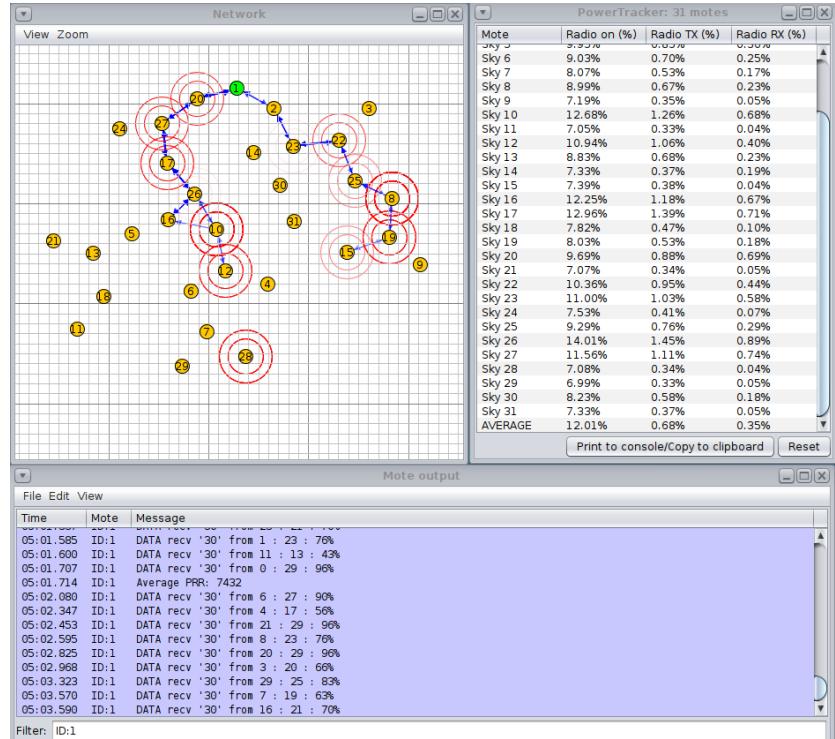


Figure 8. Cxmax - Loose network - Low traffic

Điều kiện kiểm tra:

```
#define NETSTACK_CONF_MAC    csma_driver  
#define NETSTACK_CONF_RDC    <thông số thay đổi>  
#define NETSTACK_CONF_FRAMER framer_802154
```

Mạng có 31 nodes:

- 1 server
- 30 clients

High traffic: client gửi packet cho server với chu kì 1s.

Low traffic: client gửi packet cho server với chu kì 10s.

Dense network – High traffic

	Average RDC (%)	PRR (Packet reception)
Contiki mac	16.62	43.54
Xmac	25.91	24.73

Dense network – Low traffic

	Average RDC (%)	PRR (Packet reception)
Contiki mac	6.95	97.47
Xmac	11.43	79.86

Loose network – High traffic

	Average RDC (%)	PRR (Packet reception)
Contiki mac	20.32	22.53
Xmac	22.23	15.10

Loose network – Low traffic

	Average RDC (%)	PRR (Packet reception)
Contiki mac	9.29	94.29
Xmac	12.01	74.32

Nhân xét: Contiki mac cho RDC thấp hơn trong khi PRR cao hơn X-Mac

➔ Contiki mac tốt hơn X-Mac

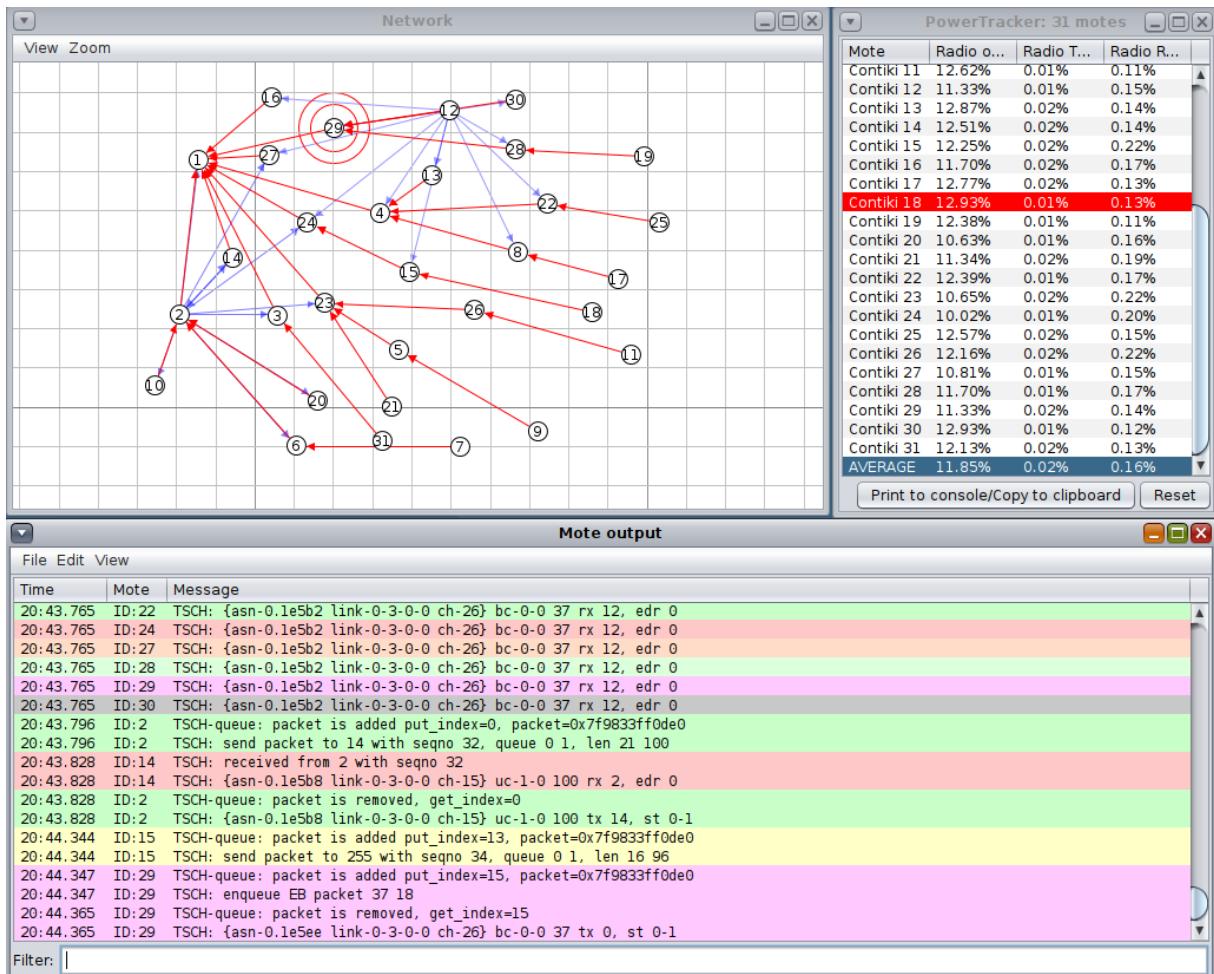
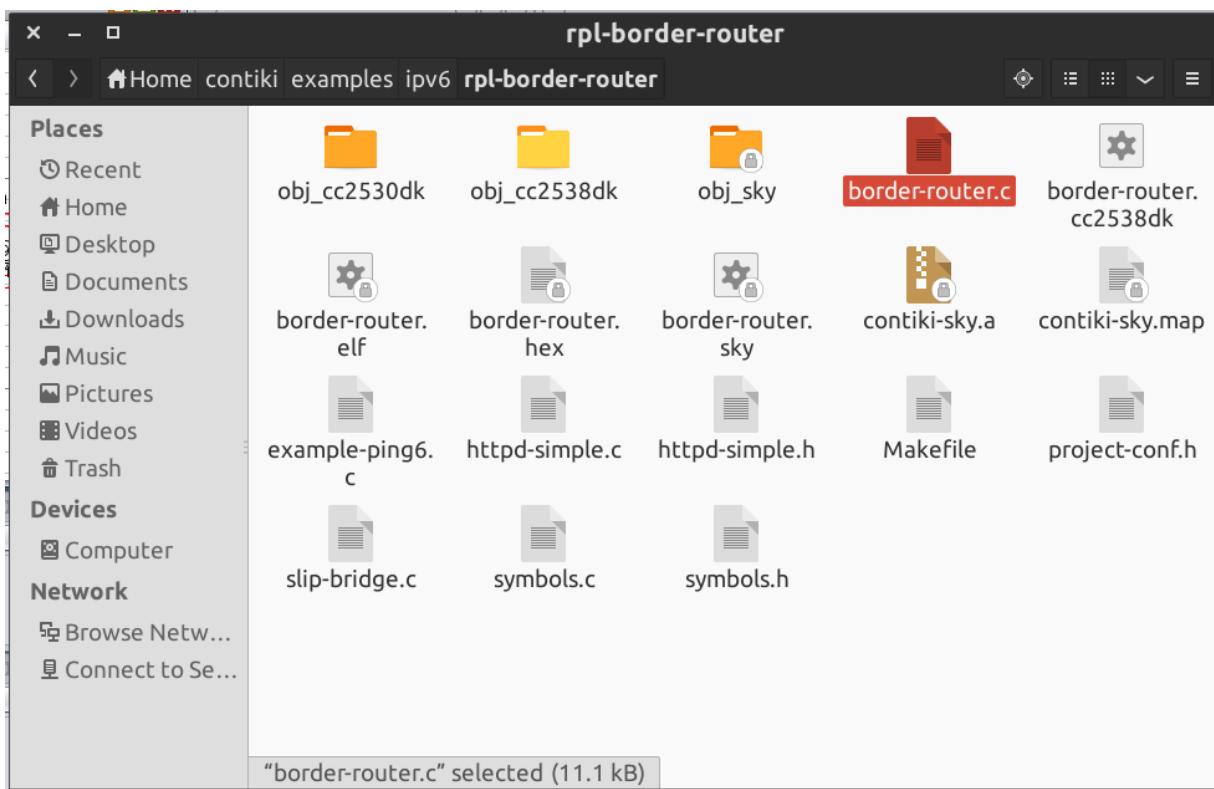


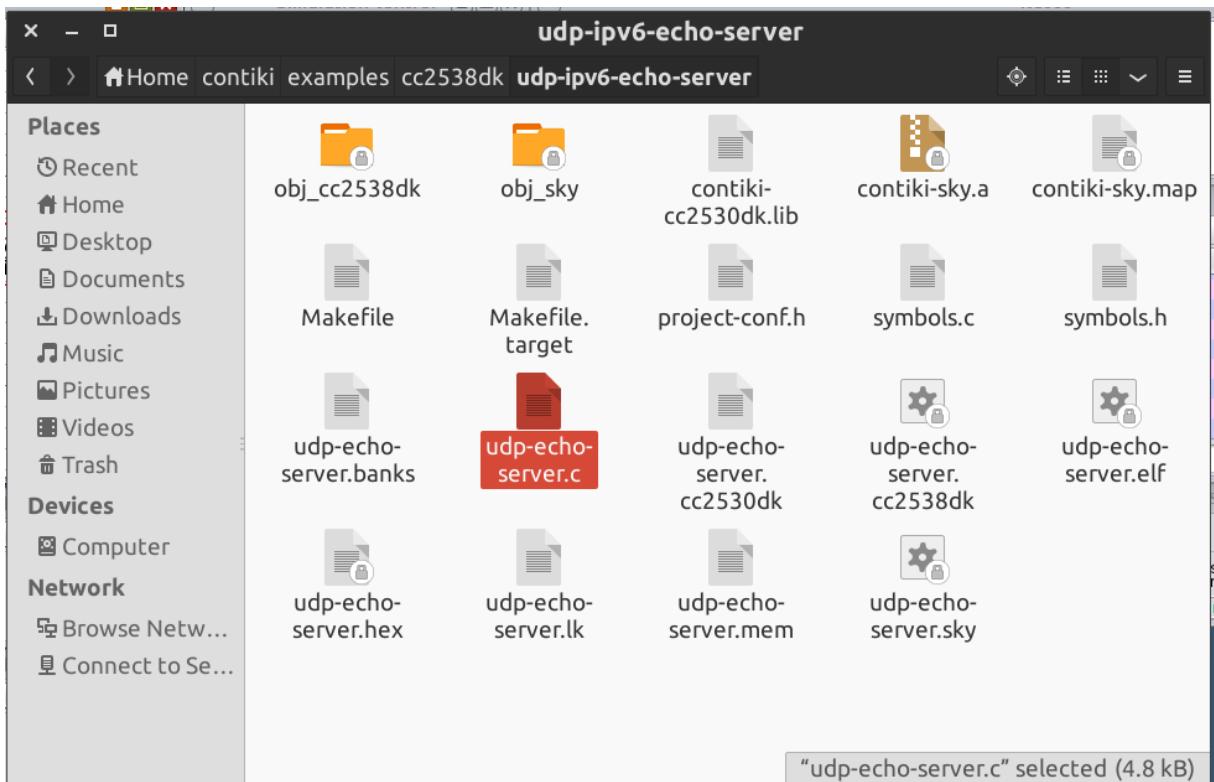
Figure 9. TSCH

IV. Assignment 4: 6LoWPAN Networks

Node 1 được nạp chương trình border_router.



Thực hiện chỉnh sửa code trong thư mục udp-ipv6-echo-server.



Khi node nhận data từ netcat sẽ được lưu trữ ở biến uip_appdata (đây là biến con trỏ kiểu Void), ta dùng hàm memcpy(buf, uip_appdata, len) lấy data đã nhận và ép kiểu dữ liệu thành chuỗi kí tự Char. Ở đây data đã được lừa vào biến buf.

```
udp-echo-server.c x
tcp-echo-server(vvto)
{
    memset(buf, 0, MAX_PAYLOAD_LEN);
    if(uip_newdata())
    {
        char a = 0;
        len = uip_datalen();
        memcpy(buf, uip_appdata, len);
        if(strcmp(buf, "led_red_on\n") == 0)
        {
            leds_on(LEDS_RED);
            a=1;
        }
        else if (strcmp(buf, "led_red_off\n") == 0)
        {
            leds_off(LEDS_RED);
            a=1;
        }
        else if (strcmp(buf, "led_green_on\n") == 0)
        {
            leds_on(LEDS_GREEN);
            a=1;
        }
        else if (strcmp(buf, "led_green_off\n") == 0)
        {
            a=1;
            leds_off(LEDS_GREEN);
        }
        else if (strcmp(buf, "led_blue_on\n") == 0)
        {
            leds_on(LEDS_BLUE);
            a=1;
        }
        else if (strcmp(buf, "led_blue_off\n") == 0)
        {
            leds_off(LEDS_BLUE);
        }
    }
}
C - Tab Width: 8 - Ln 76, Col 1 INS
```

Dùng hàm strcmp() để so sánh buf và các lệnh command để thực thi bật tắt led.

```
udp-echo-server.c x
char a = 0;
len = uip_datalen();
memcpy(buf, uip_appdata, len);
if(strcmp(buf, "led_red_on\n") == 0)
{
    leds_on(LEDS_RED);
    a=1;
}
else if (strcmp(buf, "led_red_off\n") == 0)
{
    leds_off(LEDS_RED);
    a=1;
}
else if (strcmp(buf, "led_green_on\n") == 0)
{
    leds_on(LEDS_GREEN);
    a=1;
}
else if (strcmp(buf, "led_green_off\n") == 0)
{
    a=1;
    leds_off(LEDS_GREEN);
}
else if (strcmp(buf, "led_blue_on\n") == 0)
{
    leds_on(LEDS_BLUE);
    a=1;
}
else if (strcmp(buf, "led_blue_off\n") == 0)
{
    leds_off(LEDS_BLUE);
    a=1;
}
else
    a=0;
}
C - Tab Width: 8 - Ln 76, Col 1 INS
```

Nếu data từ netcat trùng với các command: Note sẽ thực hiện bật tắt led và gửi về một messenger “Succeeded” để thông báo command đã thực thi.

Nếu data từ netcat khác với các command: Note sẽ gửi một messenger “Error” để thông báo command lỗi.

```

    a=1;
}
else
a=0;

PRINTF("%u bytes from [", len);
PRINT6ADDR(&UIP_IP_BUF->srcipaddr);
PRINTF("]:%u, command: %s", UIP_HTONS(UIP_UDP_BUF->srcport),buf);
uiplib_ipaddr_copy(&server_conn->ripaddr, &UIP_IP_BUF->srcipaddr);
server_conn->rport = UIP_UDP_BUF->srcport;
if (a == 1)
{
    uiplib_udp_packet_send(server_conn, "Succeeded\n",10 );
    uiplib_create_unspecified(&server_conn->ripaddr);
    server_conn->rport = 0;
}
else
{
    uiplib_udp_packet_send(server_conn, "Error\n",6 );
    uiplib_create_unspecified(&server_conn->ripaddr);
    server_conn->rport = 0;
}
//leds_off(LEDS_RED);
return;
}/*-----*/

```

Note sẽ mở Port 3000 lắng nghe các kết nối UDP từ netcat.

```

PROCESS_THREAD(udp_echo_server_process, ev, data)
{

PROCESS_BEGIN();
PRINTF("Starting UDP echo server\n");

server_conn = udp_new(NULL, UIP_HTONS(0), NULL);
udp_bind(server_conn, UIP_HTONS(3000));

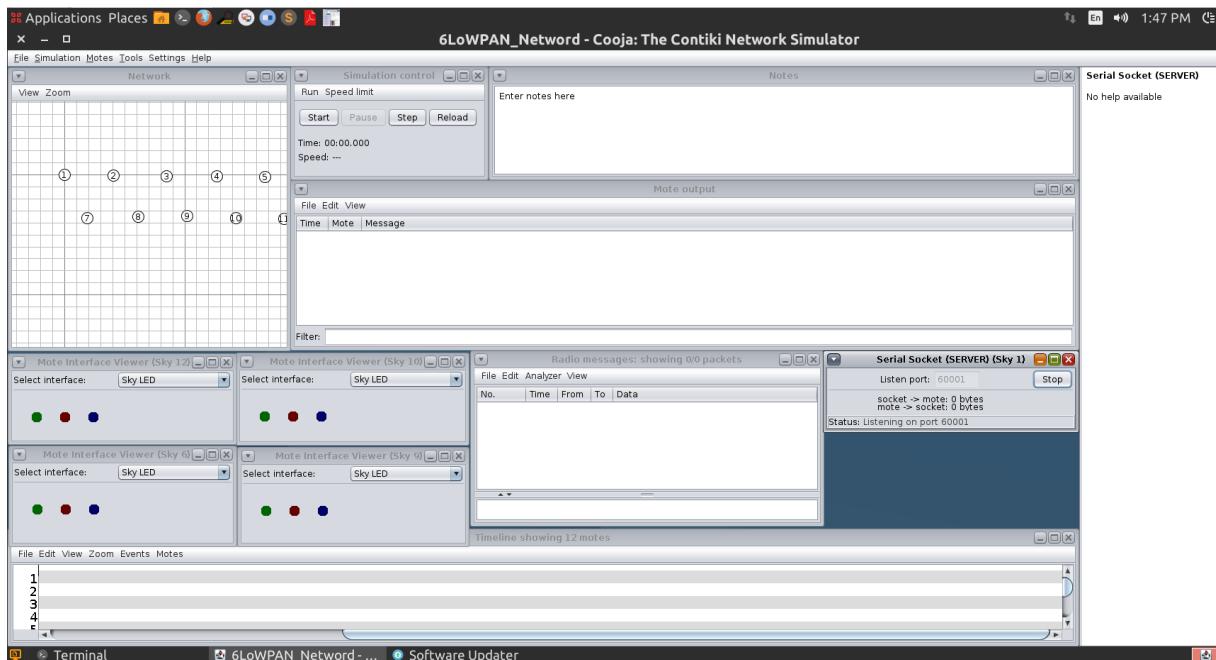
PRINTF("Listen port: 3000, TTL=%u\n", server_conn->ttl);

while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
        tcpip_handler();
    }
}

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

```

Kết quả mô phỏng.



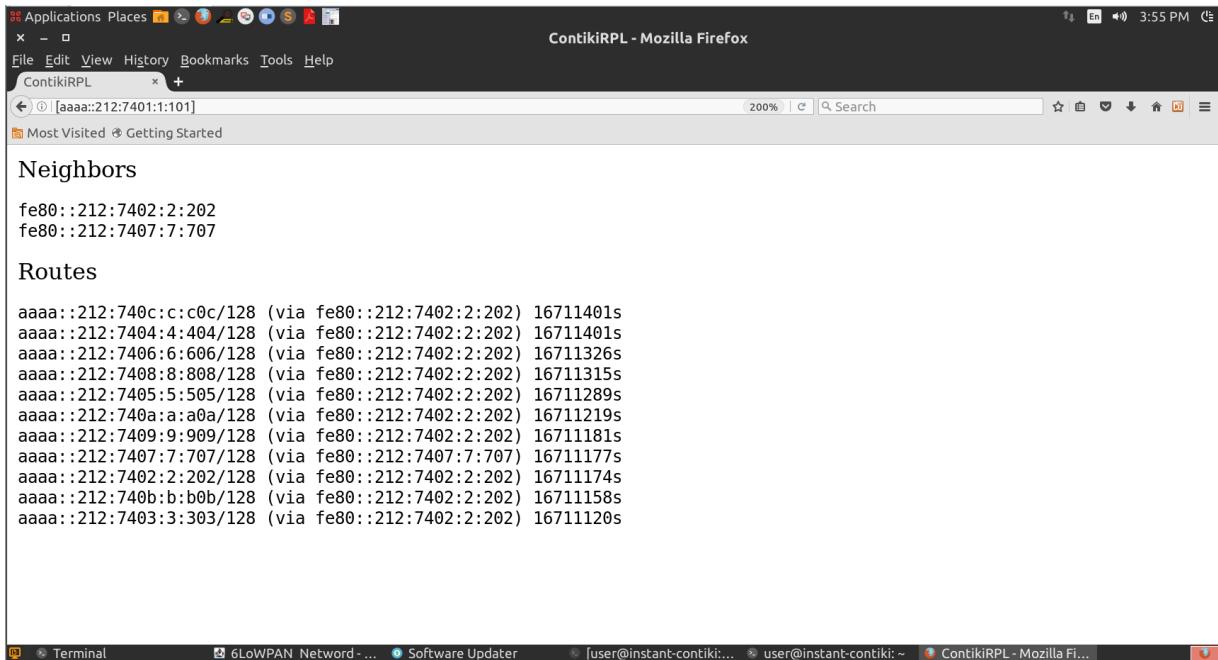
13

```

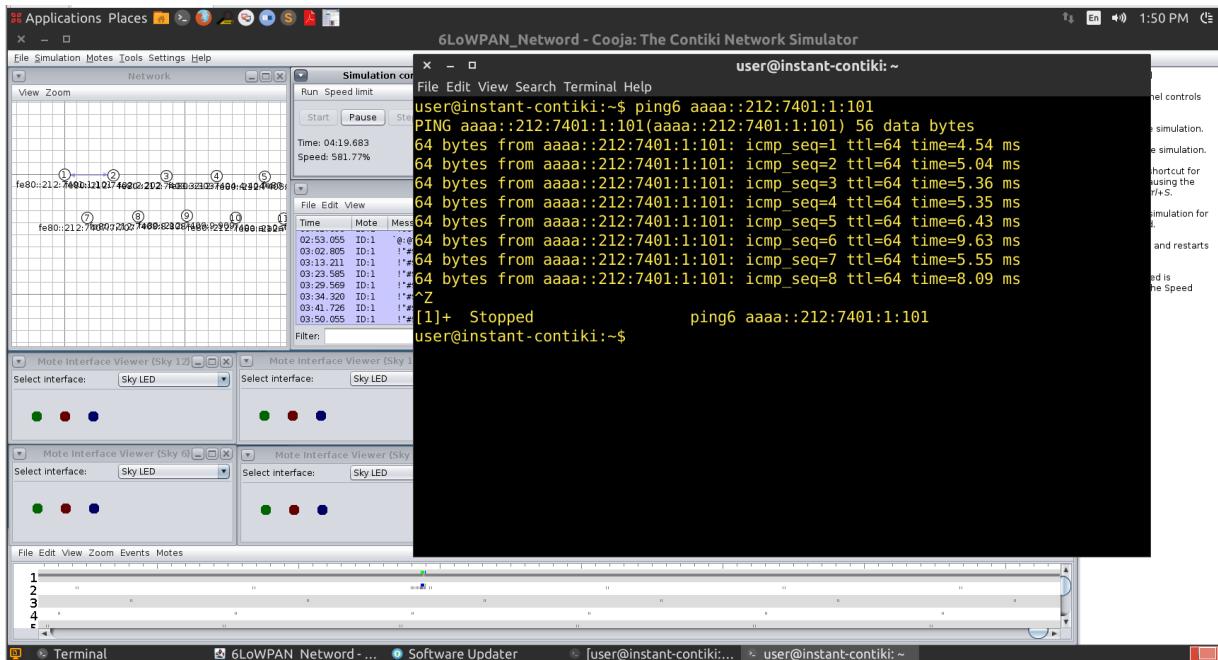
Applications Places
File Edit View Search Terminal Help
user@instant-contiki: ~/contiki/examples/ipv6/rpl-border-router
File Edit View Zoom Events Notes
1
2
3
4
Terminal 6LoWPAN Network - ... Software Updater
user@instant-contiki:~$ cd contiki/examples/ipv6/rpl-border-router/
user@instant-contiki:~/contiki/examples/ipv6/rpl-border-router$ make connect-router-cooja
TARGET not defined, using target 'native'
sudo ../../tools/tunslip6 -a 127.0.0.1 aaaa::1/64
[sudo] password for user:
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::0:0:1/64
ifconfig tun0
tun0      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          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)

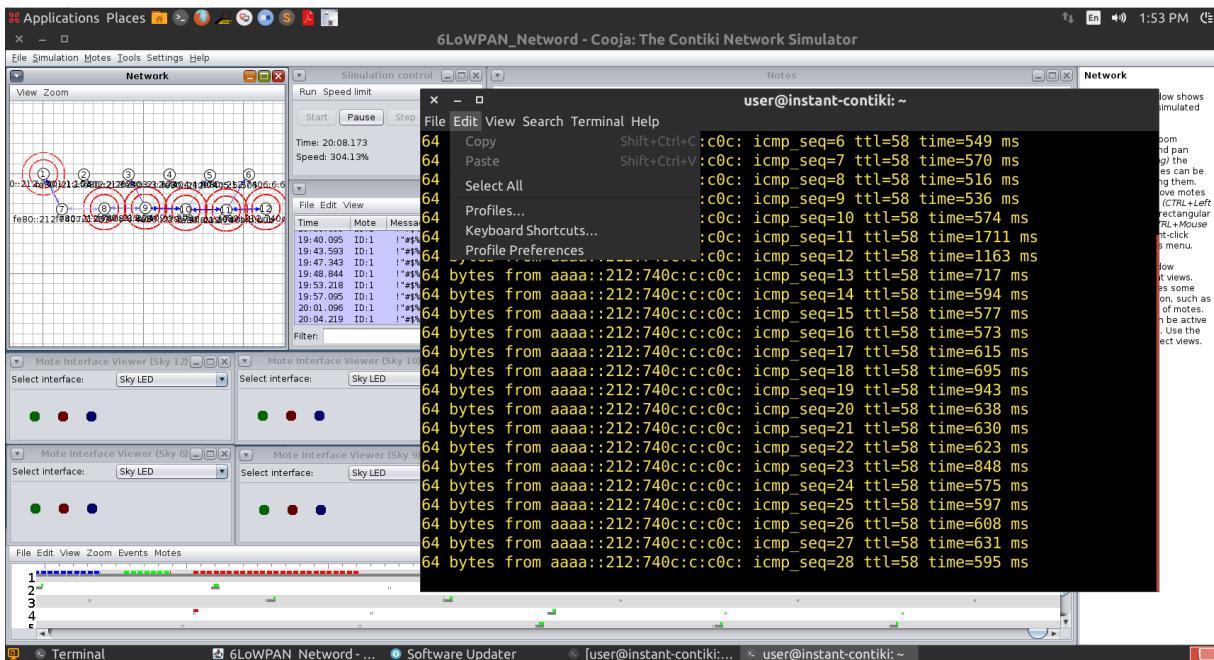
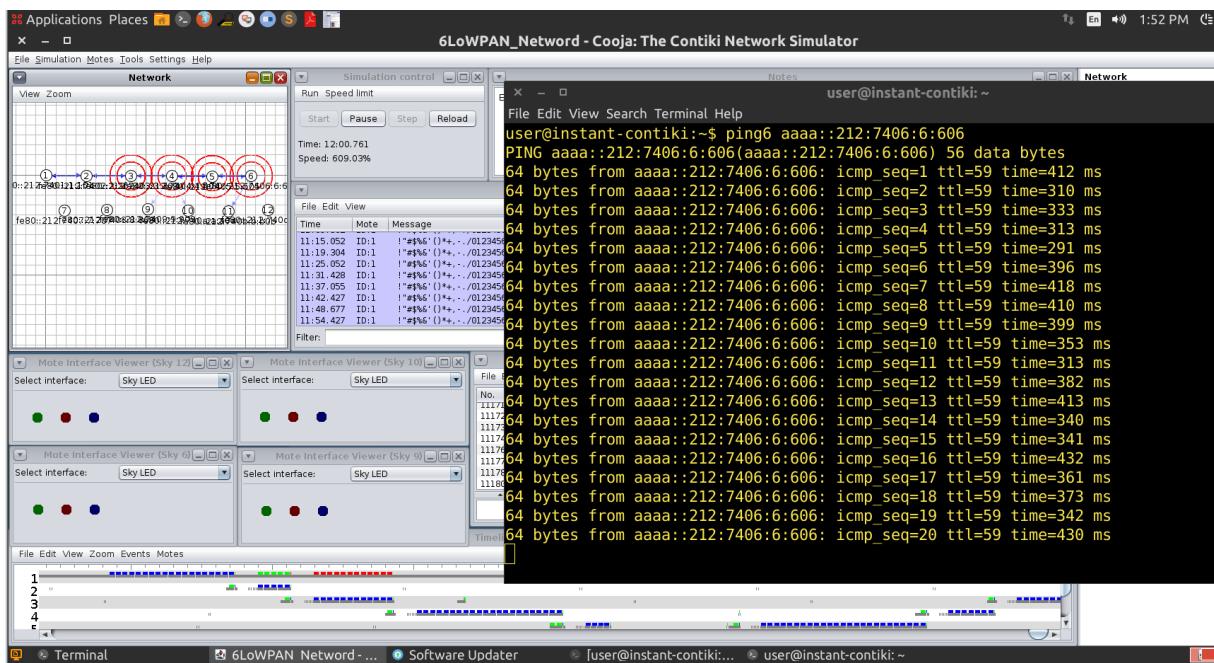
```

16



Sử dụng lệnh ping6 để ping đến các Node.





Thực hiện điều khiển Led bằng command truyền qua netcat.

