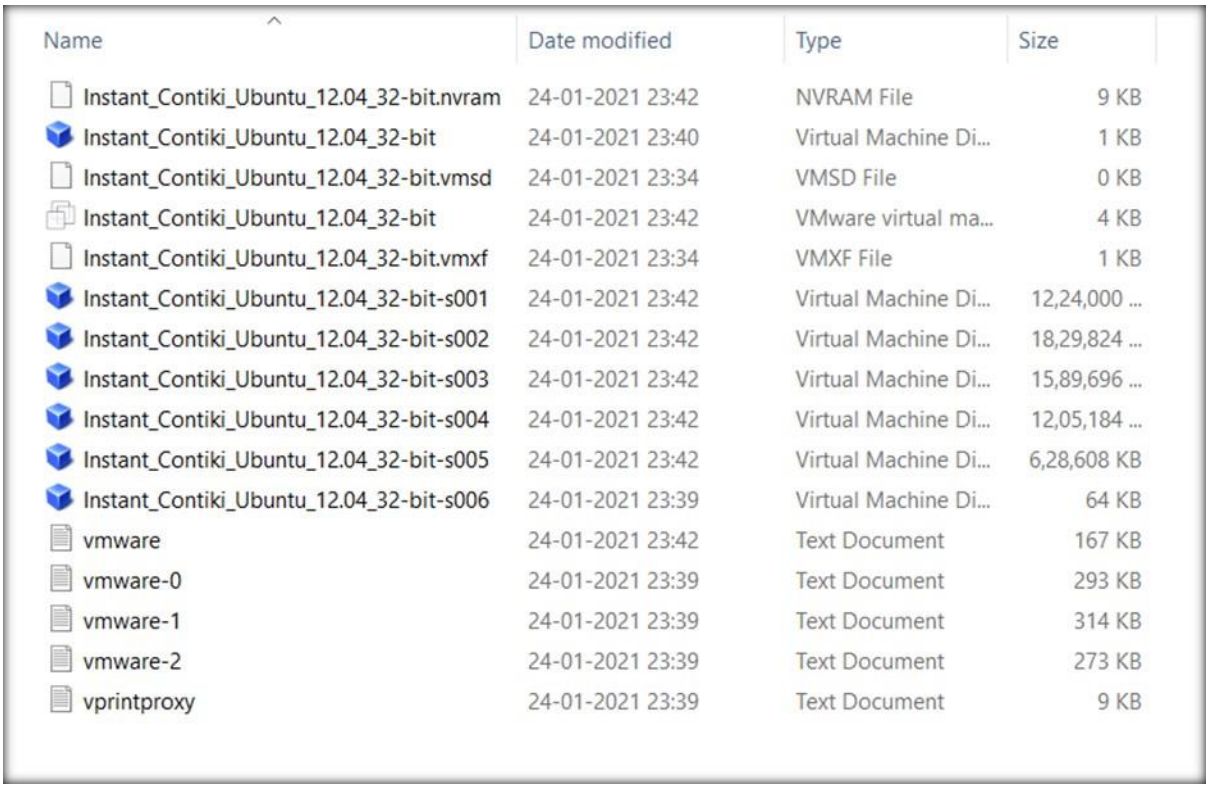
PRACTICAL - 1

AIM: Installation and configuration of Instant Contiki OS with Cooja.

THEORY:

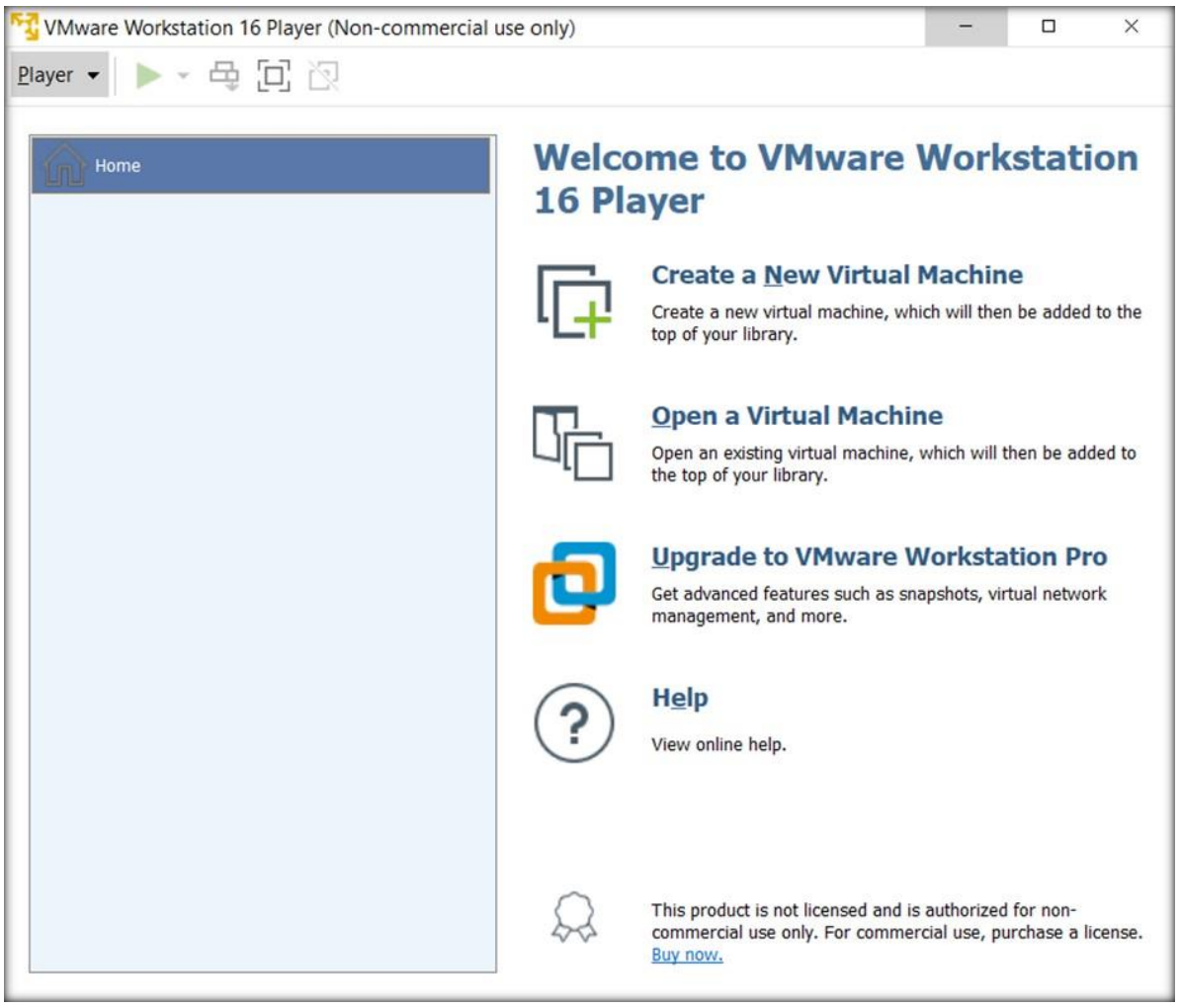
The following steps show how to install Contiki OS in VMware.

- Download the required files of Contiki OS from https://sourceforge.net/projects/contiki/
- Extract all the files that you downloaded.

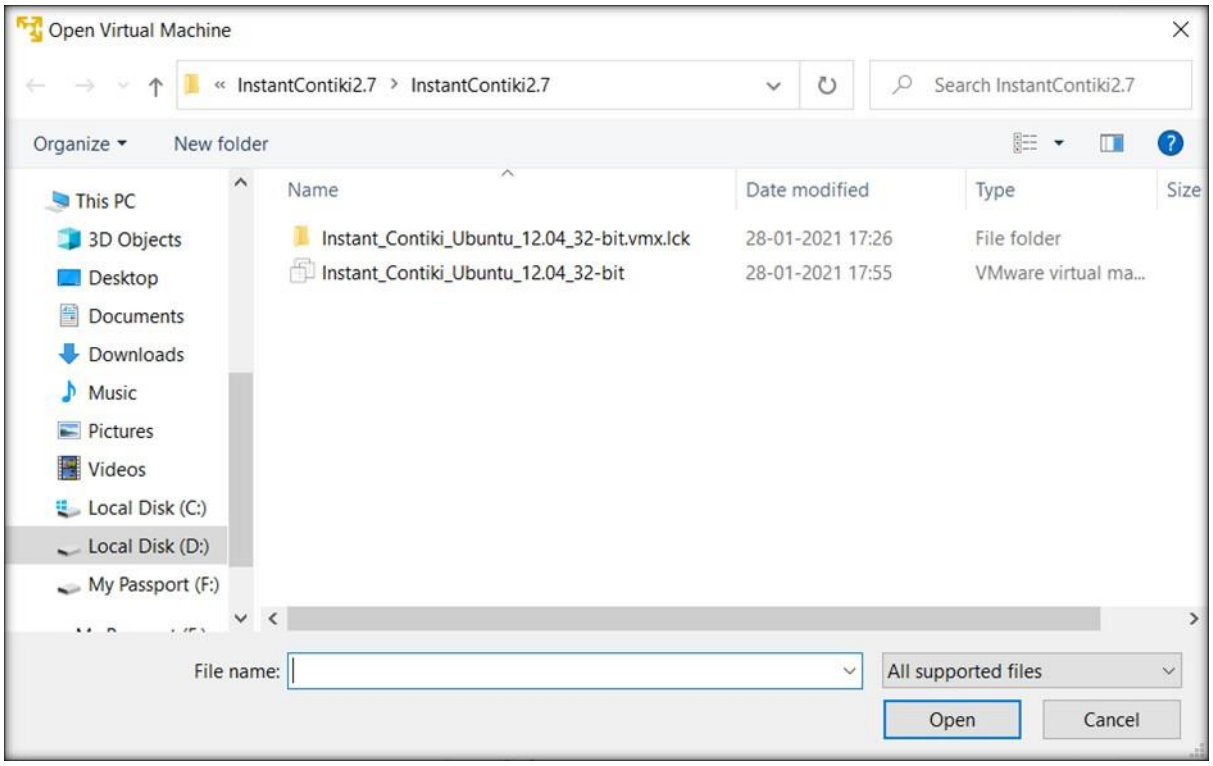


Name	Date modified	Type	Size
Instant_Contiki_Ubuntu_12.04_32-bit.nvram	24-01-2021 23:42	NVRAM File	9 KB
Instant_Contiki_Ubuntu_12.04_32-bit	24-01-2021 23:40	Virtual Machine Di...	1 KB
Instant_Contiki_Ubuntu_12.04_32-bit.vmsd	24-01-2021 23:34	VMSD File	0 KB
Instant_Contiki_Ubuntu_12.04_32-bit	24-01-2021 23:42	VMware virtual ma...	4 KB
Instant_Contiki_Ubuntu_12.04_32-bit.vmxf	24-01-2021 23:34	VMXF File	1 KB
Instant_Contiki_Ubuntu_12.04_32-bit-s001	24-01-2021 23:42	Virtual Machine Di...	12,24,000 ...
Instant_Contiki_Ubuntu_12.04_32-bit-s002	24-01-2021 23:42	Virtual Machine Di...	18,29,824 ...
Instant_Contiki_Ubuntu_12.04_32-bit-s003	24-01-2021 23:42	Virtual Machine Di...	15,89,696 ...
Instant_Contiki_Ubuntu_12.04_32-bit-s004	24-01-2021 23:42	Virtual Machine Di...	12,05,184 ...
Instant_Contiki_Ubuntu_12.04_32-bit-s005	24-01-2021 23:42	Virtual Machine Di...	6,28,608 KB
Instant_Contiki_Ubuntu_12.04_32-bit-s006	24-01-2021 23:39	Virtual Machine Di...	64 KB
vmware	24-01-2021 23:42	Text Document	167 KB
vmware-0	24-01-2021 23:39	Text Document	293 KB
vmware-1	24-01-2021 23:39	Text Document	314 KB
vmware-2	24-01-2021 23:39	Text Document	273 KB
vprintproxy	24-01-2021 23:39	Text Document	9 KB

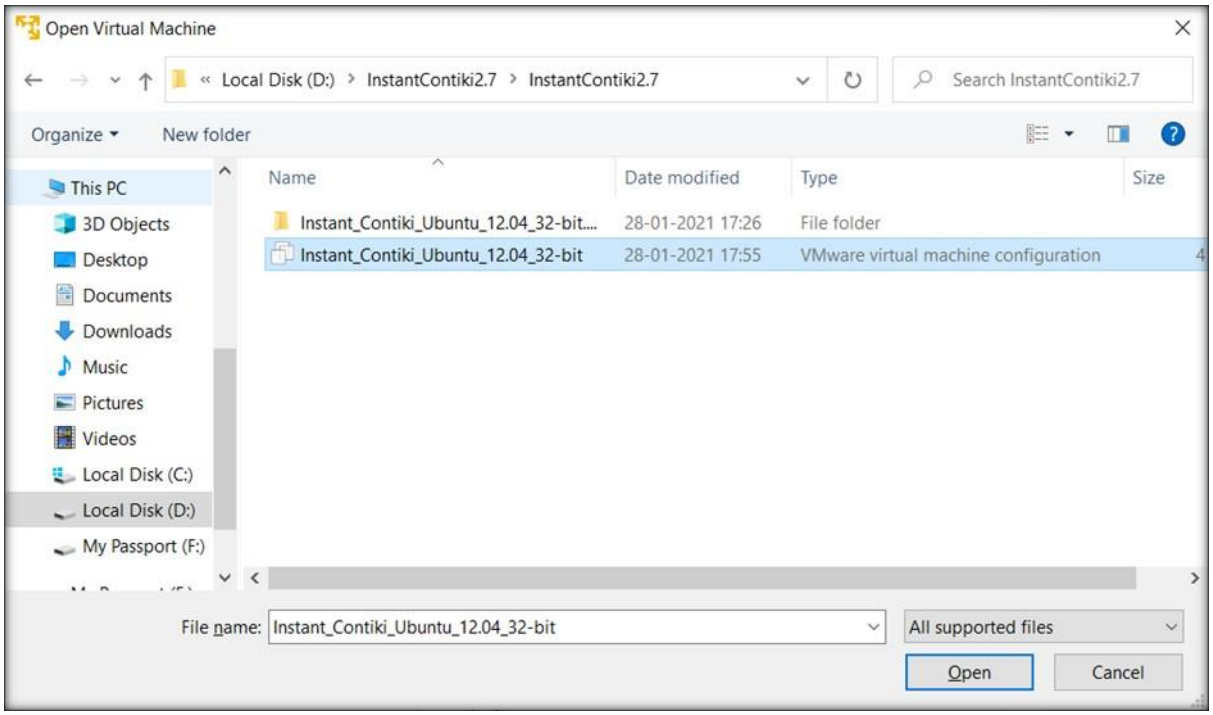
- Download and install VMWare.



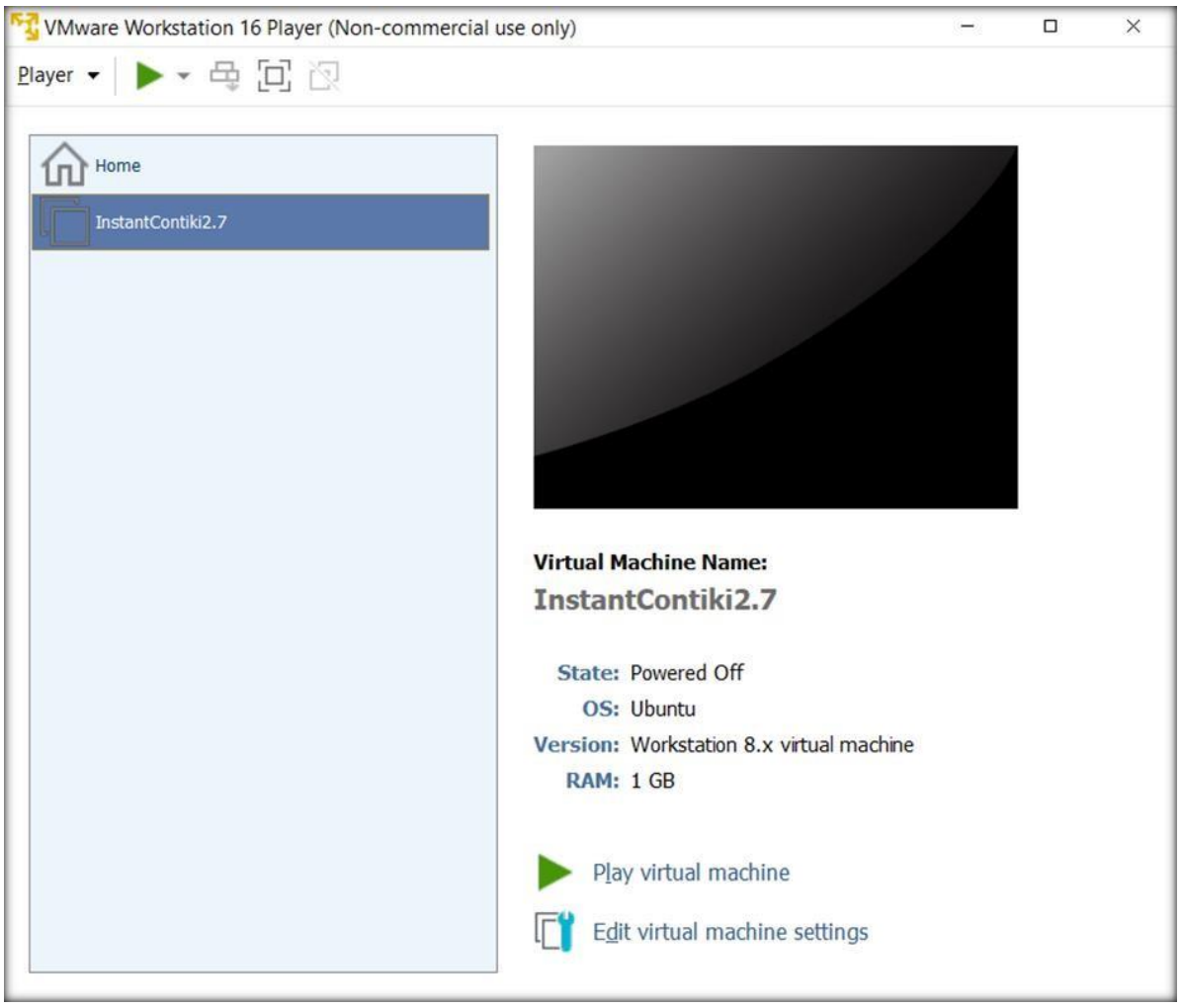
- Choose Open a Virtual Machine and explore to the folder where instant Contiki files are situated.



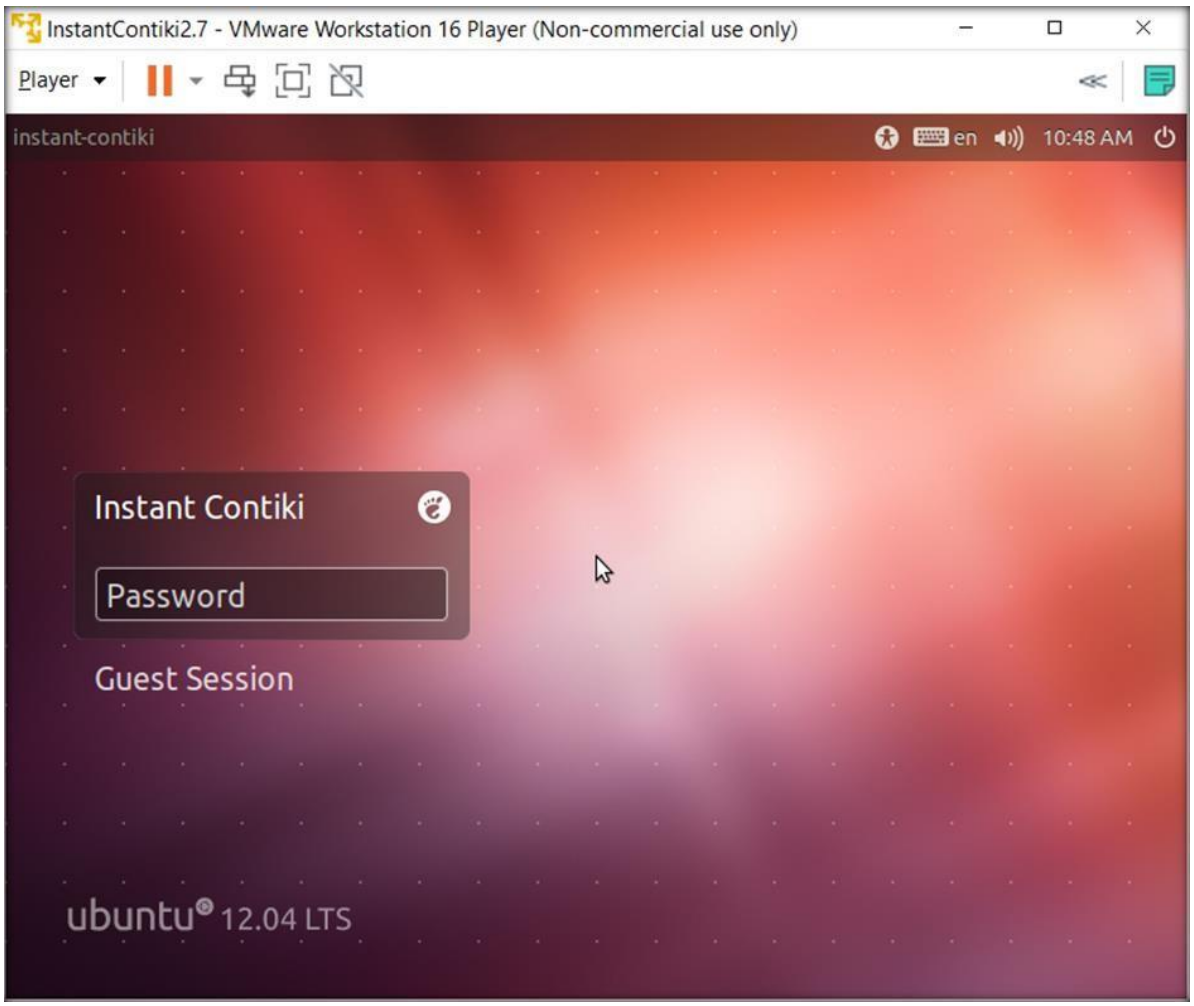
- Select VMware virtual machine configuration file and open it.



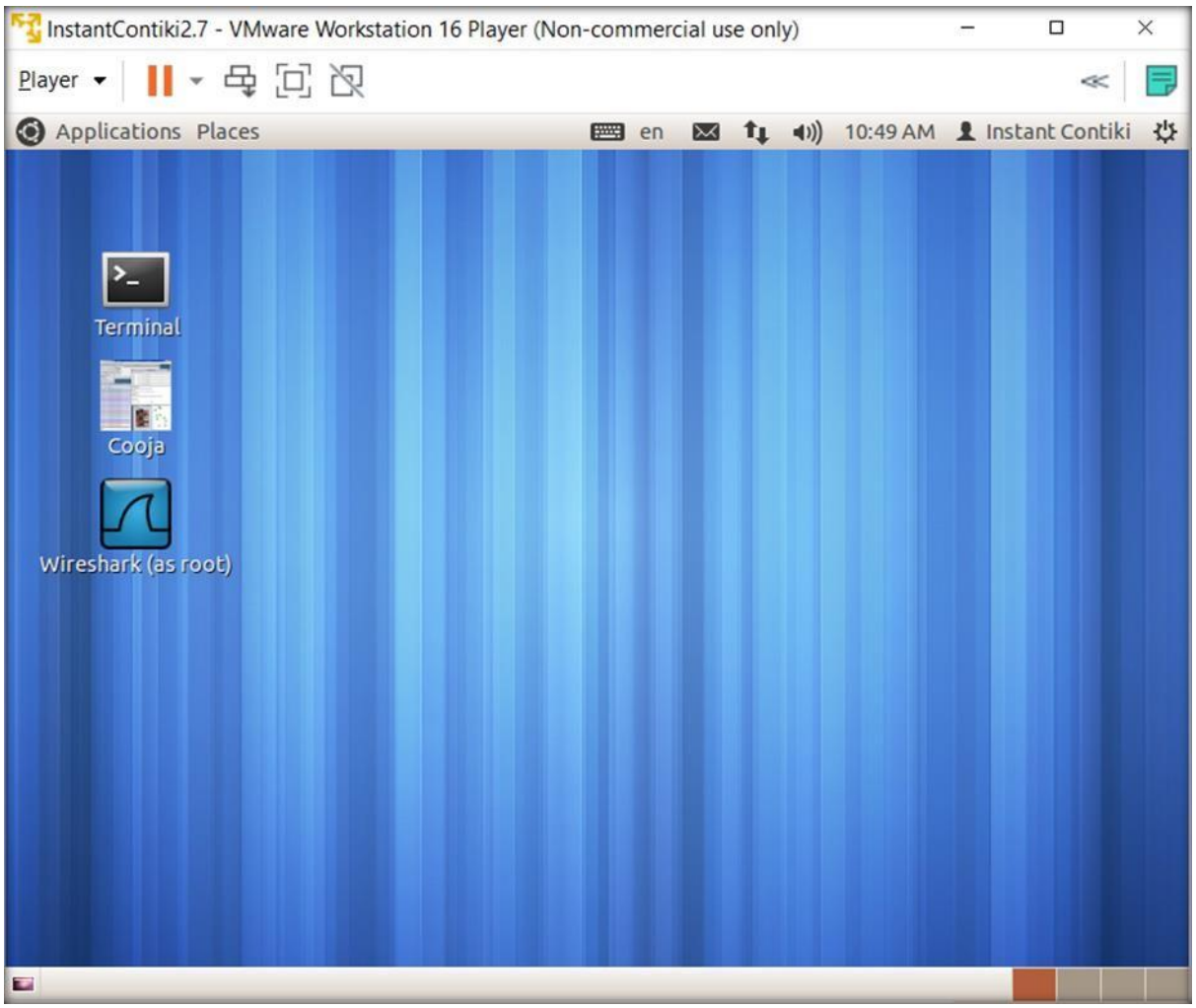
- On opening that file, new virtual machine will be created containing the given Contiki OS.



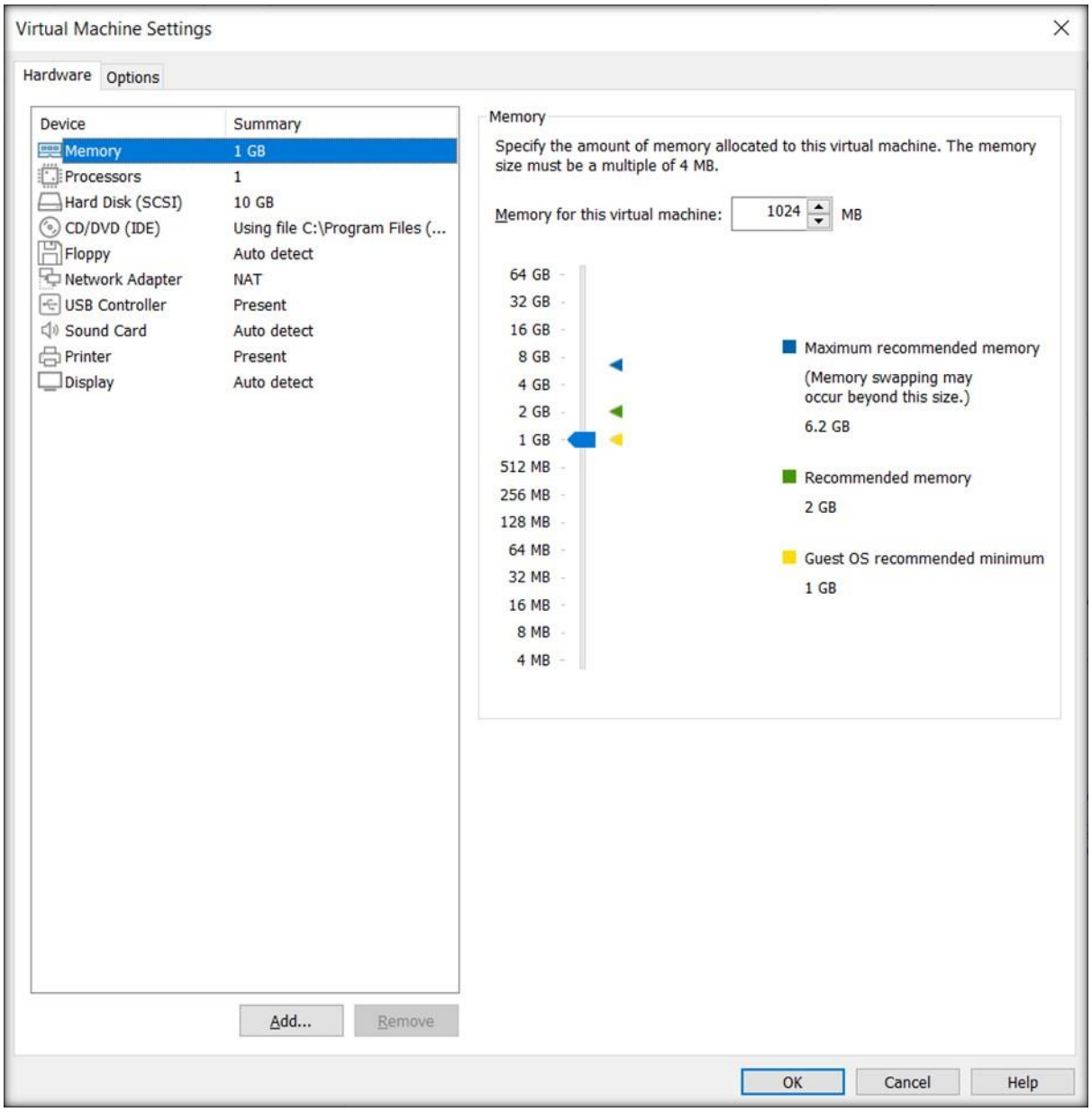
- To start Contiki OS, click on “Play virtual machine”



- The default password is “user” and Desktop will be seen.



- We can change the settings of virtual machine by clicking “edit virtual machine settings”



CONCLUSION

In this practical, we learned about how to install Contiki OS in VMware and change some settings of virtual machine.

PRACTICAL - 2

AIM: Study of different types of motes and deploy them using IoT architecture.

THEORY:

Cooja provides variety of motes to work with.
Few famous motes are described below.

- micaZ mote:
  - The MICAz is a 2.4 GHz Mote module used for enabling low-power, wireless sensor networks.
  - Supported by MoteWorks™ wireless sensor network platform for reliable, ad-hoc mesh networking
  - MoteWork enables the development of custom sensor applications and is specifically optimized for low-power, battery-operated networks.
  - MoteWorks is based on the open-source TinyOS operating system and provides reliable, ad-hoc mesh networking, over-theair-programming capabilities, cross development tools, server middleware for enterprise network integration and client user interface for analysis and a configuration.

- sky mote:
  - Tmote Sky is an ultra low power wireless module for use in sensor networks, monitoring applications, and rapid application prototyping.
  - Tmote Sky leverages industry standards like USB and IEEE 802.15.4 to interoperate seamlessly with other devices.
  - By using industry standards, integrating humidity, temperature, and light sensors, and providing flexible interconnection with peripherals, Tmote Sky enables a wide range of mesh network applications.
  - Tmote Sky is a drop-in replacement for Moteiv's successful Telos design.
  - Tmote Sky includes increased performance, functionality, and expansion.
  - With TinyOS support out-of-the-box, Tmote Sky leverages emerging wireless protocols and the open source software movement.
  - Tmote Sky is part of a line of modules featuring on-board sensors to increase robustness while decreasing cost and package size.

- ESB
  - The ESB (Embedded Sensor Board) is a prototype wireless sensor network device developed at FU Berlin.
  - The ESB consists of a Texas Instruments MSP430 low-power microcontroller with 2k RAM and 60k flash ROM, a TR1001 radio transceiver, a 32k serial EEPROM, an RS232 port, a JTAG port, a beeper, and a number of sensors (passive IR, active IR sender/receiver, vibration/tilt, microphone, temperature).
  - The Contiki/ESB port contains drivers for most of the sensors. The drivers were mostly adapted from sources from FU Berlin.

- CC430 mote:
  - The TI CC430 family of ultra-low-power system-on-chip (SoC) microcontrollers with integrated RF transceiver cores consists of several devices that feature different sets of peripherals targeted for a wide range of applications.
  - The architecture, combined with five low-power modes, is optimized to achieve extended battery life in portable measurement applications.
  - The devices feature the powerful MSP430 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency.
  - The CC430 family provides a tight integration between the microcontroller core, its peripherals, software, and the RF transceiver, making these true SoC solutions easy to use as well as improving performance.

- Z1 mote:
  - The Z1 module is a general purpose development platform for wireless sensor networks (WSN) designed for researchers, developers, enthusiasts and hobbyists.
  - It is a platform compatible with the successful Tmote-family motes with several enhancements that offers roughly a 2x performance in several aspects.

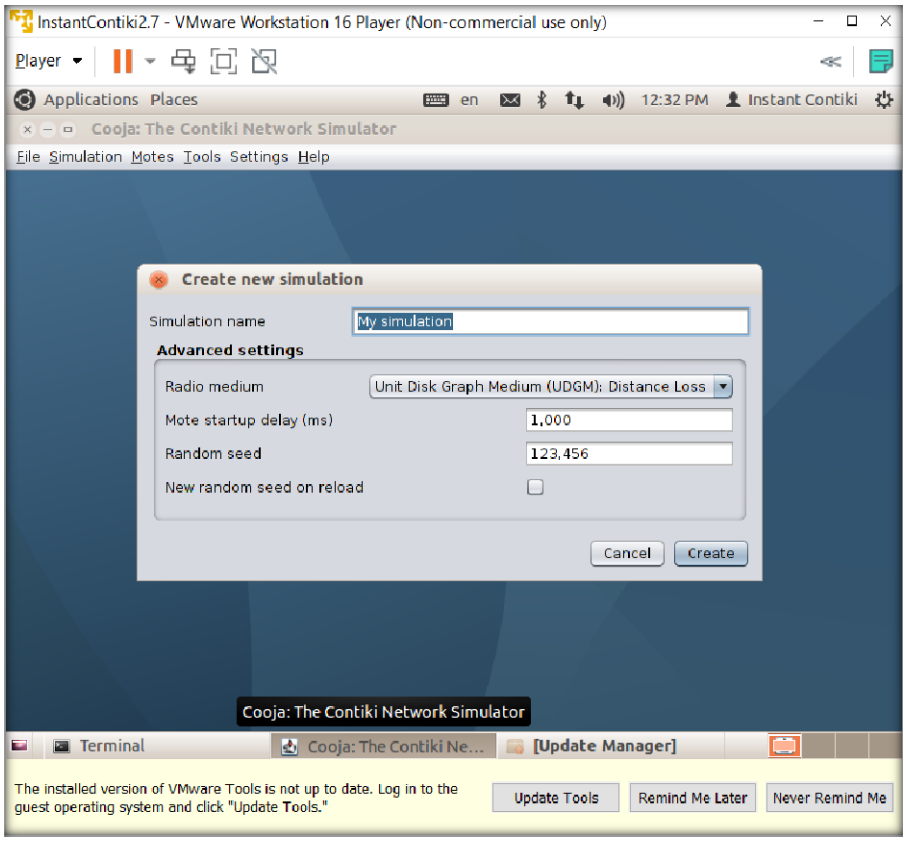
- Wismote mote:
  - WiSMote is a sensor/actuator module well adapted to Wireless Sensor Network (WSN) applications.
  - The wireless link operates over the 2.4GHz ISM, a duty free frequency band.
  - With its wide range of embedded sensors and its variety of extension connectors, WiSMote is able to monitor any kind of physical measurements in fields like environment, healthcare, domotics, smart building, logistics or industrial applications.
  - WiSMote embed an small footprint operating system (Contiki) plus an IEEE 802.15.4 protocol stack compatbile with Zigbee and 6LoWPAN (IPv6).

AIM: Simulate Hello World program using Cooja.

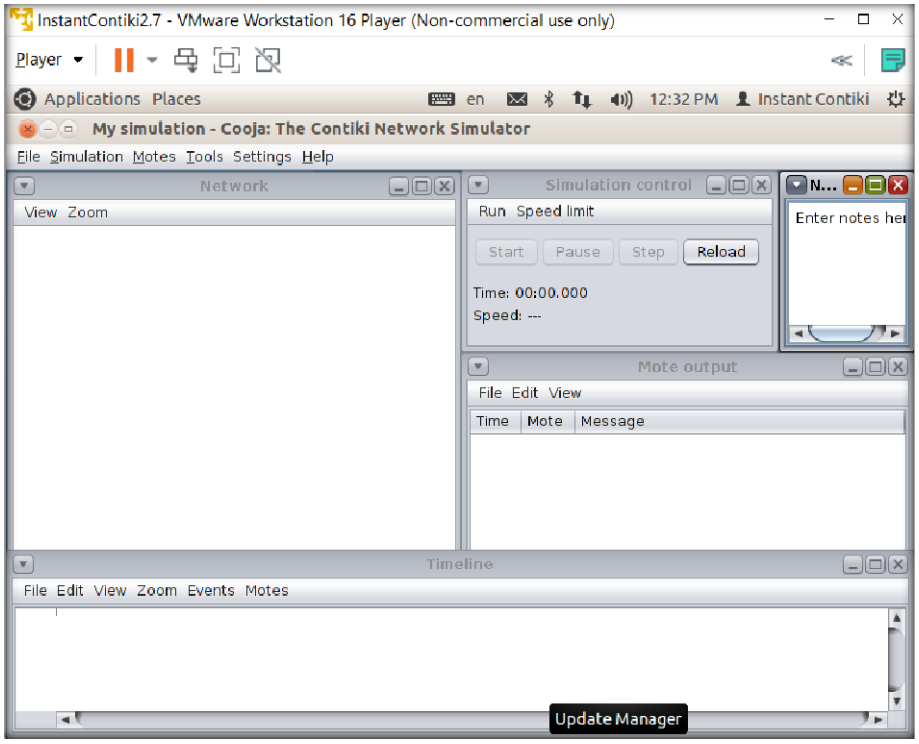
PROGRAM:

Open Cooja simulator.

Go to file > create new simulation

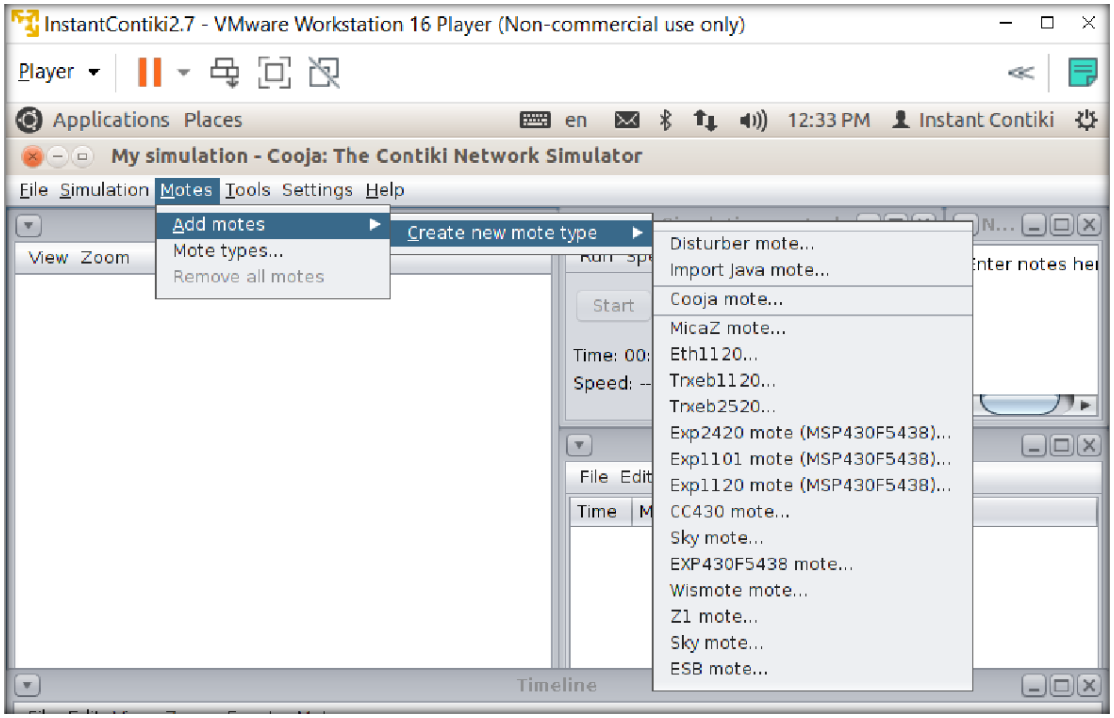


Give it any name you like and keep the rest of settings as default.

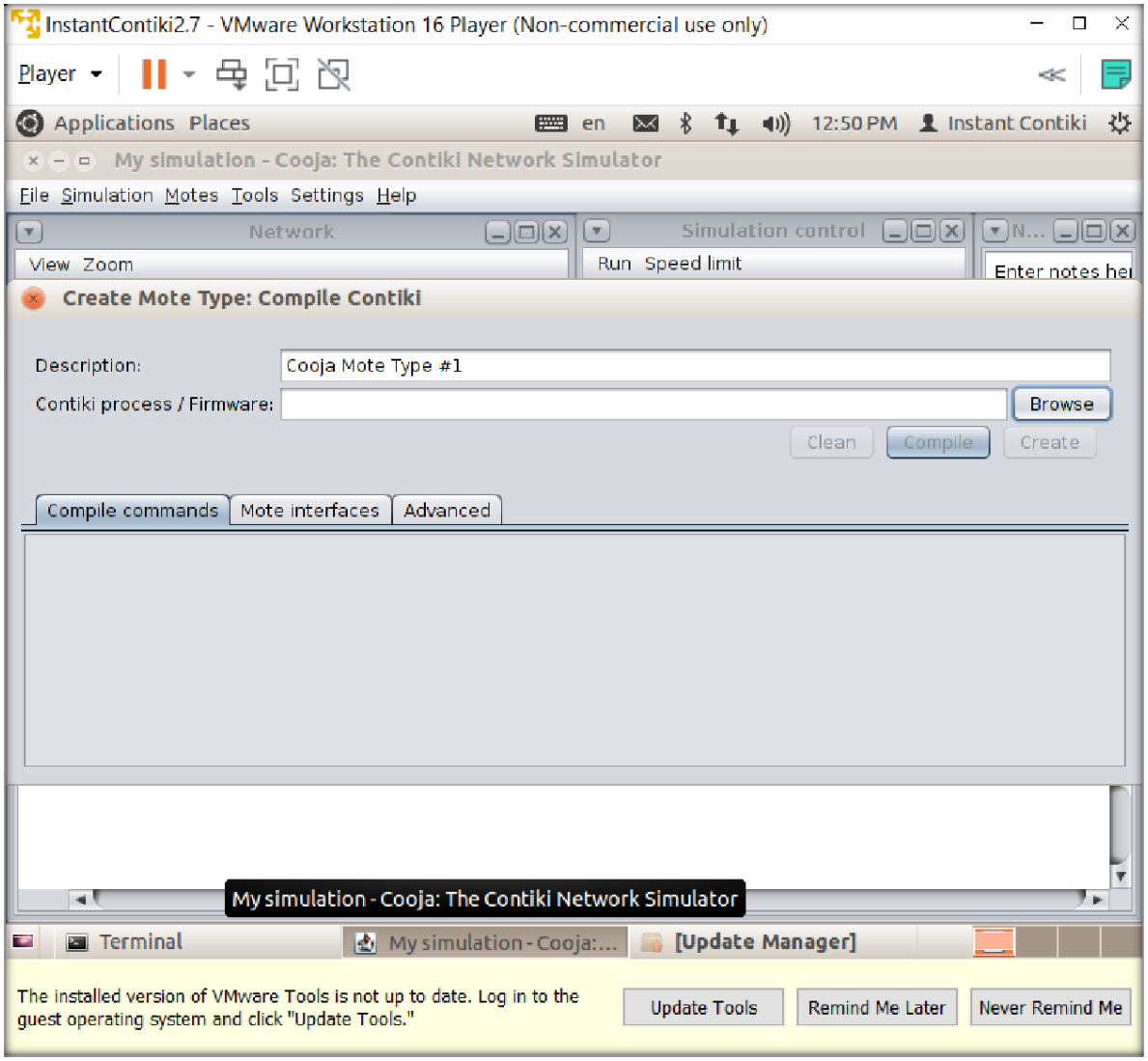


We will add 3 cooja motes.

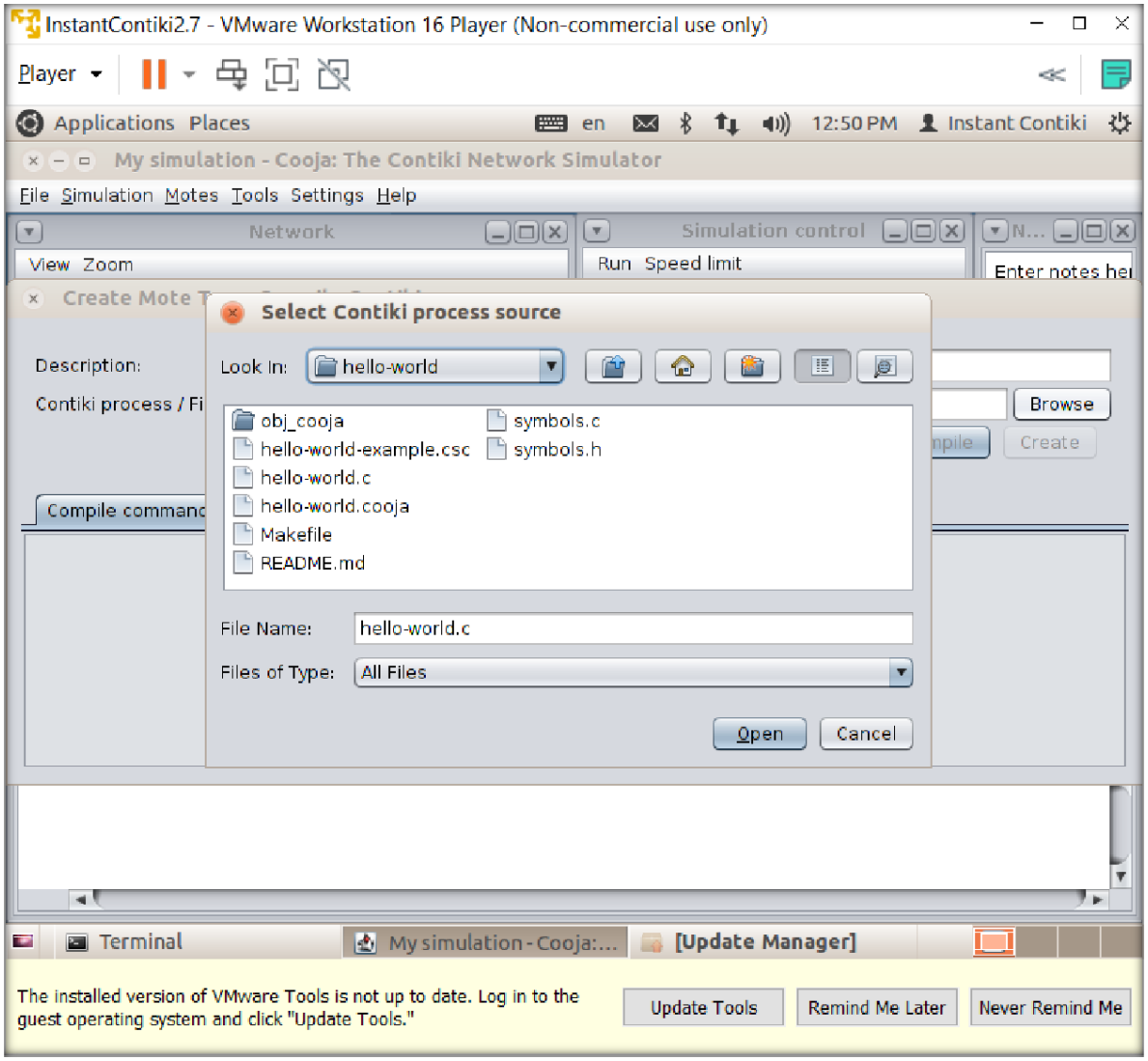
For that, go to Motes > Add motes > Create new mote type > Cooja mote.



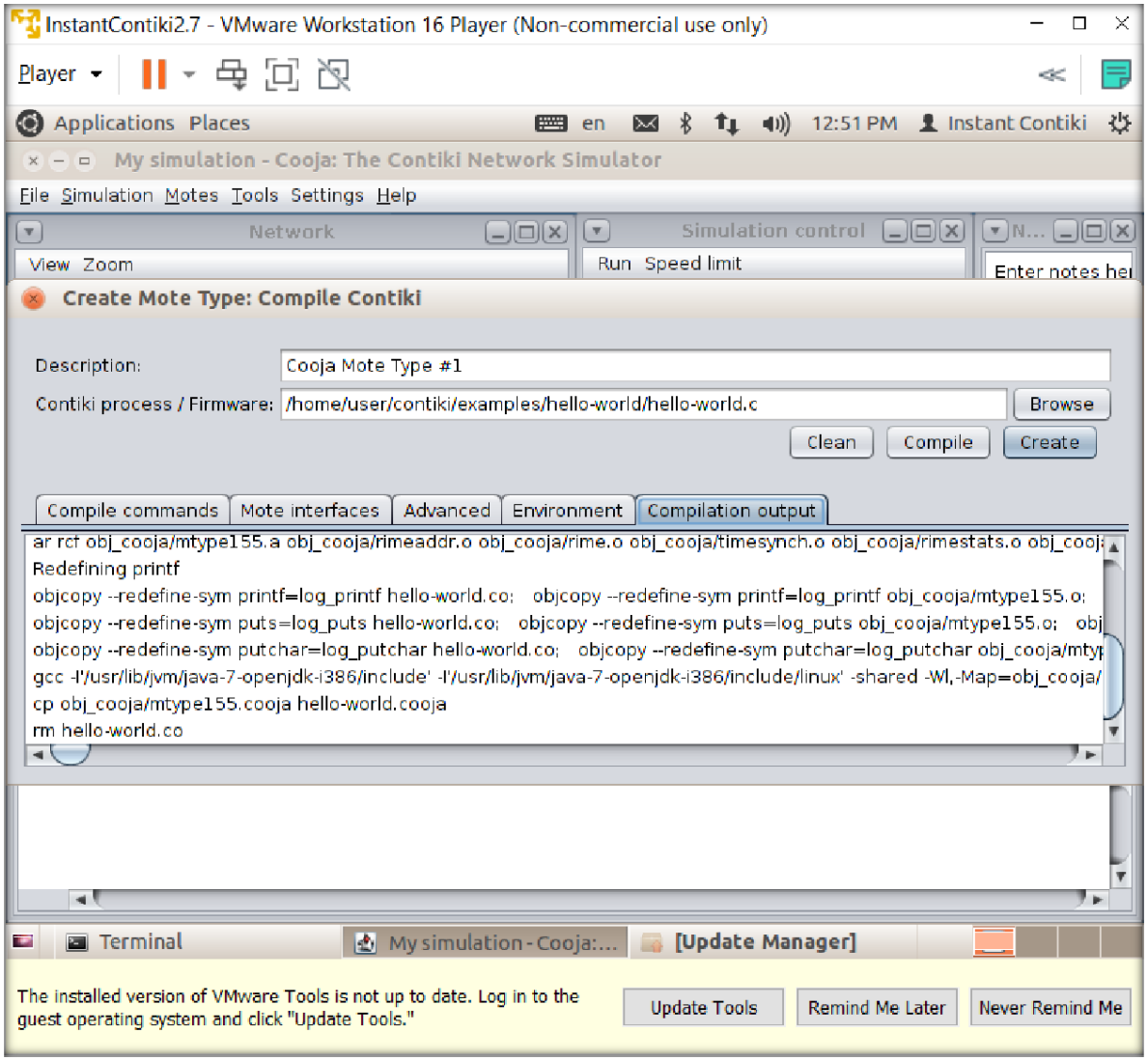
It will open a new window where we can select other properties of our motes.



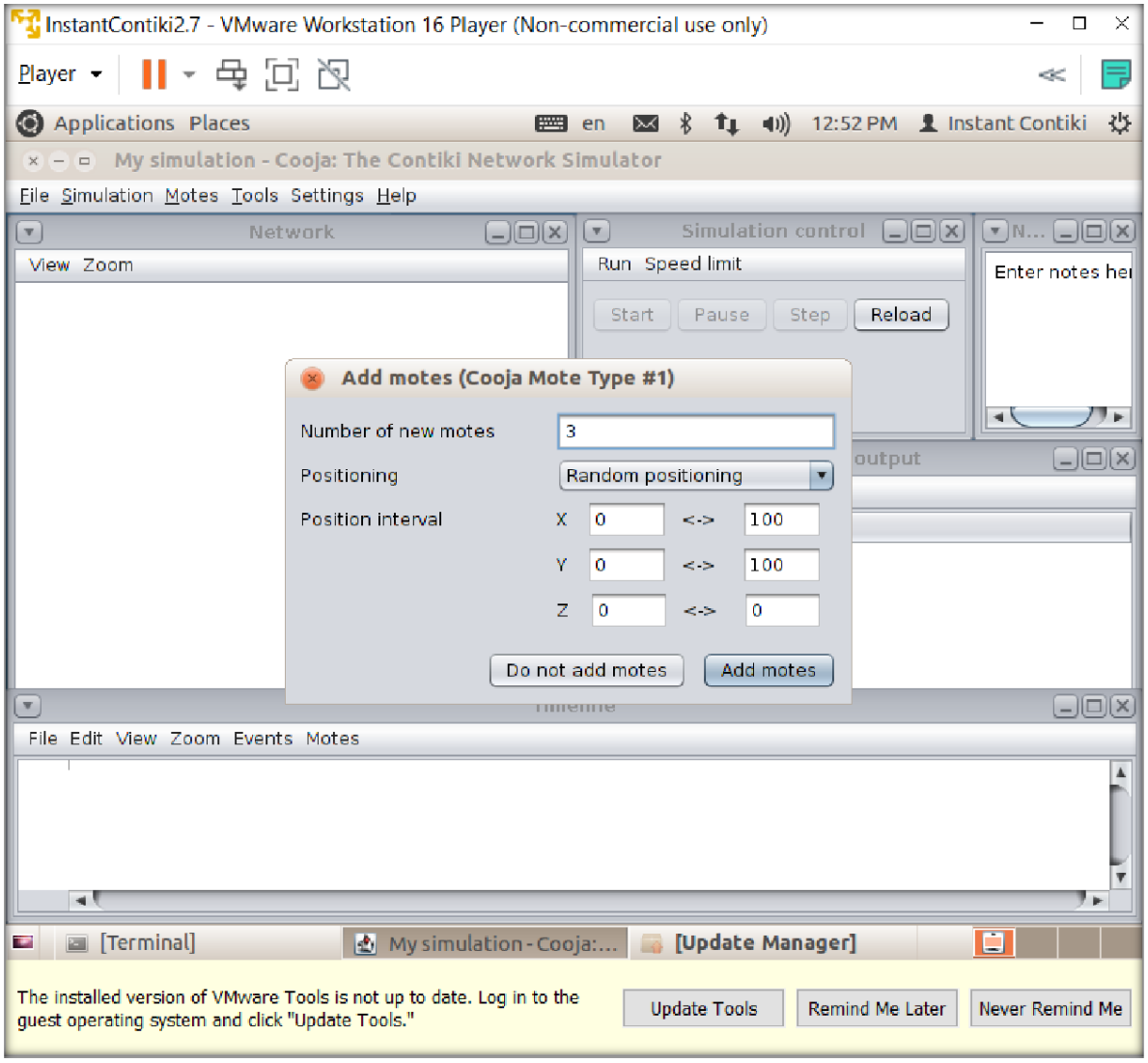
In Contiki process/firmware, browse to hello-world folder in examples.



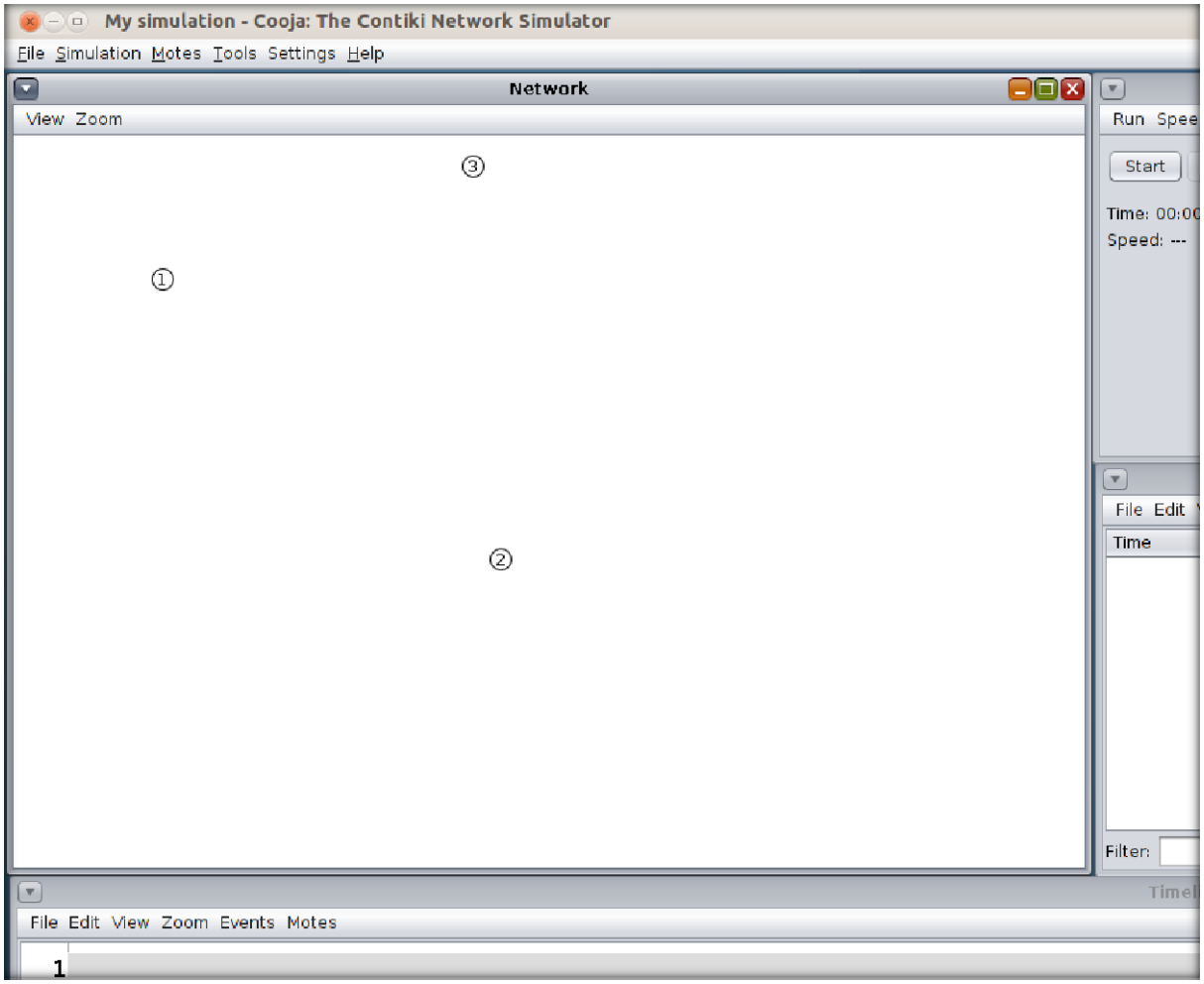
Choose hello-world.c and click “open”.



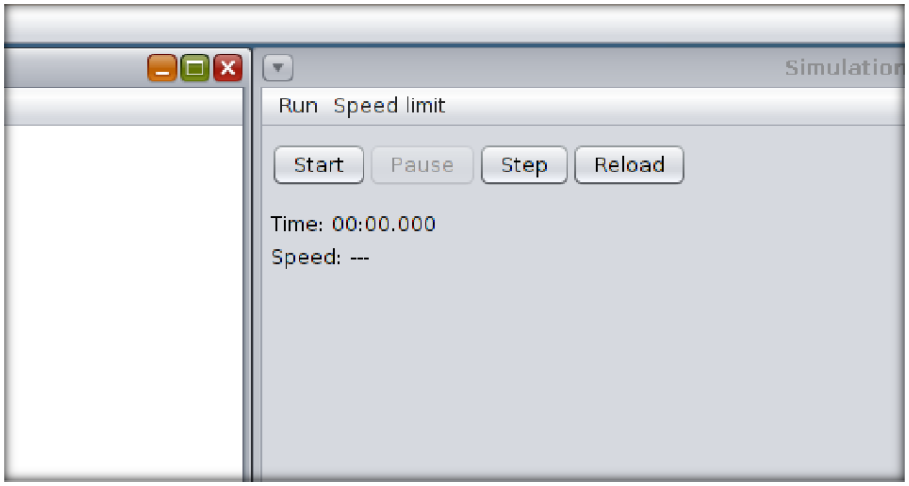
It will compile the program and show compilation output.
Click on “create”



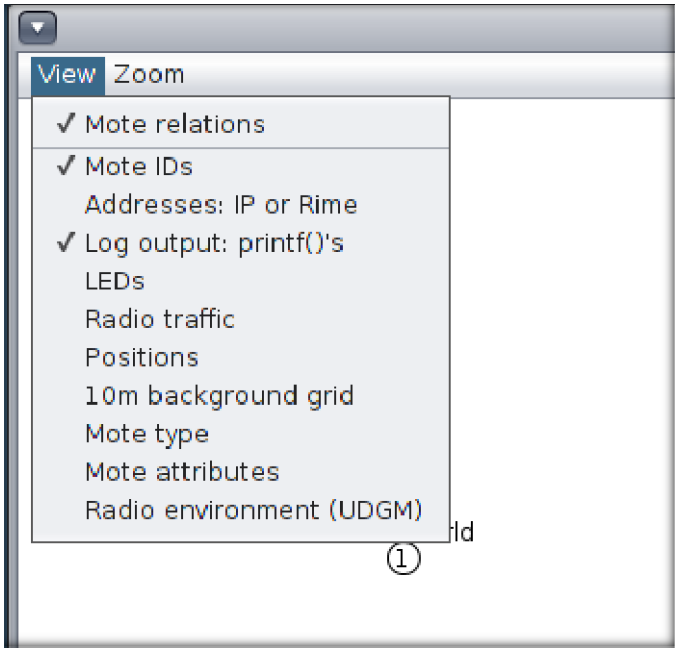
Enter the number of new motes.



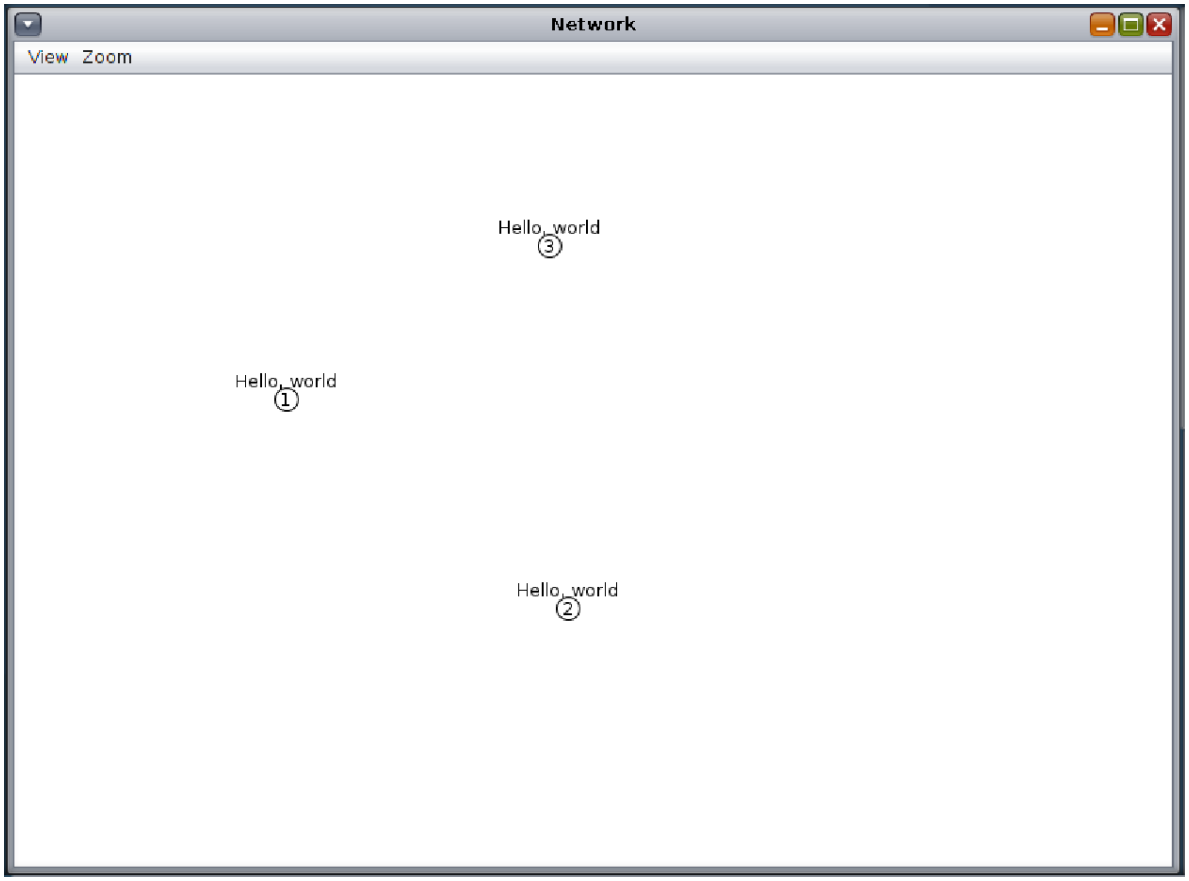
3 motes will be added at random positions now.
To run the simulation, click on start.



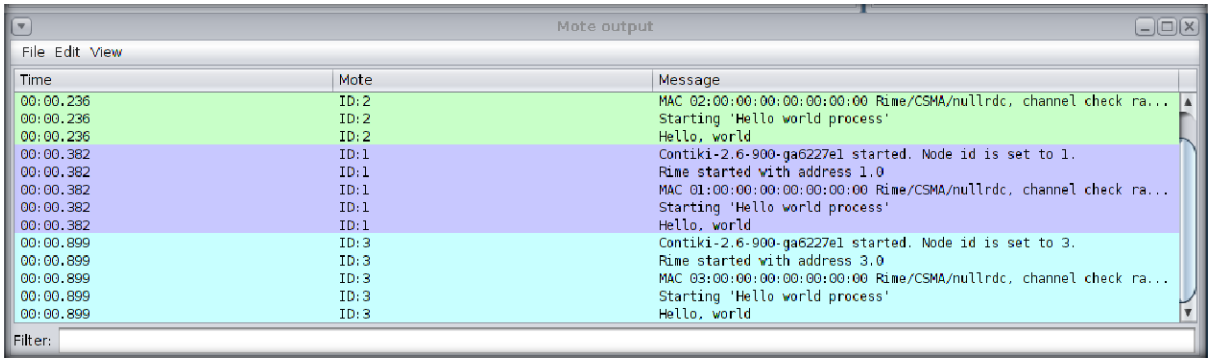
Go to view > Log output: printf()'s



Now output will be written beside all motes.



On the mote output, we will be able to see Hello world written there.



CONCLUSION:

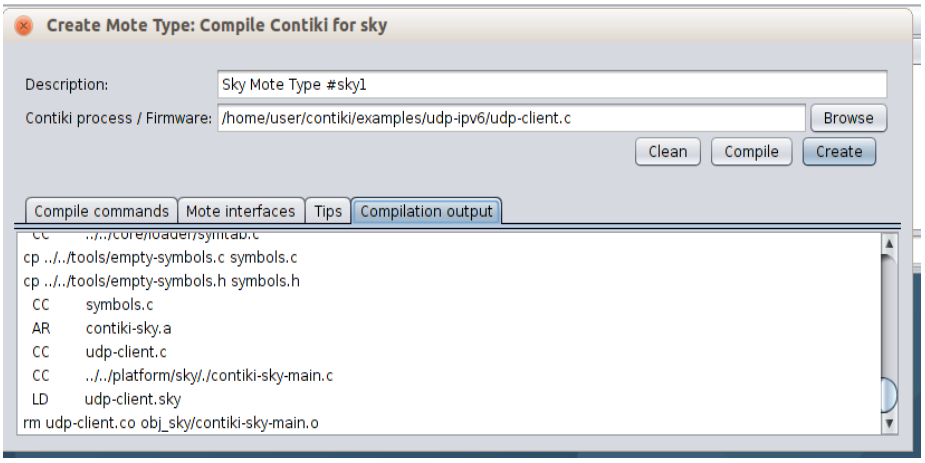
In this practical, we created our first simulation in cooja, We used the hello-world code and learned how to create new simulator and add motes to it and how to attach firmware to them.

PRACTICAL - 3

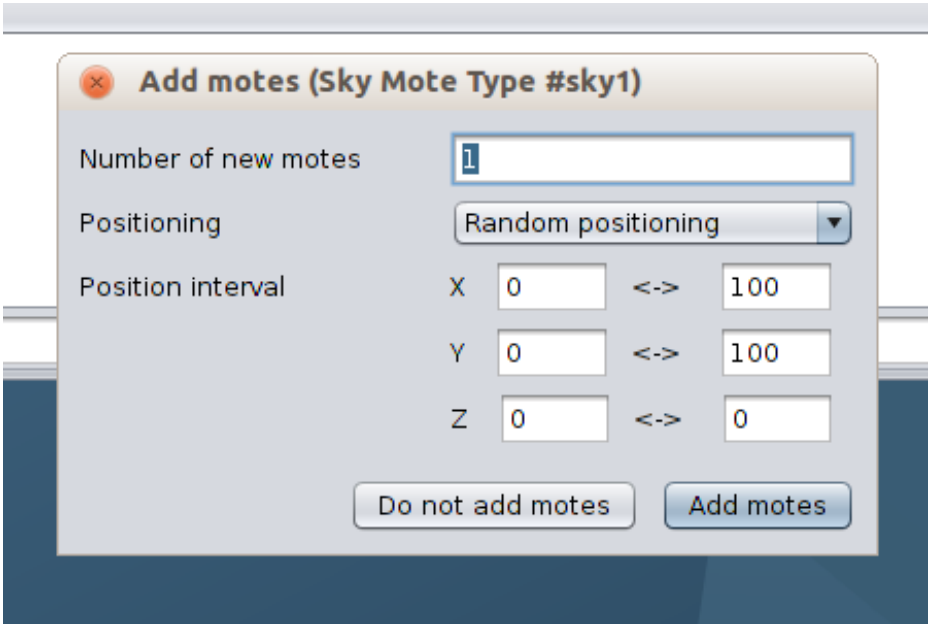
AIM: Create a scenario by adding some motes. Do the simulation for the same. Also observe the result for said scenario. When one mote sends the signal then led should turn green while one receive then it should show red color.

Implementation:

→Go to File menu, create New Simulation.
→Now, add skymote as shown below(First we create for client…)
→Go to **udp-client** folder as shown below. /examples/udp-ipv6/udp-client and selectfile **udp-client.c.**

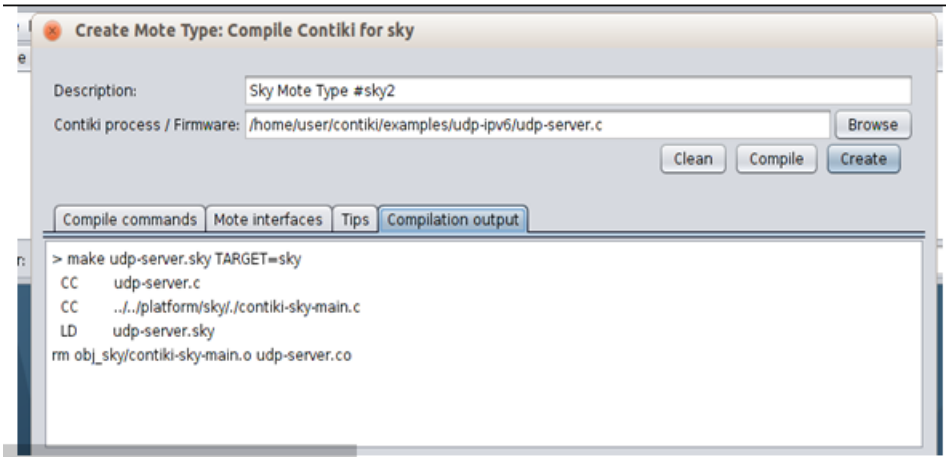


→Now, add number of motes…



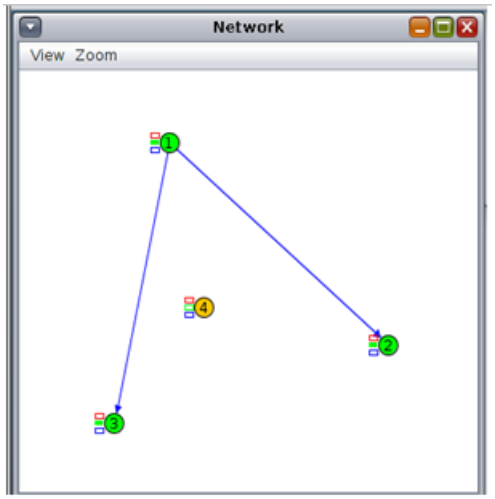
→Now, add another skymote as shown below(for server)

→Go to **udp-ipv6** folder as shown below. /examples/udp-ipv6 and select file **udp-server.c** as shown below…

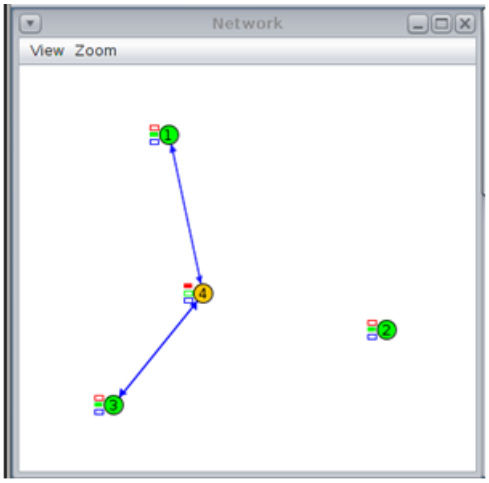


→Now comile and create that…

→In below figure you can see configured network between motes(Client and Server)…Client blinks green light in below figure which means they connected…



→In below figure, server mote is blinking with red light…which means conectionbetwewn all four motes was successfully established…



Conclusion:

→From this practical, I have learned about connection between motes.

PRACTICAL - 4

AIM: Simulate BGP and RPL protocol in Cooja.

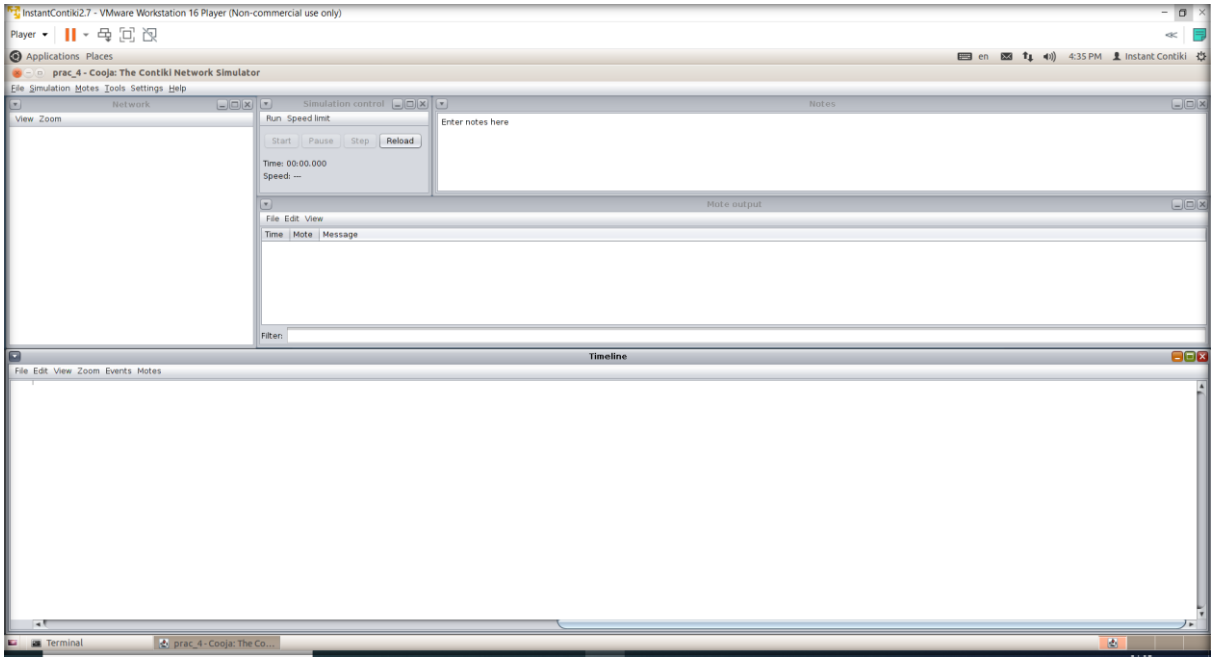
THEORY:

BGP:
Border Gateway Protocol (BGP) is a standardized exterior gateway protocol designed to exchange routing and reachability information between autonomous systems (AS) on the Internet. The protocol is often classified as a path vector protocol but is sometimes also classed as a distance-vector routing protocol."
In plain English, BGP (a.k.a. Border Gateway Protocol) is the routing method that enables the Internet to function. Without it, we wouldn't be able to do a Google search or send an email.
Each BGP speaker, which is called a "peer", exchanges routing information with its neighboring peers in the form of network prefix announcements. This way, an AS doesn't need to be connected to another AS to know its network prefix.
The BGP decision-making mechanism analyzes all the data and sets one of its peers as the next stop, to forward packets for a certain destination.
Each peer manages a table with all the routes it knows for each network and propagates that information to its neighboring autonomous systems.
In this way, BGP allows an AS to collect all the routing information from its neighboring autonomous systems and "advertise" that information further. Each peer transfers the information internally inside its own autonomous system.
Just like in real life, usually more than one route exists to reach a given destination. BGP is responsible for determining the most suitable route according to the information collected and an organization's routing policy, which is based on cost, reliability, speed, etc.

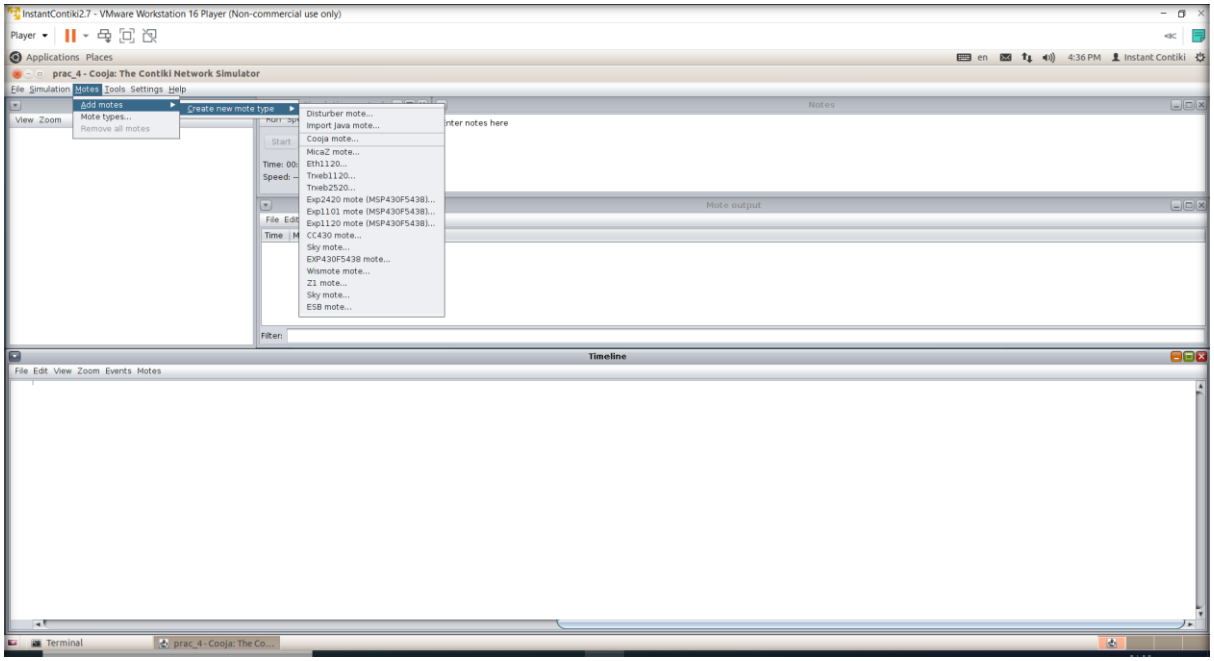
RPL:
RPL (Routing Protocol for Low-Power and Lossy Networks) is a routing protocol for wireless networks with low power consumption and generally susceptible to packet loss. It is a proactive protocol based on distance vectors and operates on IEEE 802.15.4, optimized for multi-hop and many-to-one communication, but also supports one-to-one messages.

This protocol is specified in RFC 6550 with special applications in RFCs 5867, 5826, 5673 and 5548. RPL can support a wide variety of link layers, including those with limitations, with potential losses or that are used in devices with limited resources. This protocol can quickly create network routes, share routing knowledge and adapt the topology in an efficient way.
The implementation of the RPL protocol occurs in wireless sensors and networks, the most used operating system for its implementation is Contiki which is a small open source operating system developed for use in a number of small systems ranging from 8-bit computers to integrated systems on microcontrollers, including sensor network nodes.

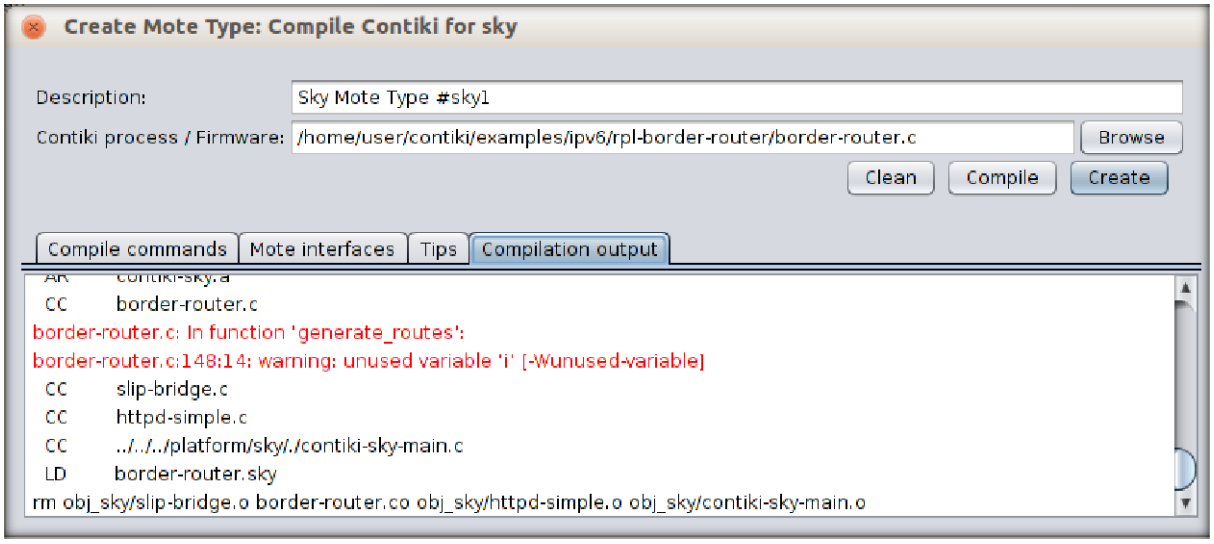
PRACTICAL:
For this task, we will add a sky mote as router and other motes to receive signals and we will observe the radio traffic between them.
First of all, we will open Cooja simulator and create new simulator.



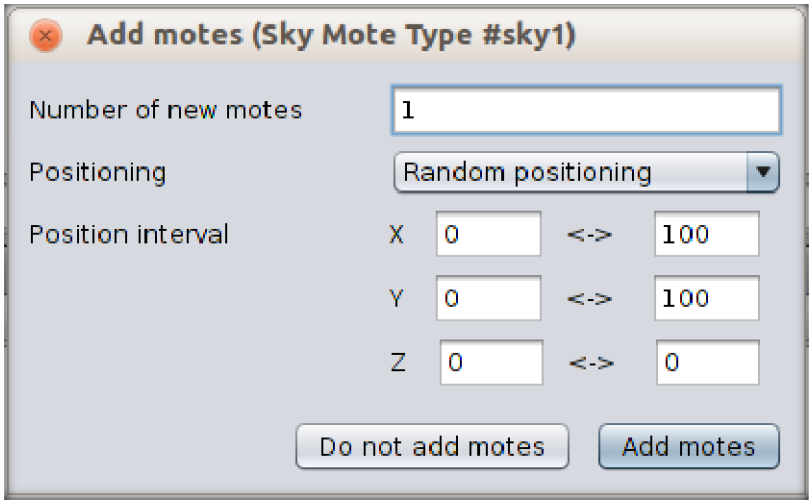
Then, we will first add the router mote, so we will go to Motes > Create new mote type > Sky mote.



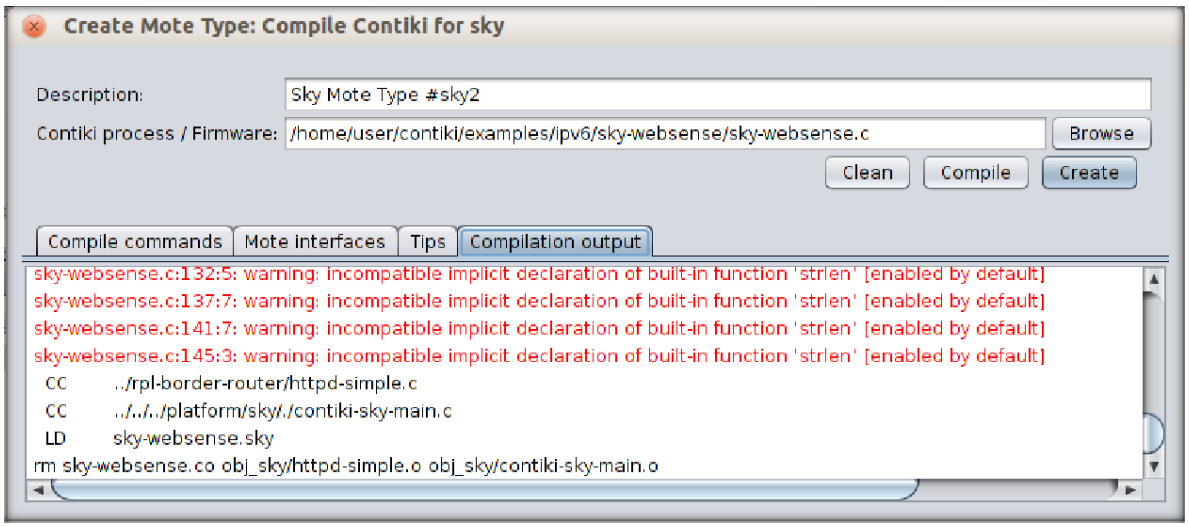
We will browse through example folder to find ipv6 > rpl-borer-router > border-router.c.
We will compile it and press create.



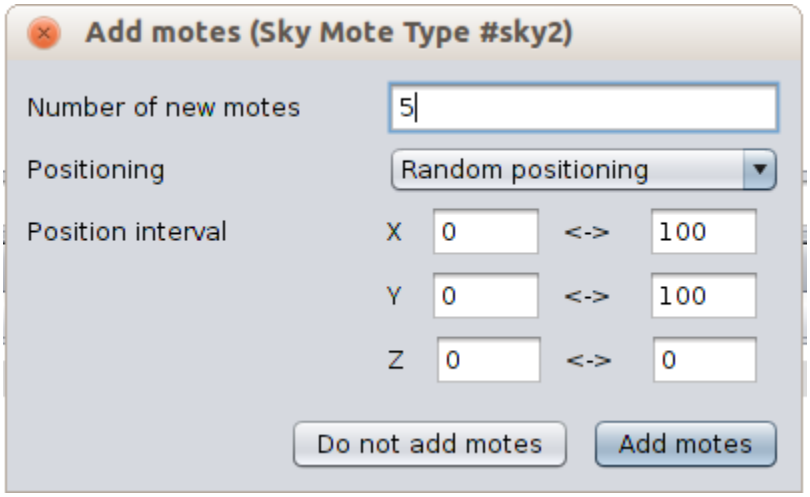
We will need only 1 router so number of new motes will be 1 only.



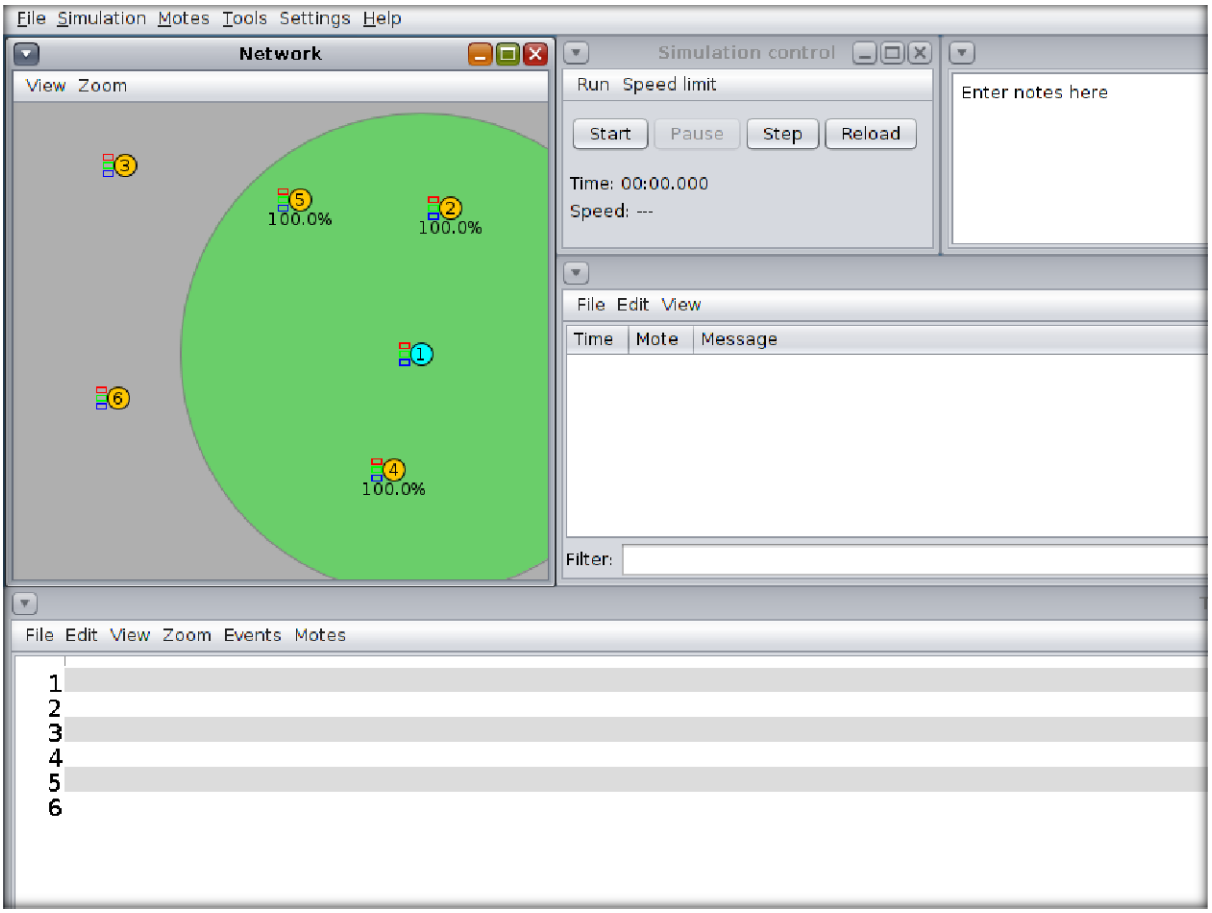
To create receiving motes, we will follow the same process to add sky motes.
This time, we will browse to ipv6 > sky-websense > sky-websense.c.



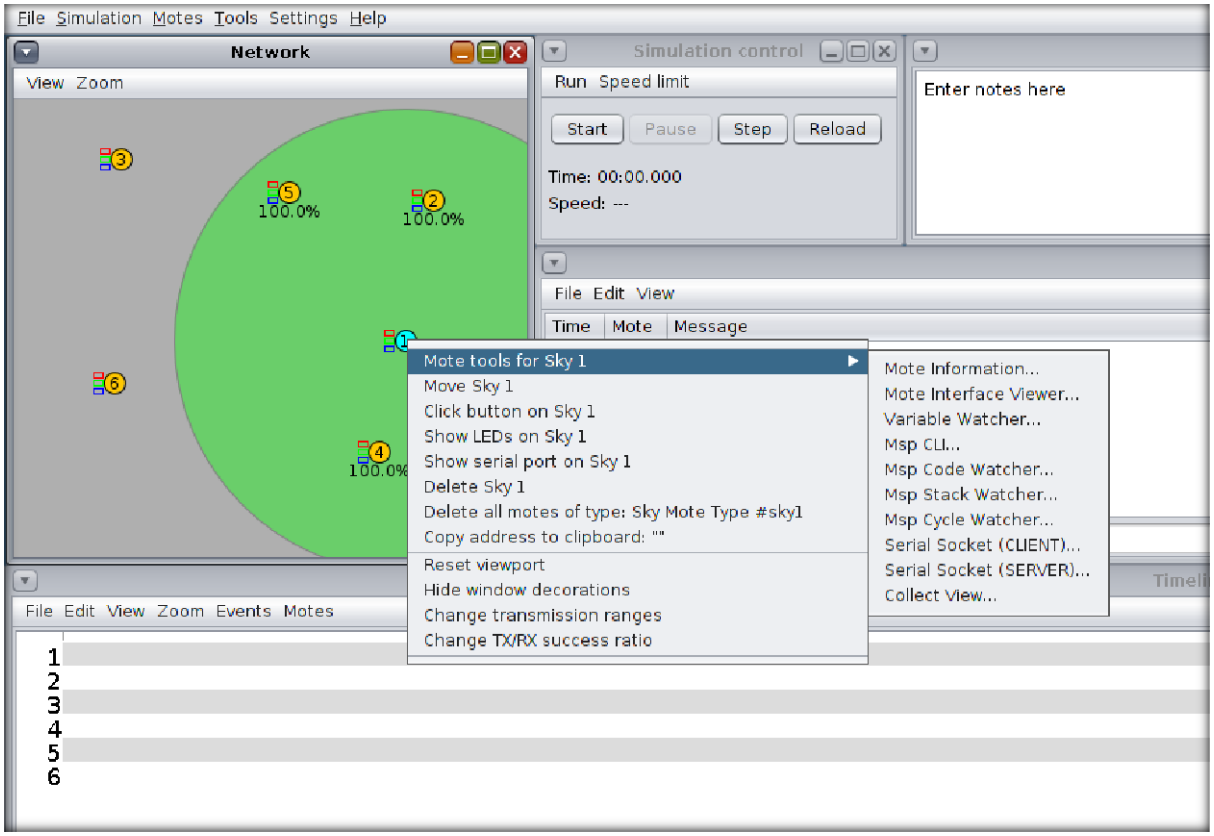
We will need more receivers for better visualization so we will add 5 motes.



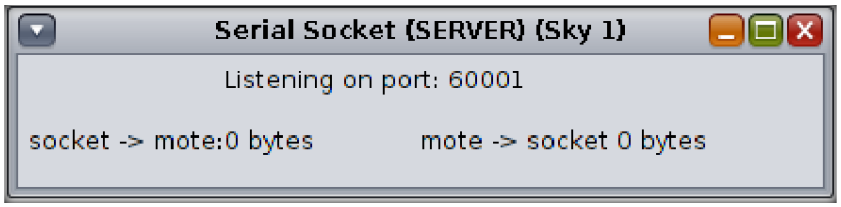
When we will add motes, we will be able to see a screen like this.
By clicking on the mote, you will be able to see its range.
We can change some view options as our convenience.



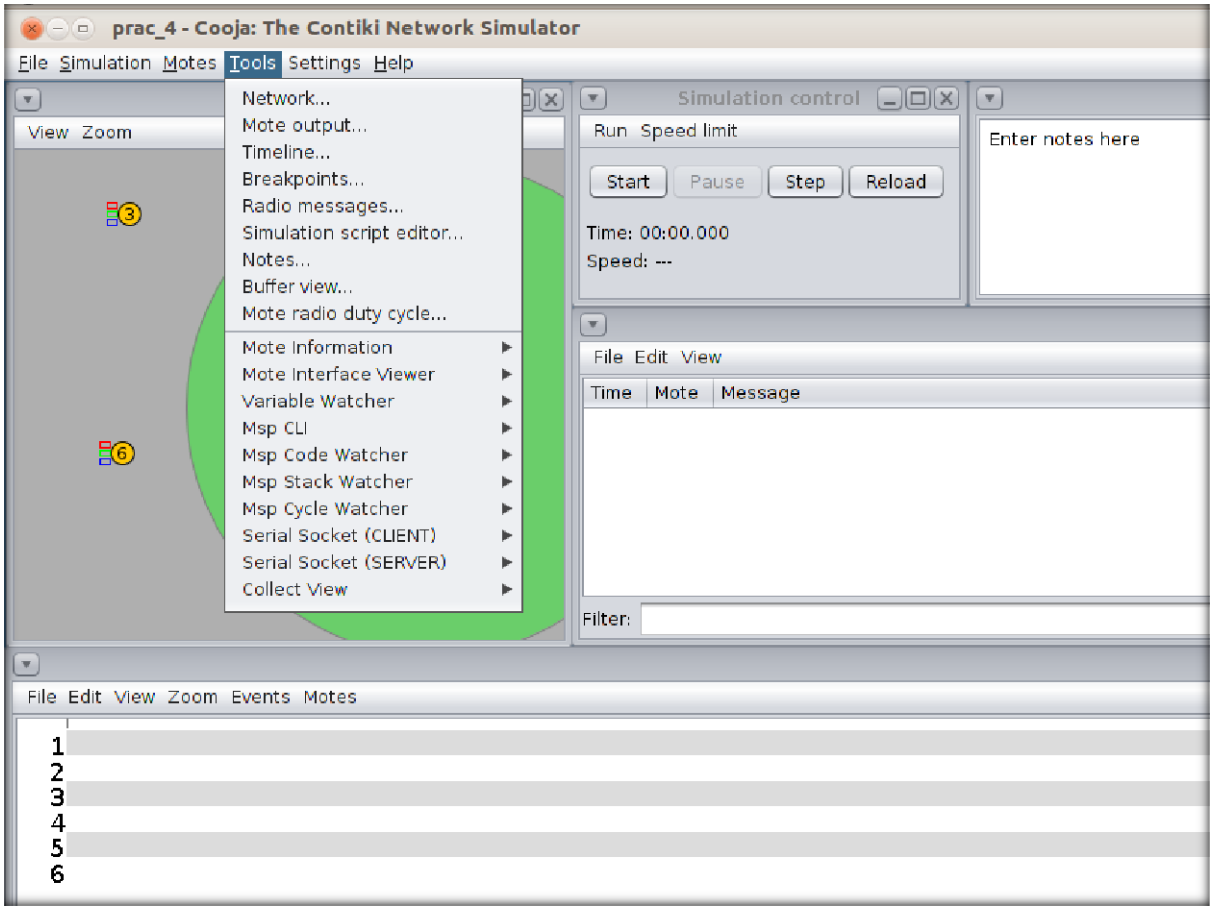
To see the traffic, we will right click on server and go to Mote tools for Sky 1 > Serial Socket (SERVER)



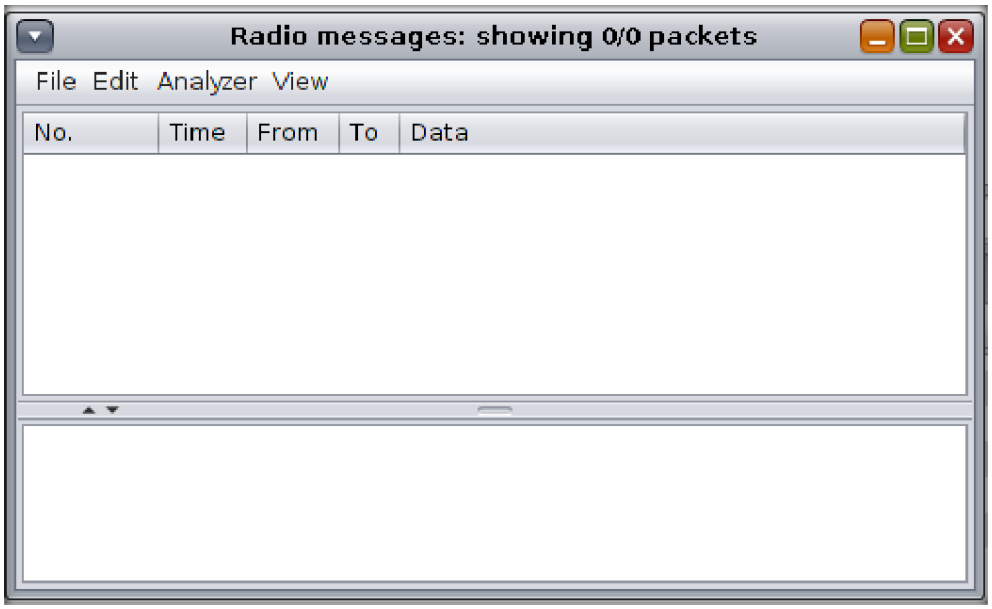
We will be able to see a new window like below.



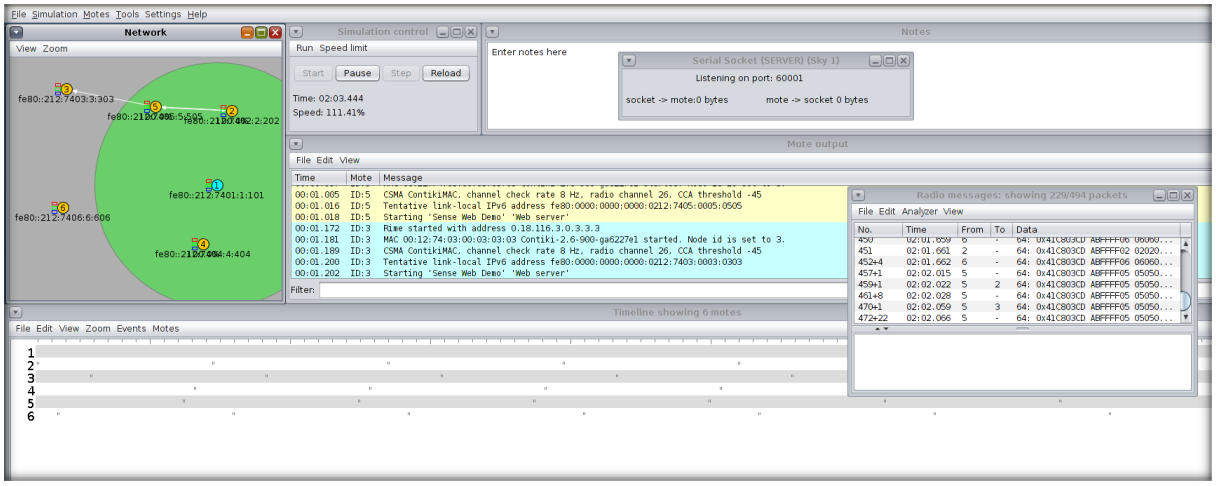
To get more information about traffic, we can go to tools > Radio messages.



We will be able to see another window like below.



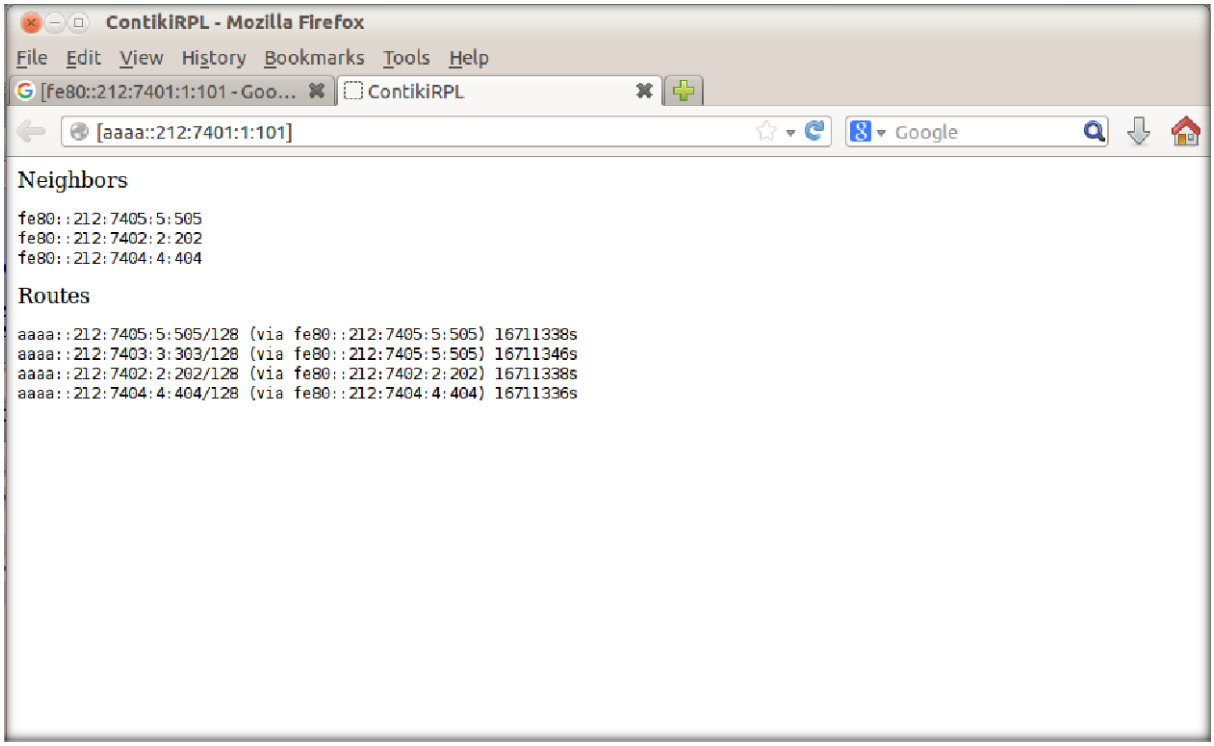
Then, we will finally start the simulator to see the radio traffic.



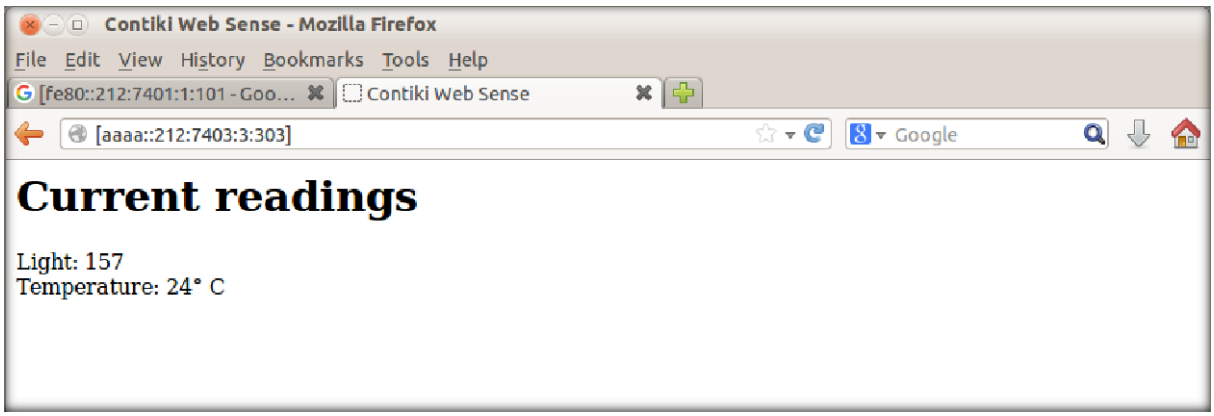
Now, to visualize the content of simulator in browser, we will run the following command into different terminal.
Commands:
cd Contiki/examples/ipv6/rpl-border-router/
Make connect-router-cooja

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

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

*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::212:7401:1:101
 fe80::212:7401:1:101
```

Now, if we write ip address of router in browser like firefox, we will be able to see its neighbors information.



And if we search for ip address of any receiver mote, we will be able to get its sensed data like light and temperature.



CONCLUSION:

In this practical, we learned about BGP and RPL, we implemented those concepts in cooja by sky motes and observed the radio traffic between motes.

PRACTICAL - 5

AIM: Simulate client server architecture using UDP on contiki-os.

THEORY:

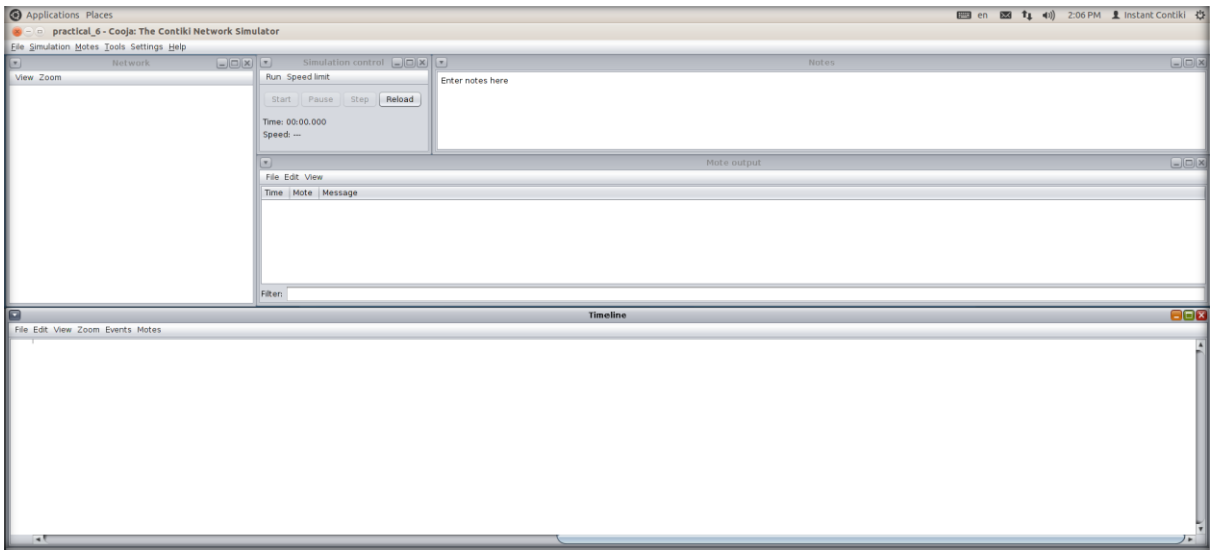
BGP:
In computer networking, the User Datagram Protocol (UDP) is one of the core members of the Internet protocol suite. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network. Prior communications are not required in order to set up communication channels or data paths.

UDP uses a simple connectionless communication model with a minimum of protocol mechanisms. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; there is no guarantee of delivery, ordering, or duplicate protection. If error-correction facilities are needed at the network interface level, an application may use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

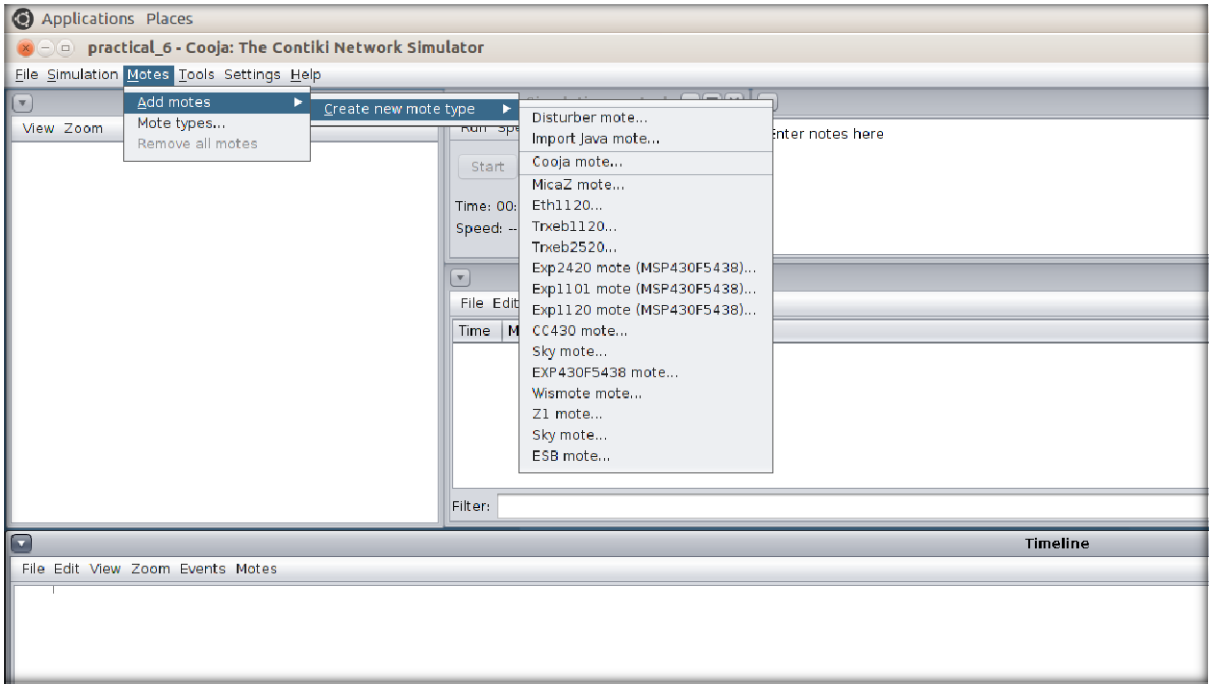
UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed due to retransmission, which may not be an option in a real-time system.
The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

PRACTICAL

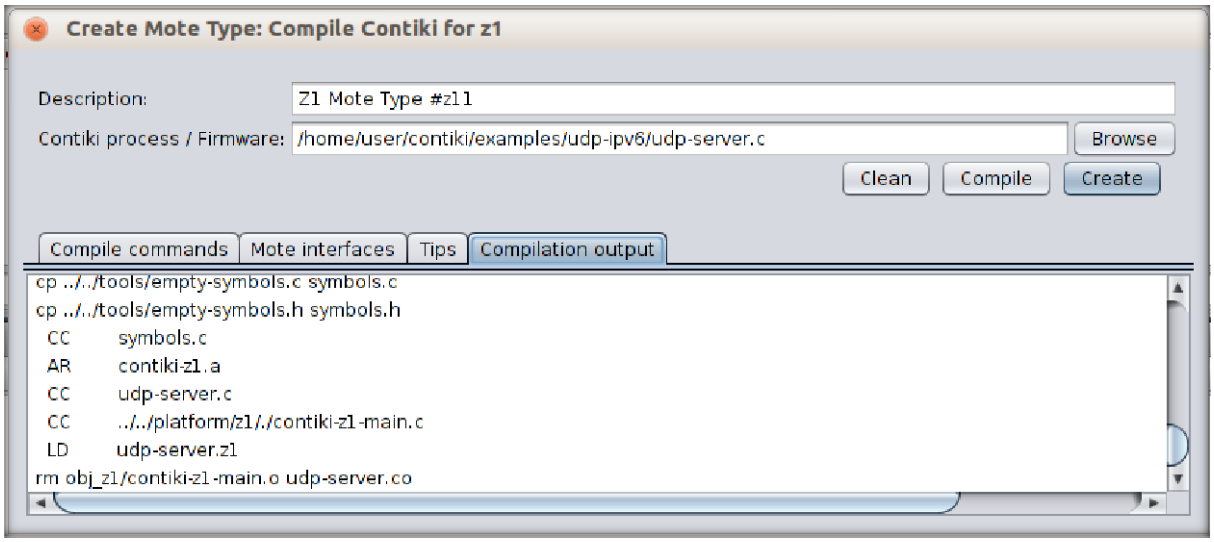
For this task, we will add a Z1 mote as router and other motes to receive signals and we will observe the UDP traffic between them.
First of all, we will open Cooja simulator and create new simulator.



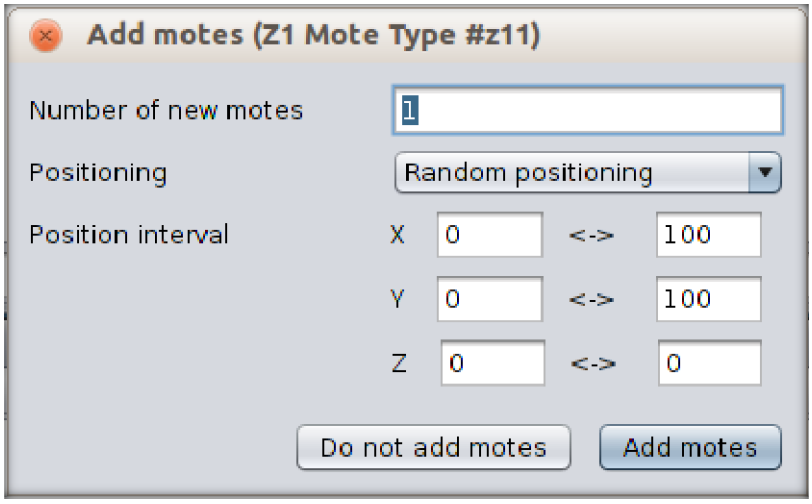
Then, we will first add the router mote, so we will go to Motes > Create new mote type > Z1 mote.



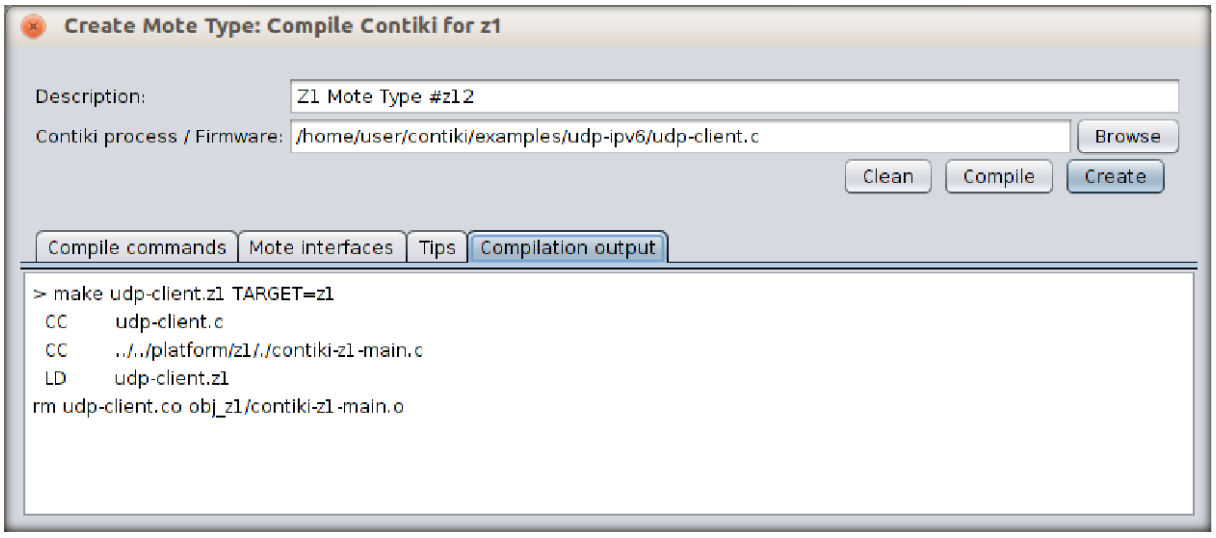
We will browse through example folder to find udp-ipv6 > udp-server.c.
We will compile it and press create.



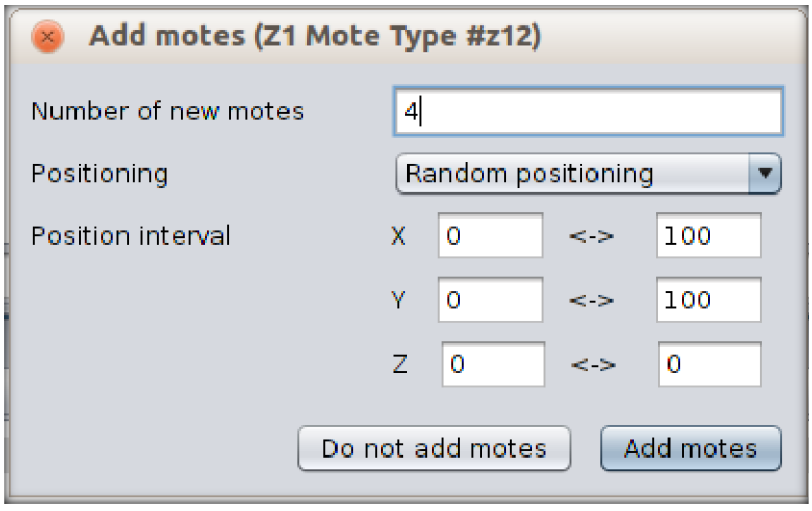
We will need only 1 router so number of new motes will be 1 only.



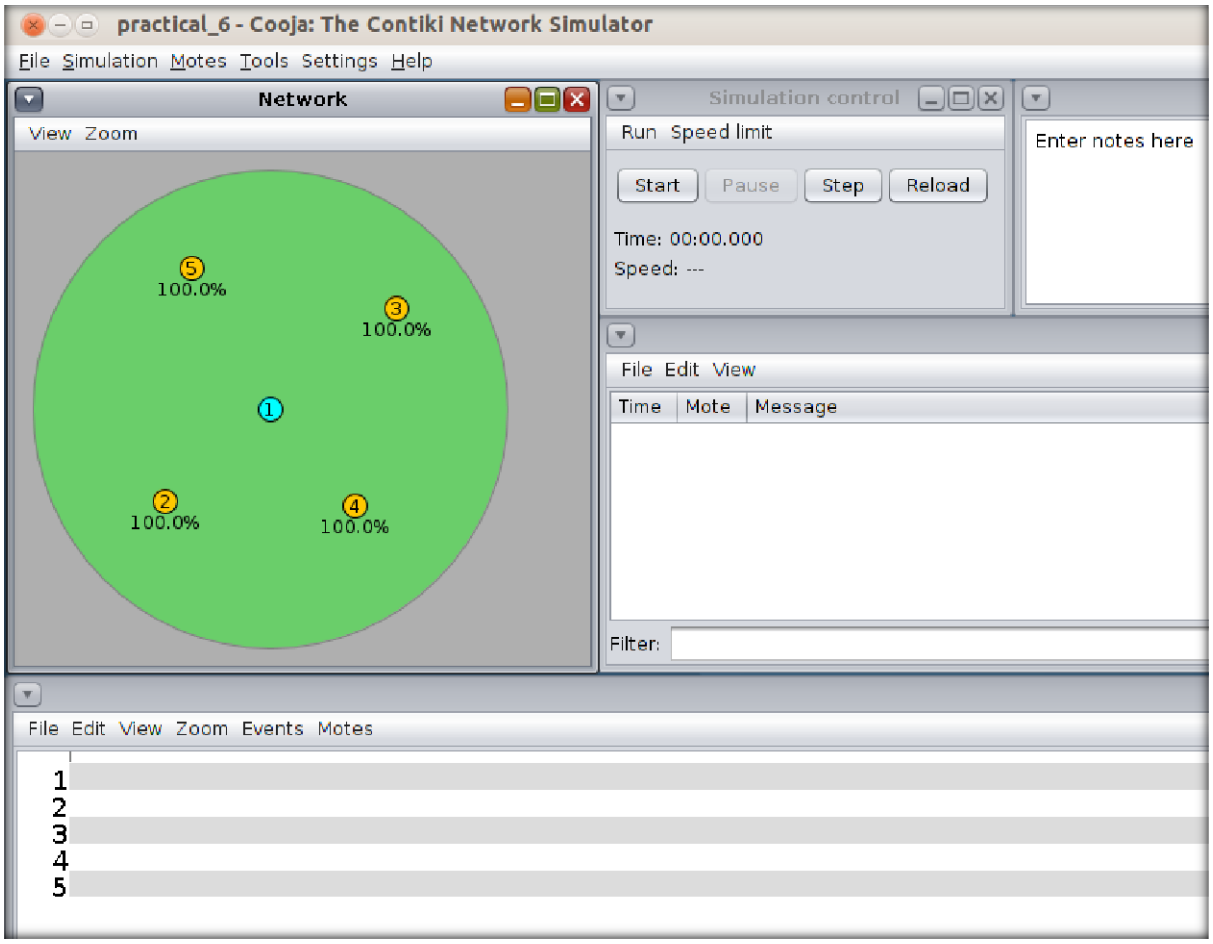
To create receiving motes, we will follow the same process to add Z1 motes.
This time, we will browse to udp-ipv6 > udp-client.c.



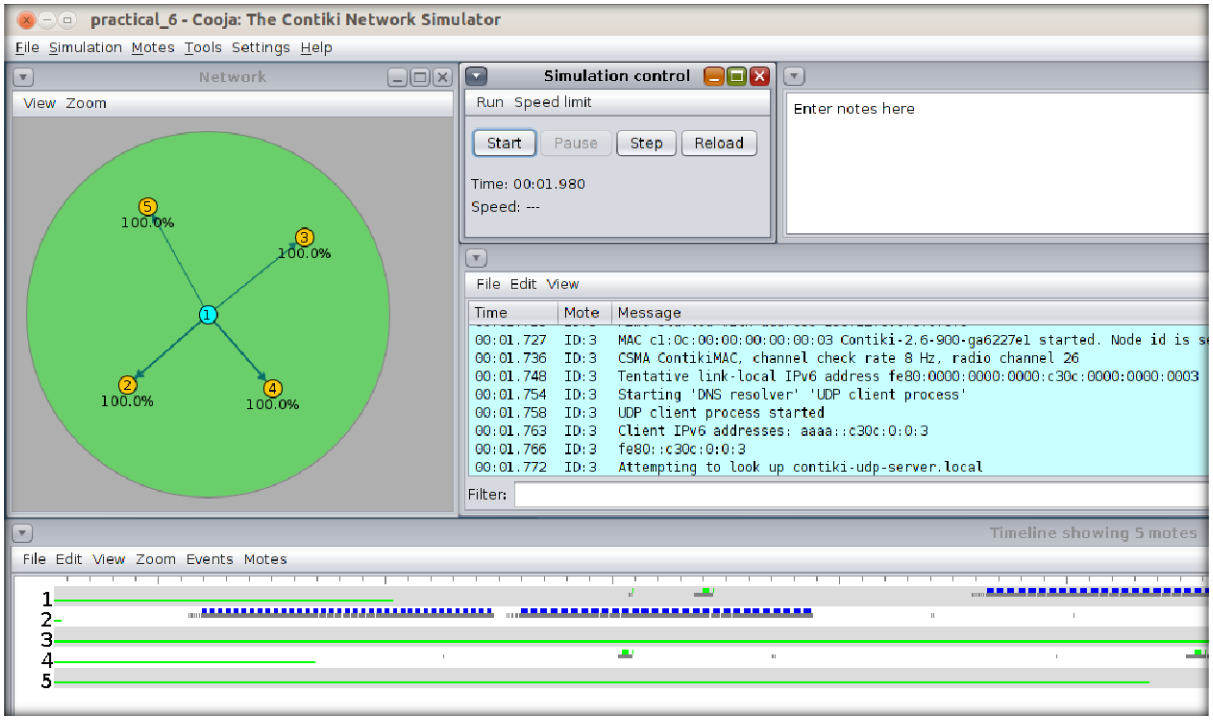
We will need more receivers for better visualization so we will add 4 motes.



When we will add motes, we will be able to see a screen like this.
By clicking on the mote, you will be able to see its range.
We can change some view options as our convenience.



Then, we will finally start the simulator to see the UDP traffic.



CONCLUSION

In this practical, we learned about UDP and implemented server client model in Cooja.

PRACTICAL - 6

AIM: Demonstrate message publish & subscribe mechanism of MQTT protocol using node red.

THEORY:

- **Node Red:**
  - Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.
  - It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime in a single-click.
  - Node-RED provides a browser-based flow editor that makes it easy to wire together flows using the wide range of nodes in the palette. Flows can be then deployed to the runtime in a single-click.
  - JavaScript functions can be created within the editor using a rich text editor.
  - A built-in library allows you to save useful functions, templates or flows for re-use.
  - The light-weight runtime is built on Node.js, taking full advantage of its event-driven, non-blocking model. This makes it ideal to run at the edge of the network on low-cost hardware such as the Raspberry Pi as well as in the cloud.
  - With over 225,000 modules in Node's package repository, it is easy to extend the range of palette nodes to add new capabilities.
  - The flows created in Node-RED are stored using JSON which can be easily imported and exported for sharing with others.
  - An online flow library allows you to share your best flows with the world.

- **MQTT**:
  - MQTT is an OASIS standard messaging protocol for the Internet of Things (IoT).
  - It is designed as an extremely lightweight publish/subscribe messaging transport that is ideal for connecting remote devices with a small code footprint and minimal network bandwidth.
  - MQTT today is used in a wide variety of industries, such as automotive, manufacturing, telecommunications, oil and gas, etc.
  - Features of MQTT:
    - Lightweight and Efficient
    - MQTT clients are very small, require minimal resources so can be used on small microcontrollers. MQTT message headers are small to optimize network bandwidth.
    - Bi-directional Communications
    - MQTT allows for messaging between device to cloud and cloud to device. This makes for easy broadcasting messages to groups of things.
    - Scale to Millions of Things
    - MQTT can scale to connect with millions of IoT devices.
    - Reliable Message Delivery
    - Reliability of message delivery is important for many IoT use cases. This is why MQTT has 3 defined quality of service levels: 0 - at most once, 1- at least once, 2 - exactly once
    - Support for Unreliable Networks
    - Many IoT devices connect over unreliable cellular networks. MQTT's support for persistent sessions reduces the time to reconnect the client with the broker.

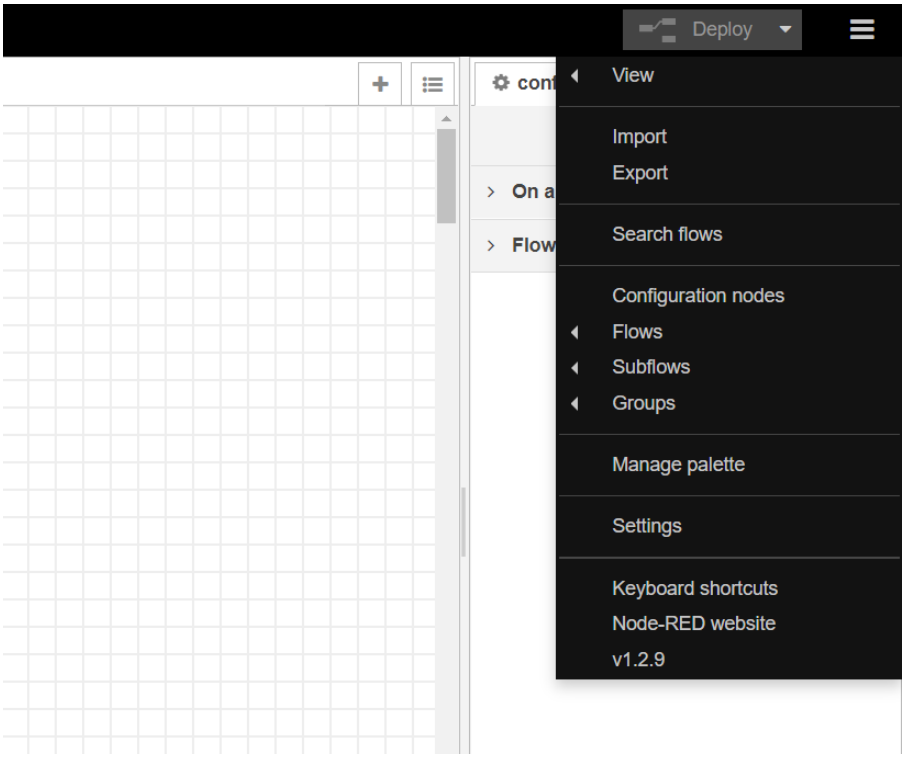
PRACTICAL

We can install node red on windows by following command.

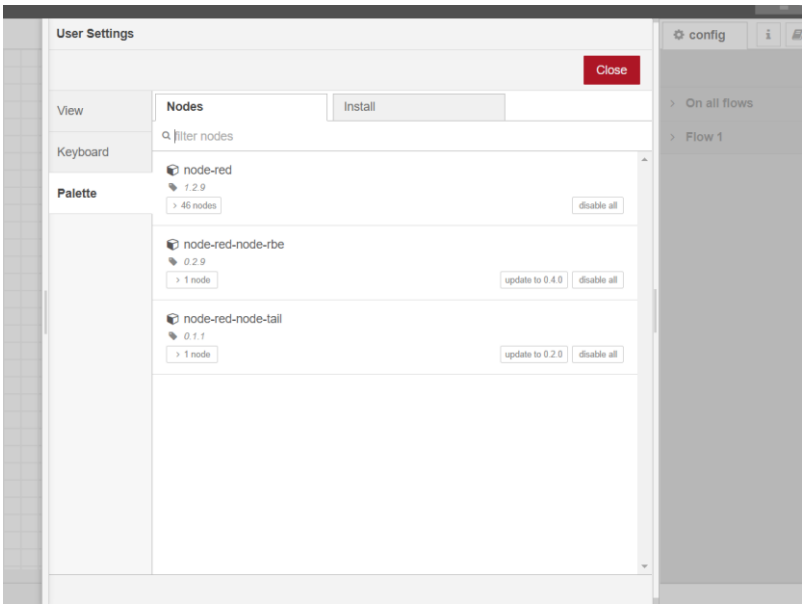
- npm install -g --unsafe-perm node-red

Then we can run “node-red” command in cmd to start the node-red.

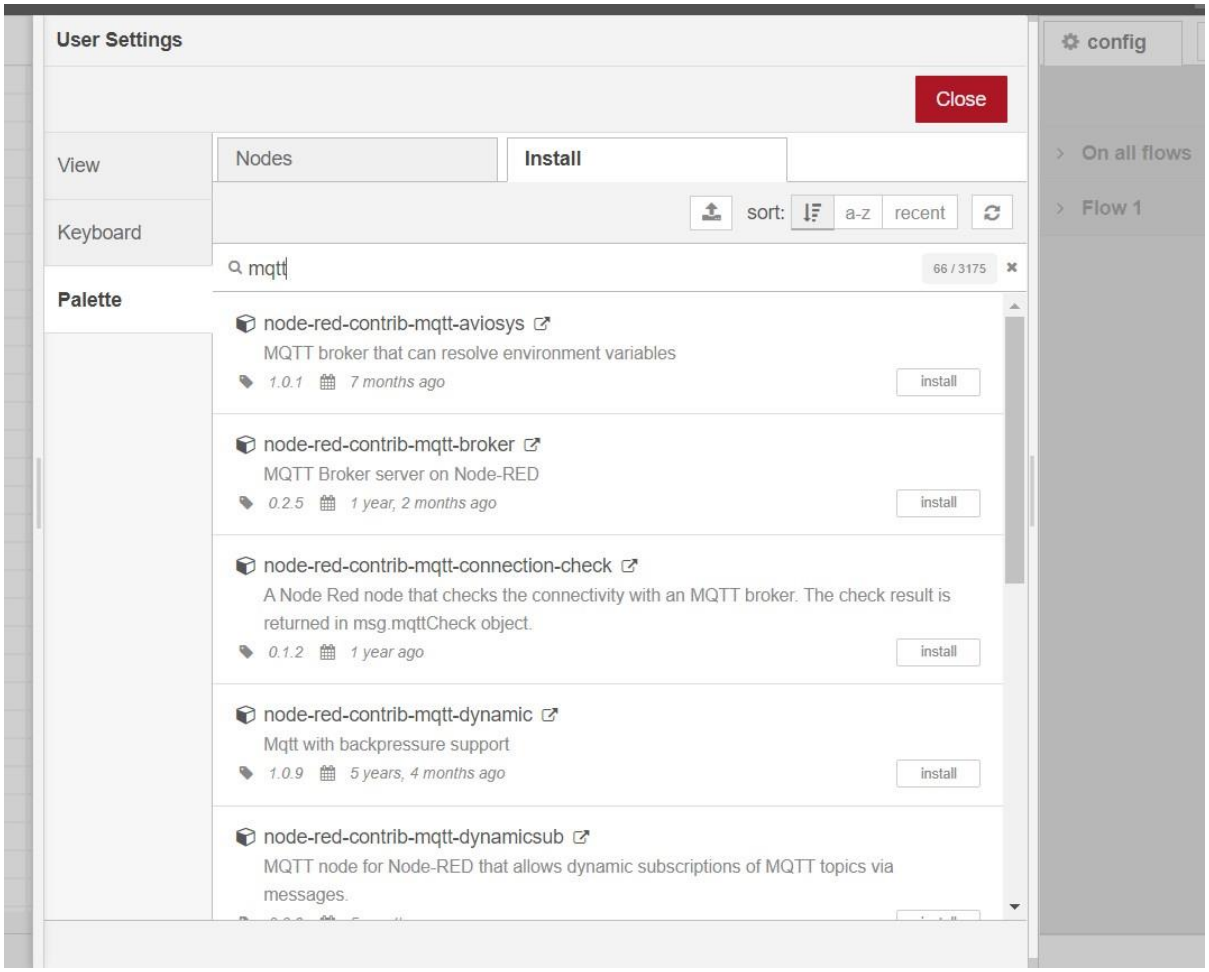
```
19 Mar 09:59:11 - [info]

Welcome to Node-RED
===================

19 Mar 09:59:11 - [info] Node-RED version: v1.2.9
19 Mar 09:59:11 - [info] Node.js  version: v14.15.4
19 Mar 09:59:11 - [info] Windows_NT 10.0.19042 x64 LE
19 Mar 09:59:12 - [info] Loading palette nodes
19 Mar 09:59:31 - [info] Settings file  : C:\Users\jilsa\.node-red\settings.js
19 Mar 09:59:31 - [info] Context store  : 'default' [module=memory]
19 Mar 09:59:31 - [info] User directory : C:\Users\jilsa\.node-red
19 Mar 09:59:31 - [warn] Projects disabled : editorTheme.projects.enabled=false
19 Mar 09:59:31 - [info] Flows file     : C:\Users\jilsa\.node-red\flows_DESKTOP-S5UT1SO.json
19 Mar 09:59:31 - [info] Creating new flow file
19 Mar 09:59:31 - [warn]

---------------------------------------------------------------------
Your flow credentials file is encrypted using a system-generated key.

If the system-generated key is lost for any reason, your credentials
file will not be recoverable, you will have to delete it and re-enter
your credentials.

You should set your own key using the 'credentialSecret' option in
your settings file. Node-RED will then re-encrypt your credentials
file using your chosen key the next time you deploy a change.
---------------------------------------------------------------------
19 Mar 09:59:31 - [info] Server now running at http://127.0.0.1:1880/
19 Mar 09:59:31 - [info] Starting flows
19 Mar 09:59:31 - [info] Started flows
```



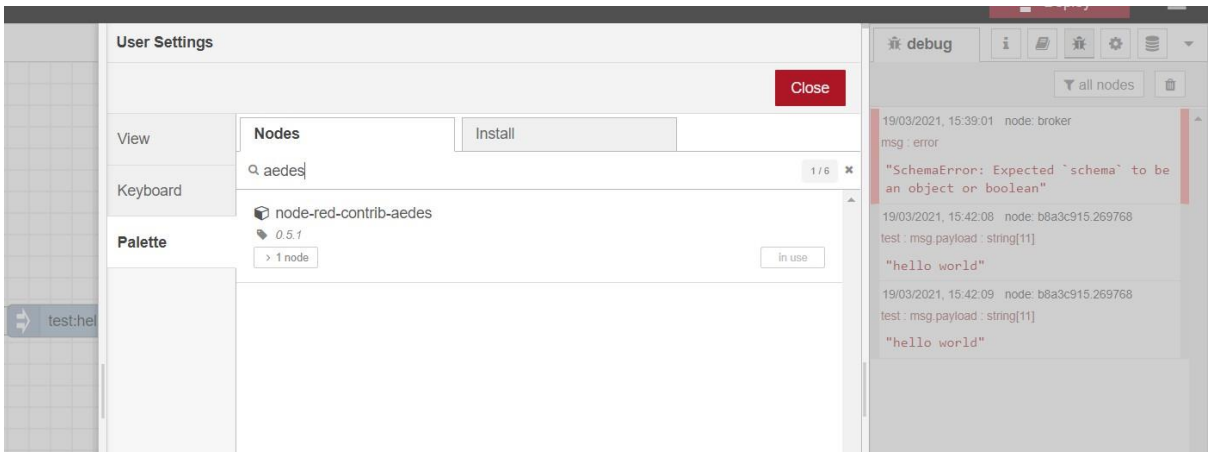
By clicking on it, User settings will be opened.

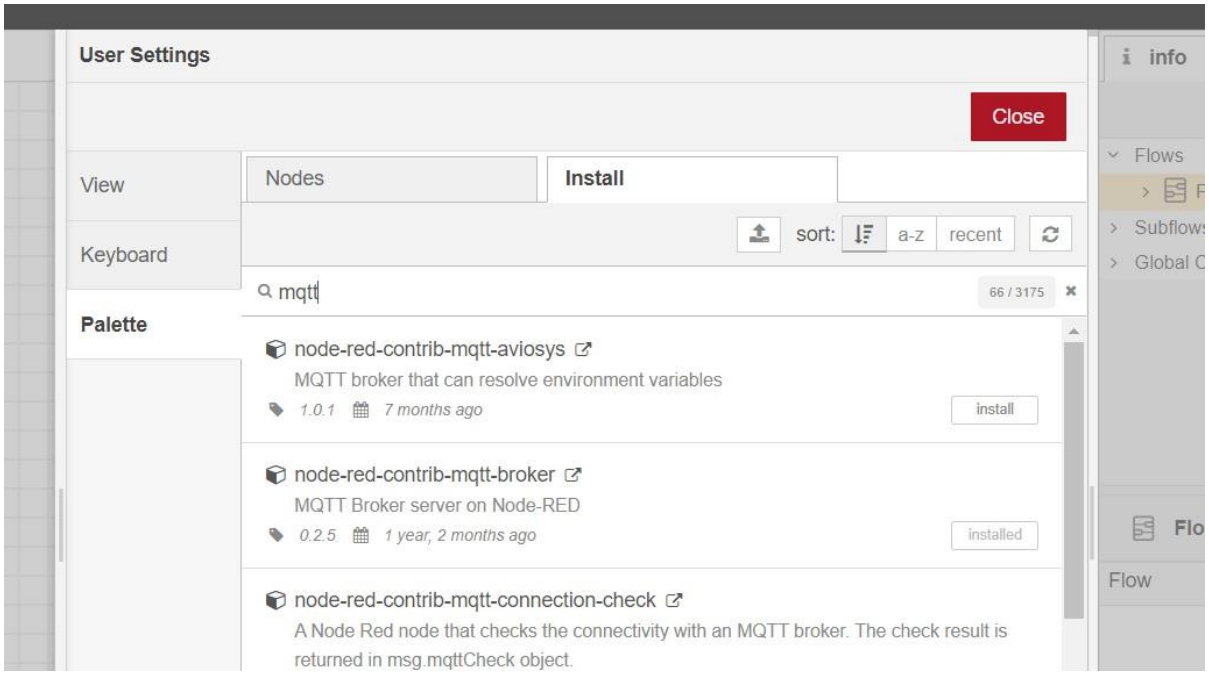


We go to Install tab and search MQTT.



We will install “node-red-contrib-mqtt-broker” or “node-red-contrib-aedes”

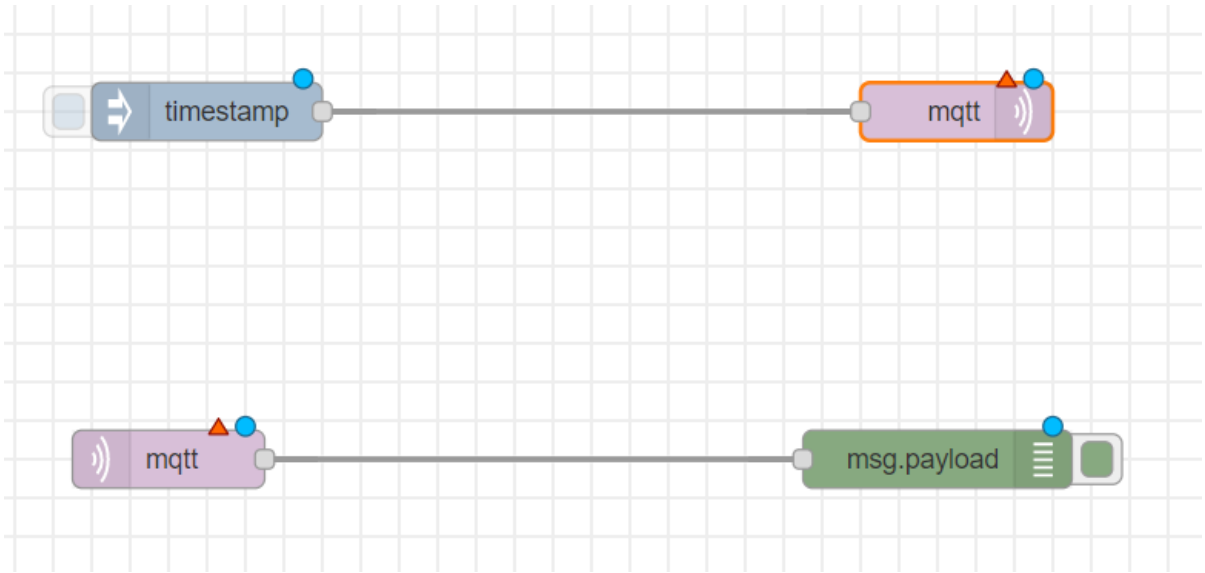




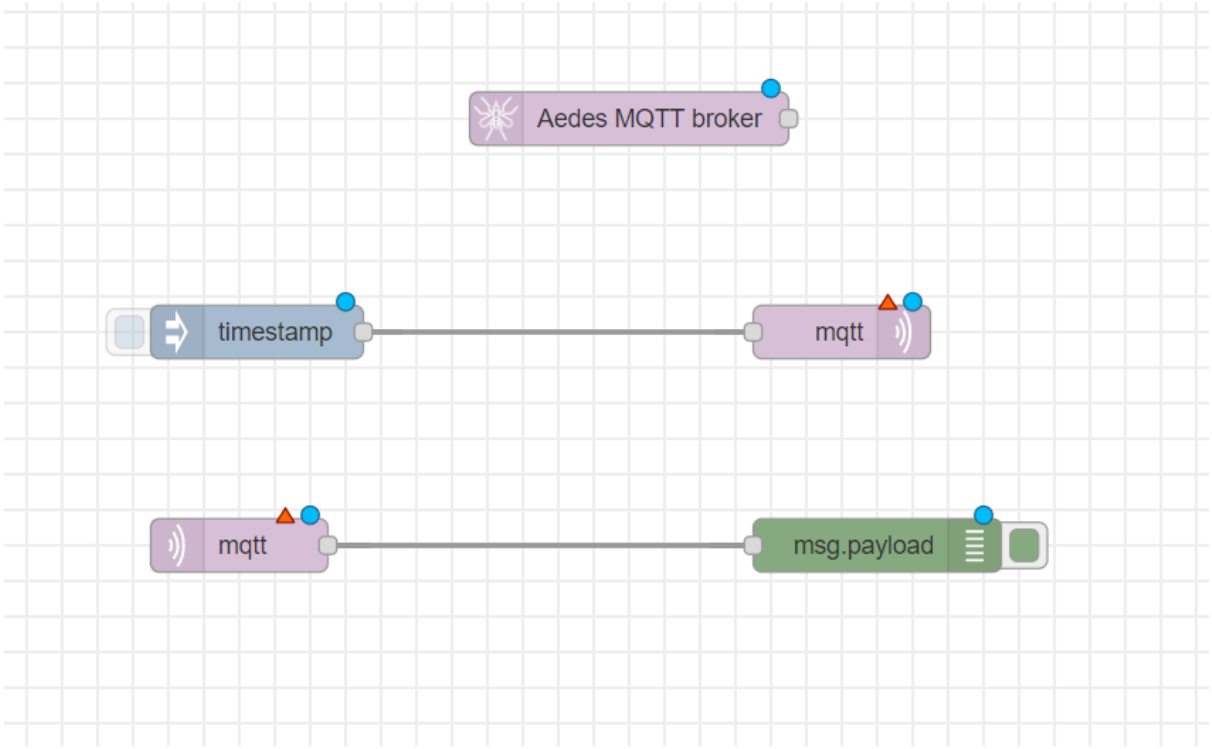
Then we will add 4 nodes.

1. Inject node, which will be renamed as Timestamp.
2. Debug node, which will be renamed as msg.payload.
3. MQTT in
4. MQTT out

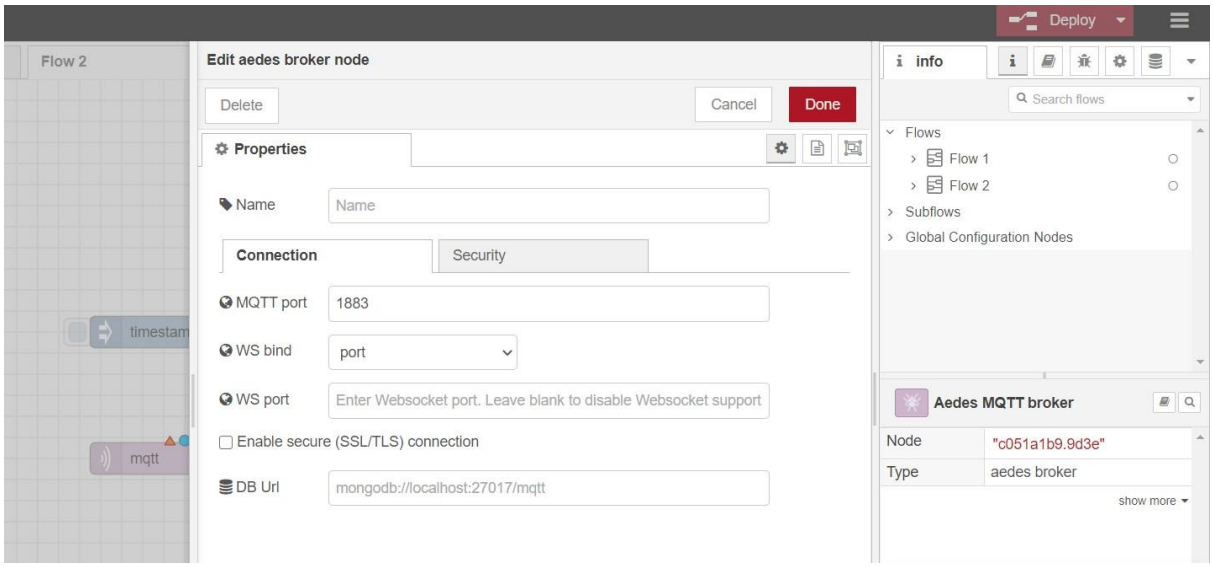
We will connect timestamp node to MQTT out and debug node to MQTT in.



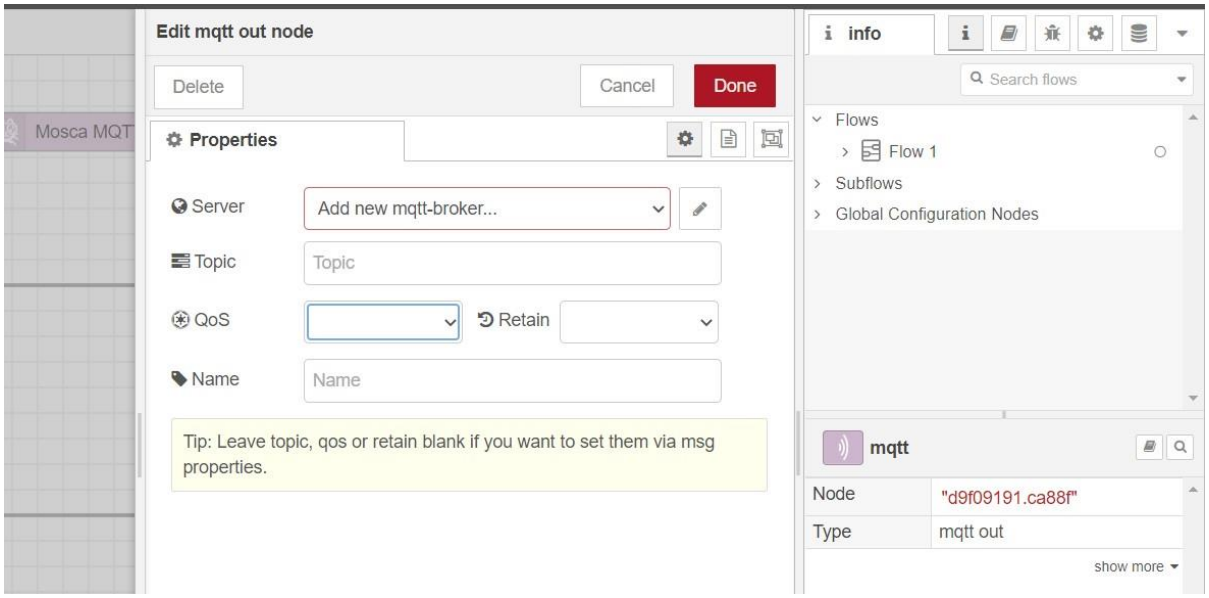
Then we will add MOSCA MQTT broker or AEDES MQTT broker.



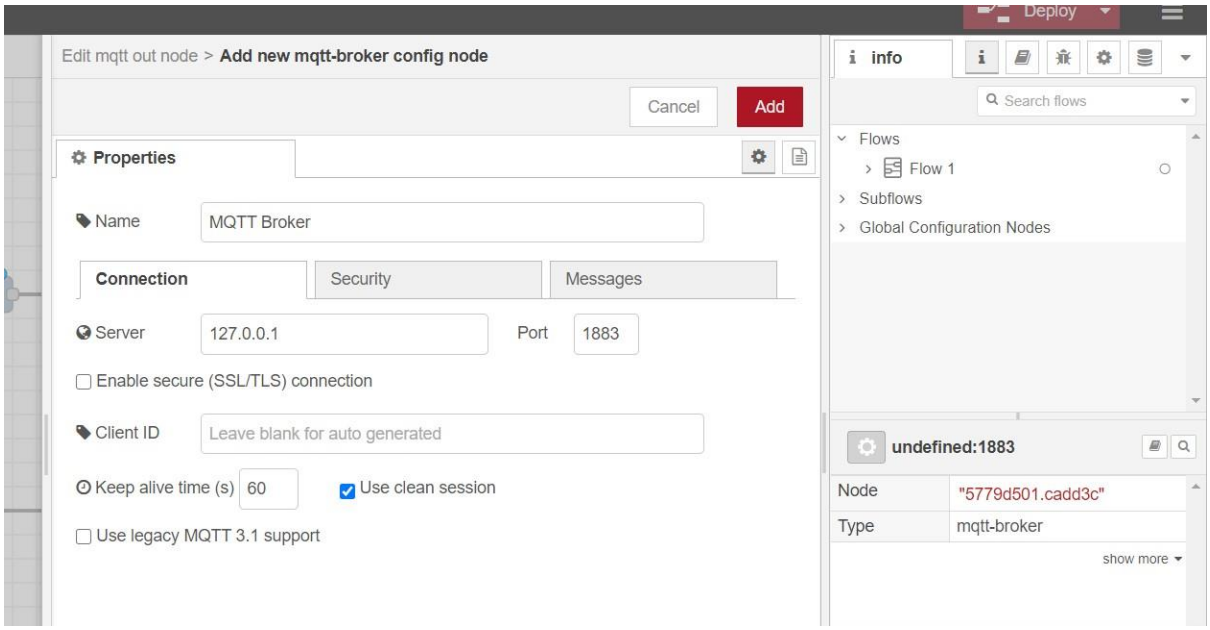
We can double click on node to see and change their propertied.



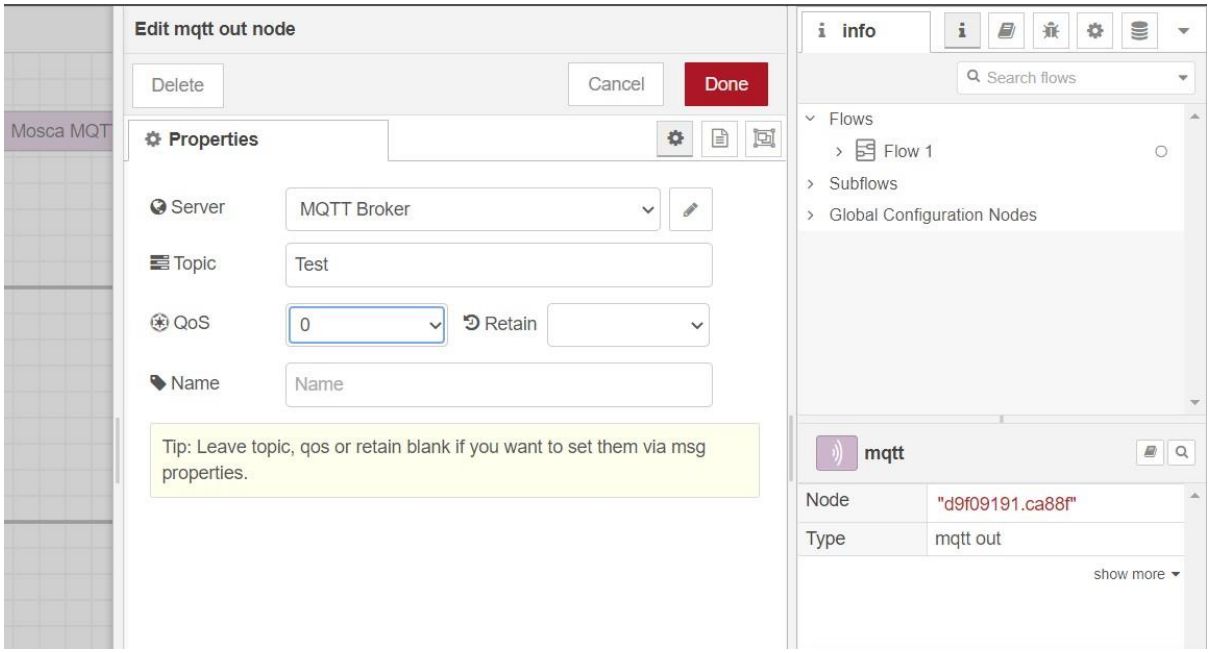
We will first configure MQTT out node.



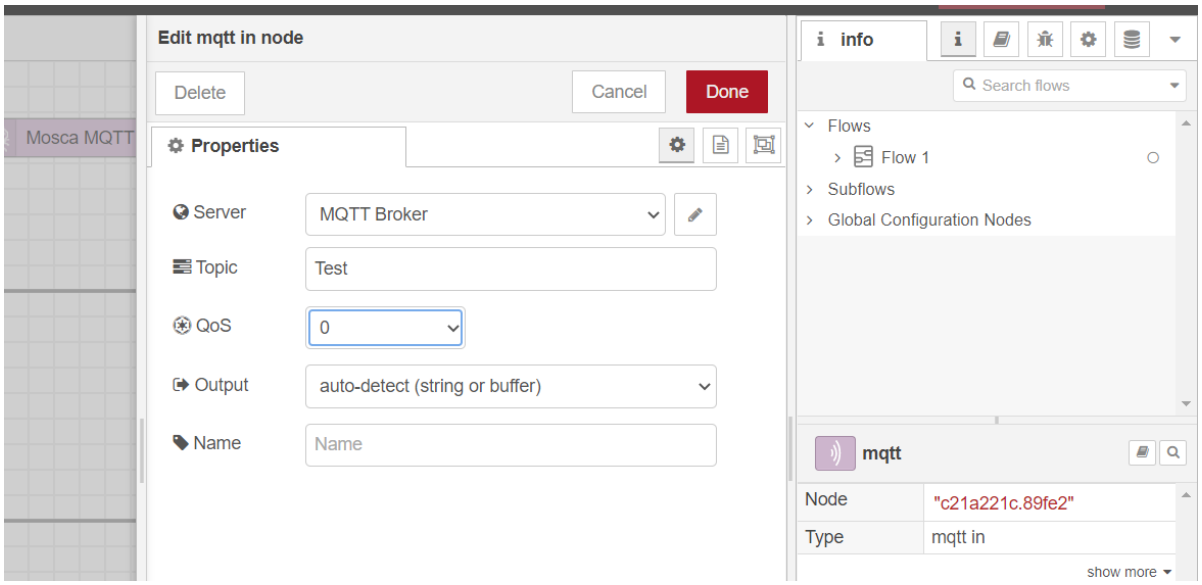
We will add new mttt-broker. So we click on the button beside it.
We will add name and server IP address.
We can add additional settings like security too if we want.



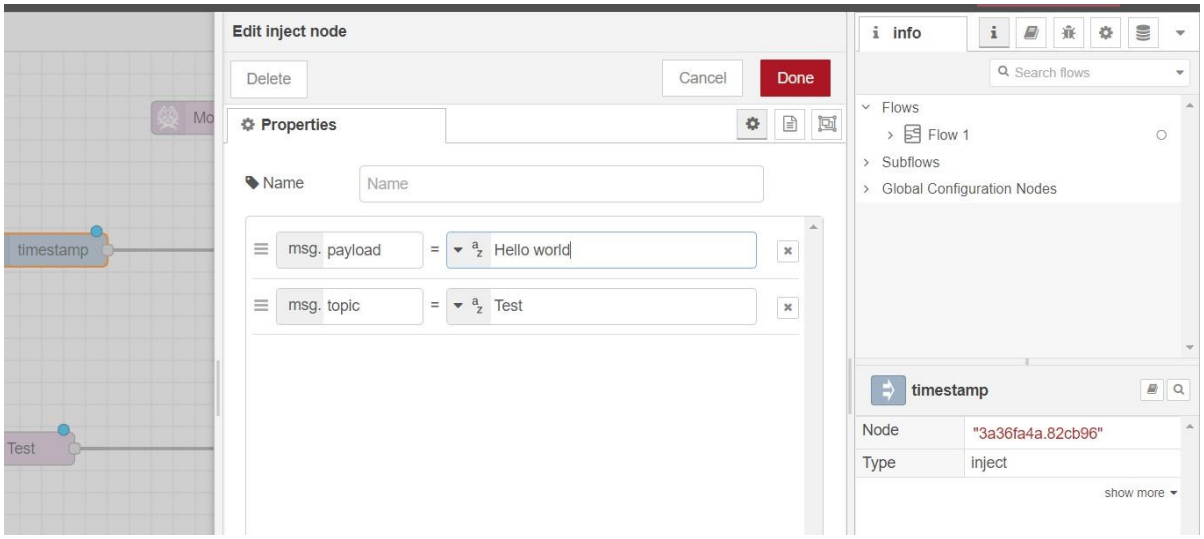
When we click add, the server will be created and we will fill a couple of fields there like topic and QoS.



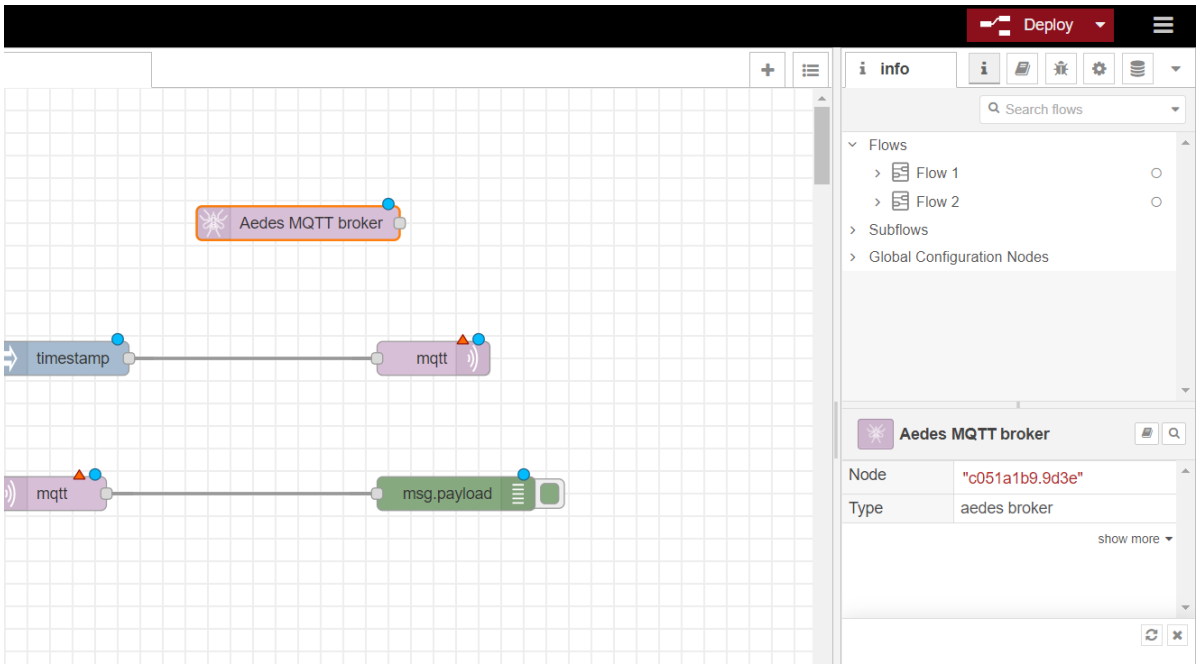
We will provide same configuration for MQTT in node but we don't need to create MQTT Broker again.



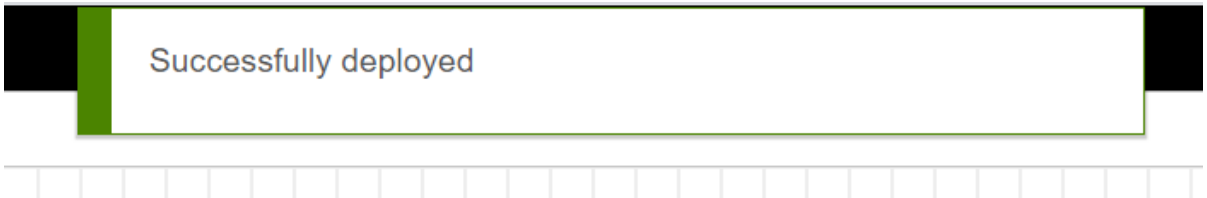
We will give input string in timestamp node. We change the timestamp to string of "Hello world" and give it the topic name same as MQTT nodes.



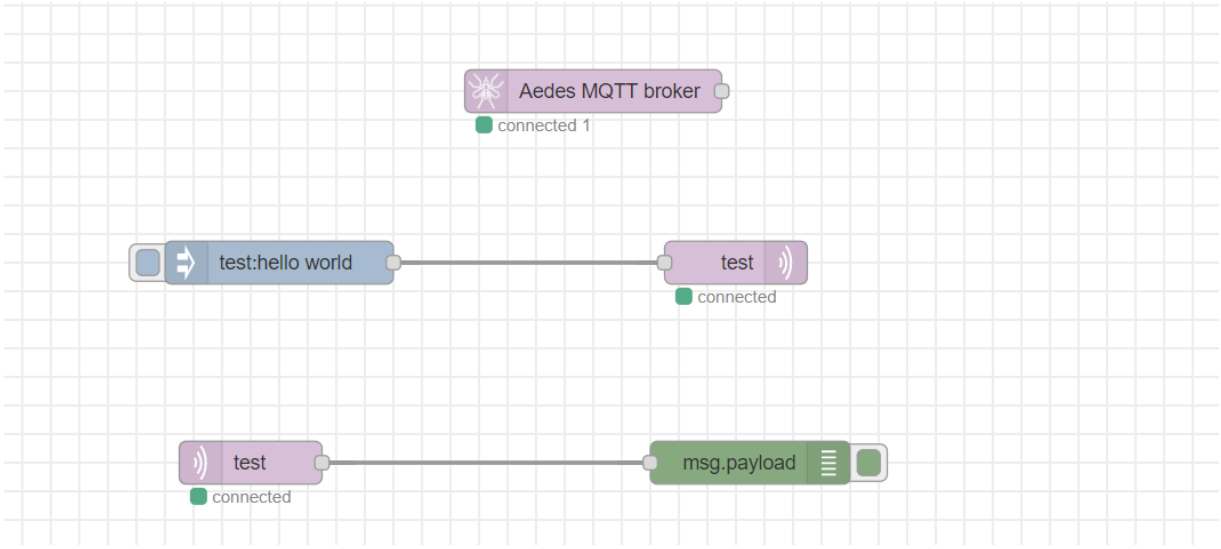
Now we can deploy our model by “Deploy” button on top right corner.



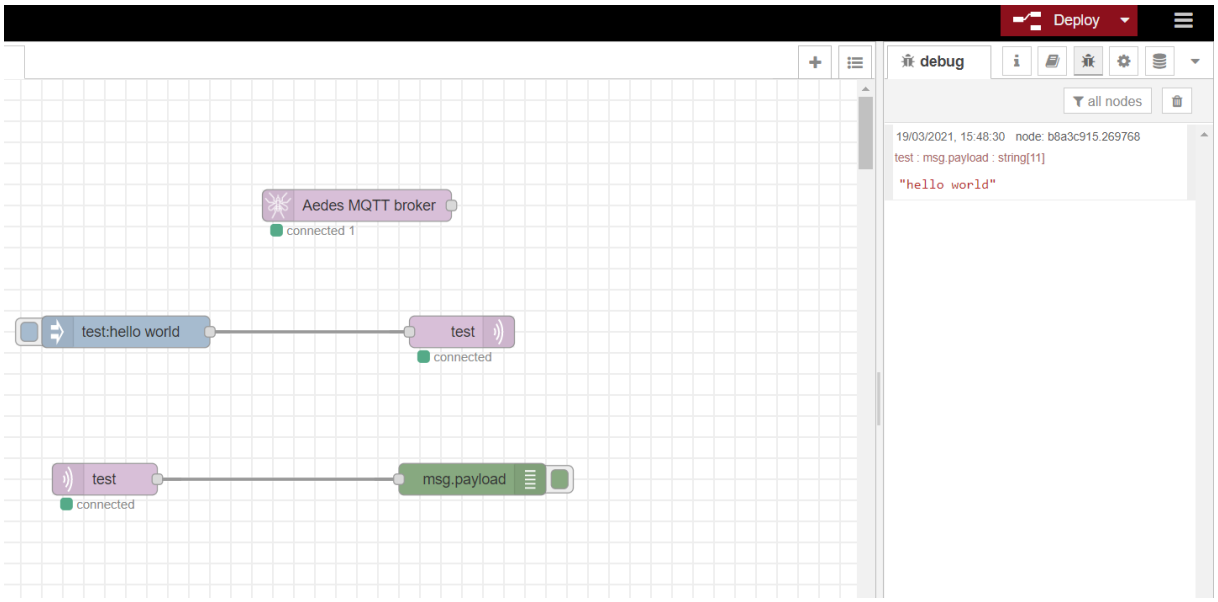
We can see “Successfully deployed” message.



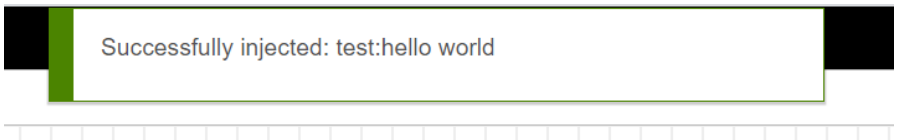
And in a moment, we will be able to see connected status if there is no error.



In debug console, which can be opened from right panel, we can see the output received by debug node after clicking inject node to send the text.



We can see the status on top too.



CONCLUSION

In this practical, we learned about Node red and MQTT. We implemented the MQTT connection using Node red.

PRACTICAL 7

AIM: Simulate CoAP protocol in Contiki-OS

Proteus Simulation for Home automation and smart city.

THEORY:

Proteus:

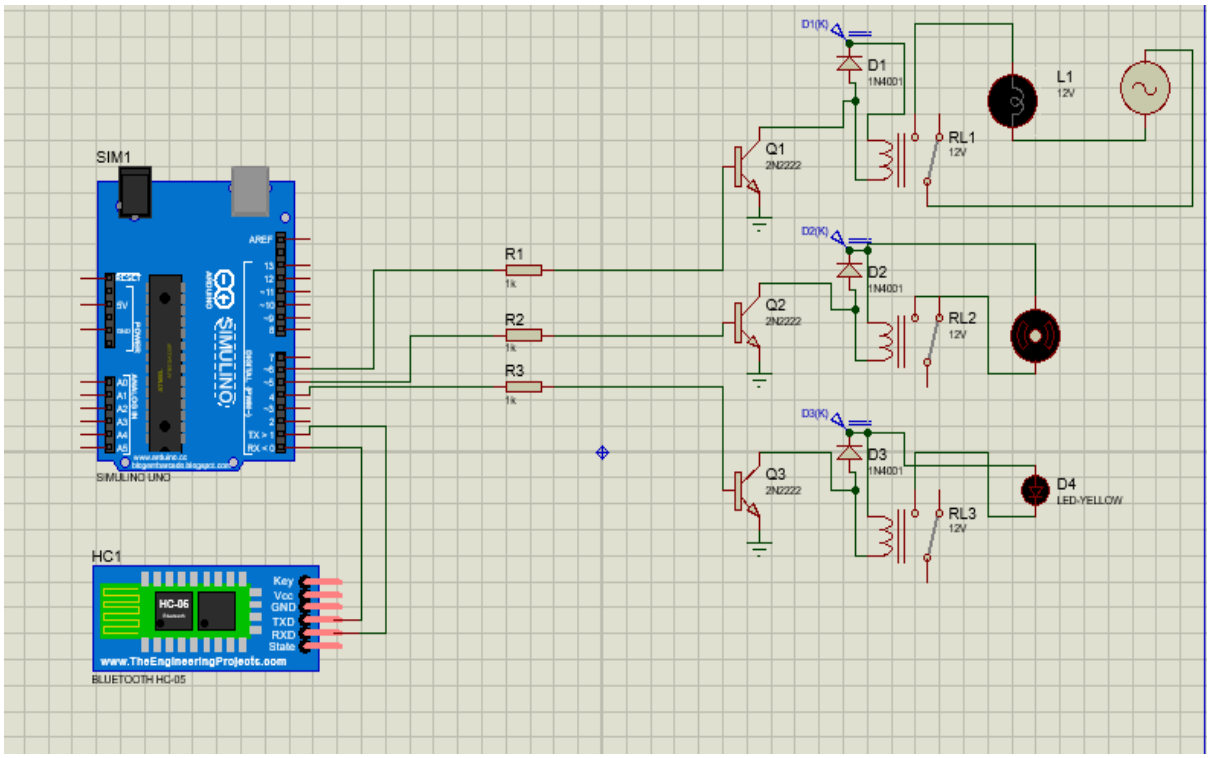
- The Proteus Design Suite is a proprietary software tool suite used primarily for electronic design automation.
- The software is used mainly by electronic design engineers and technicians to create schematics and electronic prints for manufacturing printed circuit boards.
- The Proteus Design Suite is a Windows application for schematic capture, simulation, and PCB (Printed Circuit Board) layout design.
- It can be purchased in many configurations, depending on the size of designs being produced and the requirements for microcontroller simulation.
- All PCB Design products include an autorouter and basic mixed mode SPICE simulation capabilities.
- The micro-controller simulation in Proteus works by applying either a hex file or a debug file to the microcontroller part on the schematic.
- It is then co-simulated along with any analog and digital electronics connected to it.
- This enables its use in a broad spectrum of project prototyping in areas such as motor control, temperature control and user interface design.
- It also finds use in the general hobbyist community and, since no hardware is required, is convenient to use as a training or teaching tool.

.

PRACTICAL:

Proteus has easy to use interface where we can drag and drop the components.

We will make the following topology.



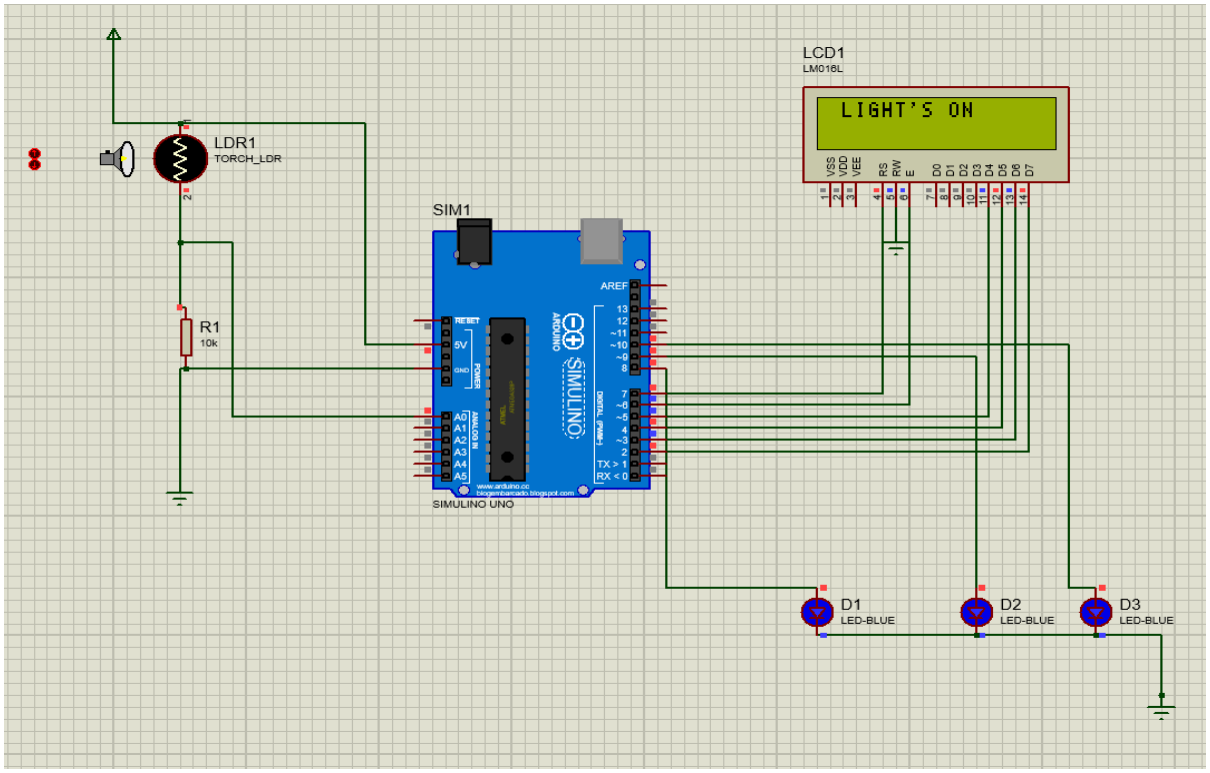
Now we need to upload the code in Arduino.
We will make the Hex file of the following code and upload it into Arduino.

```
String voice;
void setup() {
Serial.begin(9600);
pinMode(6, OUTPUT);
pinMode(5, OUTPUT);
pinMode(4, OUTPUT);
}
void loop() {
while(Serial.available()){
delay(3);
  char c = Serial.read();
  voice+=c;}

if(voice.length() >0){
  Serial.println(voice);
  if(voice == "light on")
  {digitalWrite(6, HIGH);}
  else if(voice == "light off")
  {digitalWrite(6, LOW);}
  else if(voice == "fan on")
  {digitalWrite(5, HIGH);}
  else if(voice == "fan off")
  {digitalWrite(5, LOW);}
  else if(voice == "night lamp on")
  {digitalWrite(4, HIGH);}
```

```
  else if(voice == "night lamp off")
  {digitalWrite(4, LOW);}
  else if(voice == "all on")
  {digitalWrite(4,   HIGH);
  digitalWrite(5,    HIGH);
  digitalWrite(6,   HIGH);}
  else if(voice == "all off")
  {digitalWrite(4, LOW);
  digitalWrite(5,   LOW);
  digitalWrite(6, LOW);}
  voice = "";}
}
```

We will also make a simple mobile app to control the components attached with Arduino.

We can also make the street light system for smart city by using following topology and code.




```
#include<LiquidCrystal.h>
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
void setup() {
pinMode(8,OUTPUT);
pinMode(9,OUTPUT);
pinMode(10,OUTPUT);
lcd.begin(16,2);
  lcd.print("Ldr out = ");
  lcd.setCursor(0,1);
}
```

```
void loop()
{
  int a=analogRead(A0);
  lcd.setCursor(12,0);
  lcd.print(a);
  if(a<1)
  {
    lcd.setCursor(1,0);
    lcd.println("LIGHT'S OFF");
    digitalWrite(8,LOW);
    delay(500);
    digitalWrite(9,LOW);
    delay(500);
    digitalWrite(10,LOW);
  }
  if(a>1)
  {
    lcd.setCursor(1,0);
    lcd.println("LIGHT'S ON");
    digitalWrite(8,HIGH);
    delay(500);
    digitalWrite(9,HIGH);
    delay(500);
    digitalWrite(10,HIGH);
  }
}
```

CONCLUSION

In this practical, we learned about Proteus Design Suit and also implement Proteus Simulation for Home automation.

PRACTICAL - 8

AIM: Implement mini project and connect with IBM Bluemix & Thingspeak for data collection on cloud and plot the graph of it.

Plotting data on thingspeak.com. Take analog input from ESP and pass that data to api.thingspeak.com and prepare 2 online graph.

THEORY:

- Bluemix:
    - Bluemix is the IBM open cloud platform that provides mobile and web developers access to IBM software for integration security transaction and other key functions as well as software from business partners.
    - Built on Cloud Foundry open source technology Bluemix provides pre-built Mobile Backend as a Service (MBaaS) capabilities. Bluemix offers more control to application developers by using its Platform as a Service (PaaS) offering. The goal is to simplify the delivery of an application by providing services that are ready for immediate use and hosting capabilities to enable internal scale development.
    - With the broad set of services and runtimes in Bluemix the developer gains control and flexibility and has access to various data options from predictive analytics to big data.
    - Bluemix provides the following features:
        i. A range of services that enable you to build and extend web and mobile apps fast
        ii. Processing power for you to deliver app changes continuously
        iii. Fit-for-purpose programming models and services
        iv. Manageability of services and applications
        v. Optimized and elastic workloads
        vi. Continuous availability

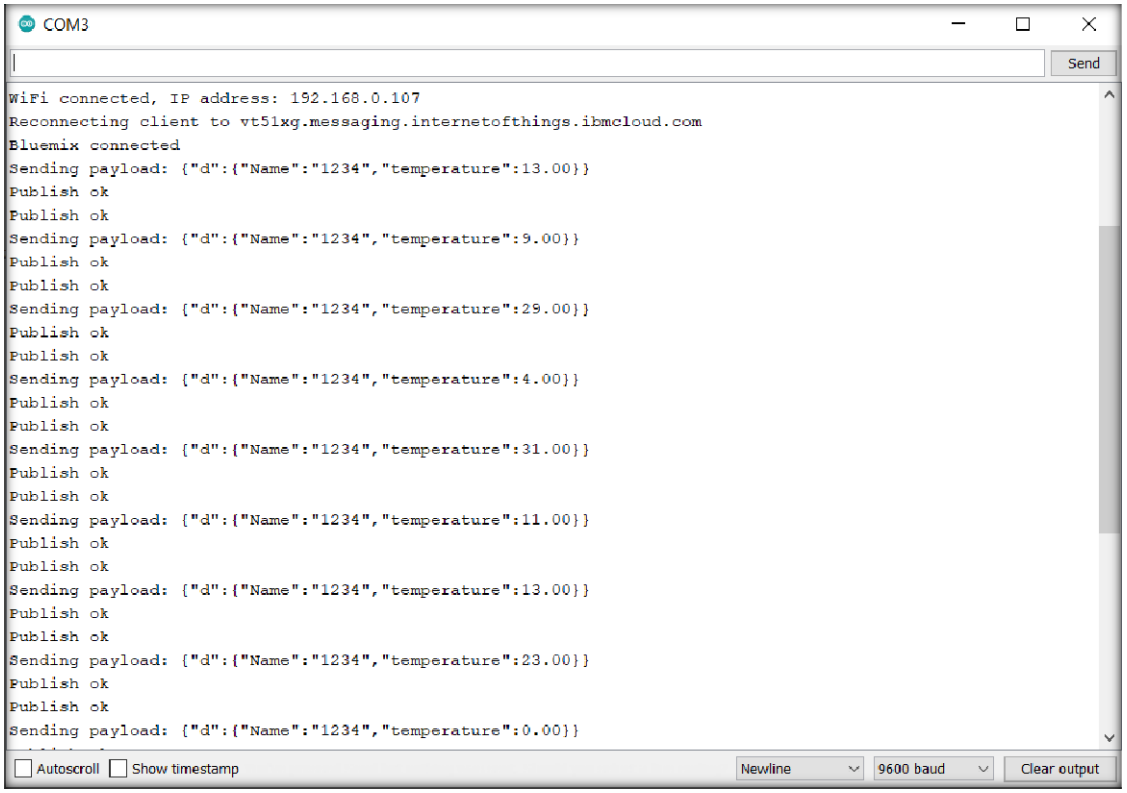
- Thingspeak:
    - ThingSpeak allows you to aggregate, visualize and analyze live data streams in the cloud. Some of the key capabilities of ThingSpeak include the ability to:
        i. Easily configure devices to send data to ThingSpeak using popular IoT protocols.
        ii. Visualize your sensor data in real-time.
        iii. Aggregate data on-demand from third-party sources.
        iv. Use the power of MATLAB to make sense of your IoT data.
        v. Run your IoT analytics automatically based on schedules or events.
        vi. Prototype and build IoT systems without setting up servers or developing web software.
        vii. Automatically act on your data and communicate using third-party services like Twilio or Twitter

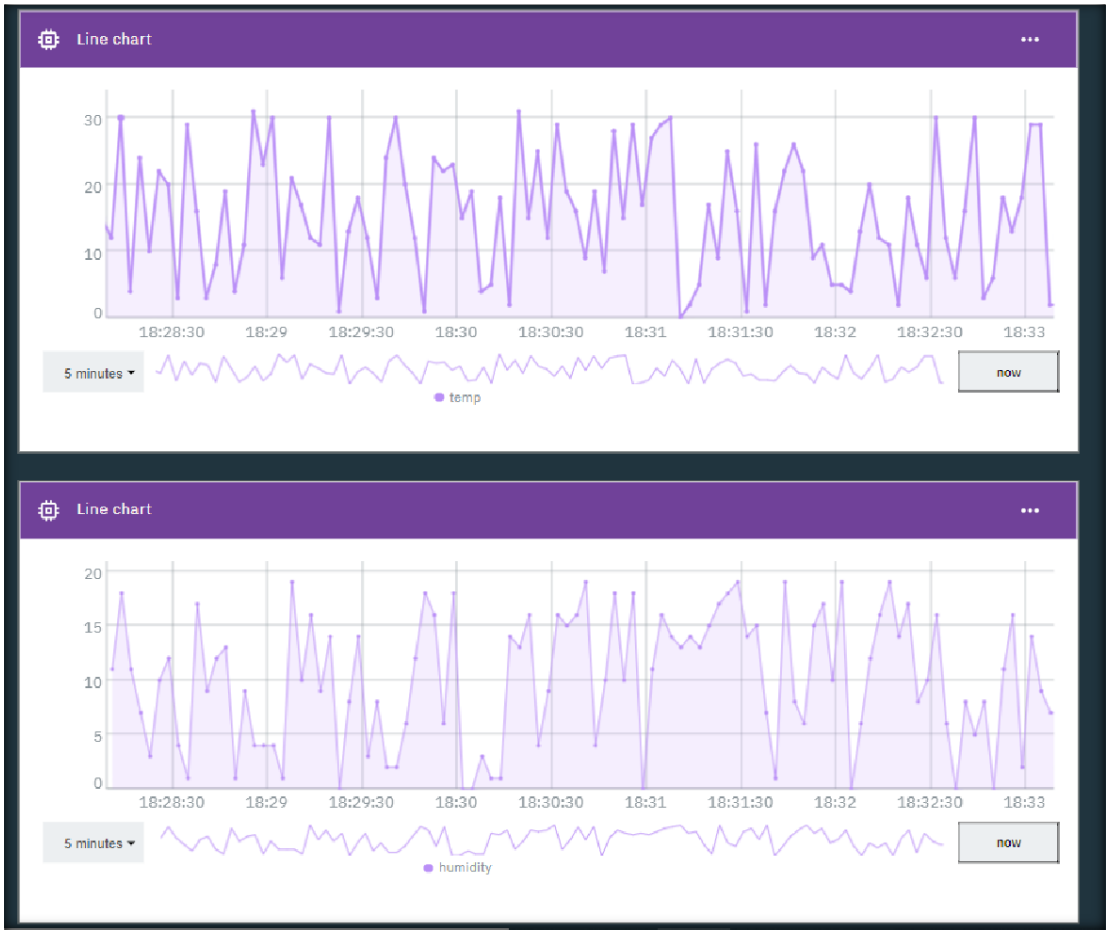
PROGRAM:

Bluemix:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <PubSubClient.h>
#include "DHT.h"

const char* ssid = "speedzone";
const char* password = "1234567890";
#define DHTPIN 4
#define DHTTYPE DHT11
#define ORG "r9xhnn*"
#define DEVICE_TYPE "Node_MCU"
#define DEVICE_ID "1234"
#define TOKEN "sXo4VQIlswL&KUtnbV"

char server[] = ORG ".messaging.internetofthings.ibmcloud.com";
char pubTopic1[] = "iot-2/evt/status1/fmt/json";
char pubTopic2[] = "iot-2/evt/status2/fmt/json";
char authMethod[] = "use-token-auth";
char token[] = TOKEN;
char clientId[] = "d:" ORG ":" DEVICE_TYPE ":" DEVICE_ID;

WiFiClient wifiClient;
PubSubClient client(server, 1883, NULL, wifiClient);
DHT dht(DHTPIN, DHTTYPE);


void setup() {
   Serial.begin(9600);
   dht.begin();
   Serial.println();
   Serial.print("Connecting to ");
   Serial.print(ssid);
   WiFi.begin(ssid, password);
   while (WiFi.status() != WL_CONNECTED) {
     delay(500);
     Serial.print(".");
   }
   Serial.println("");

   Serial.print("WiFi connected, IP address: ");
   Serial.println(WiFi.localIP());

   if (!client.connected()) {
      Serial.print("Reconnecting client to ");
      Serial.println(server);
      while (!client.connect(clientId, authMethod, token)) {
```

```
            Serial.print(".");
            delay(500);
        }
        Serial.println("Bluemix connected");
    }
}

long lastMsg = 0;
void loop() {
    client.loop();
    long now = millis();
    if (now - lastMsg > 3000) {
        lastMsg = now;
        float humidity = dht.readHumidity();
        float temperature = dht.readTemperature();
        String payload = "{\"d\":{\"Name\":\"" DEVICE_ID "\"";
             payload += ",\"temperature\":";
             payload += temperature;
             payload += "}}";

        Serial.print("Sending payload: ");
        Serial.println(payload);

        if (client.publish(pubTopic1, (char*) payload.c_str())) {
            Serial.println("Publish ok");
        } else {
            Serial.println("Publish failed");
        }
        String payload1 = "{\"d\":{\"Name\":\"" DEVICE_ID "\"";
             payload1 += ",\"humidity\":";
             payload1 += humidity;
             payload1 += "}}";

         if (client.publish(pubTopic2, (char*) payload1.c_str())) {
            Serial.println("Publish ok");
        } else {
            Serial.println("Publish failed");
        }
    }
}
```

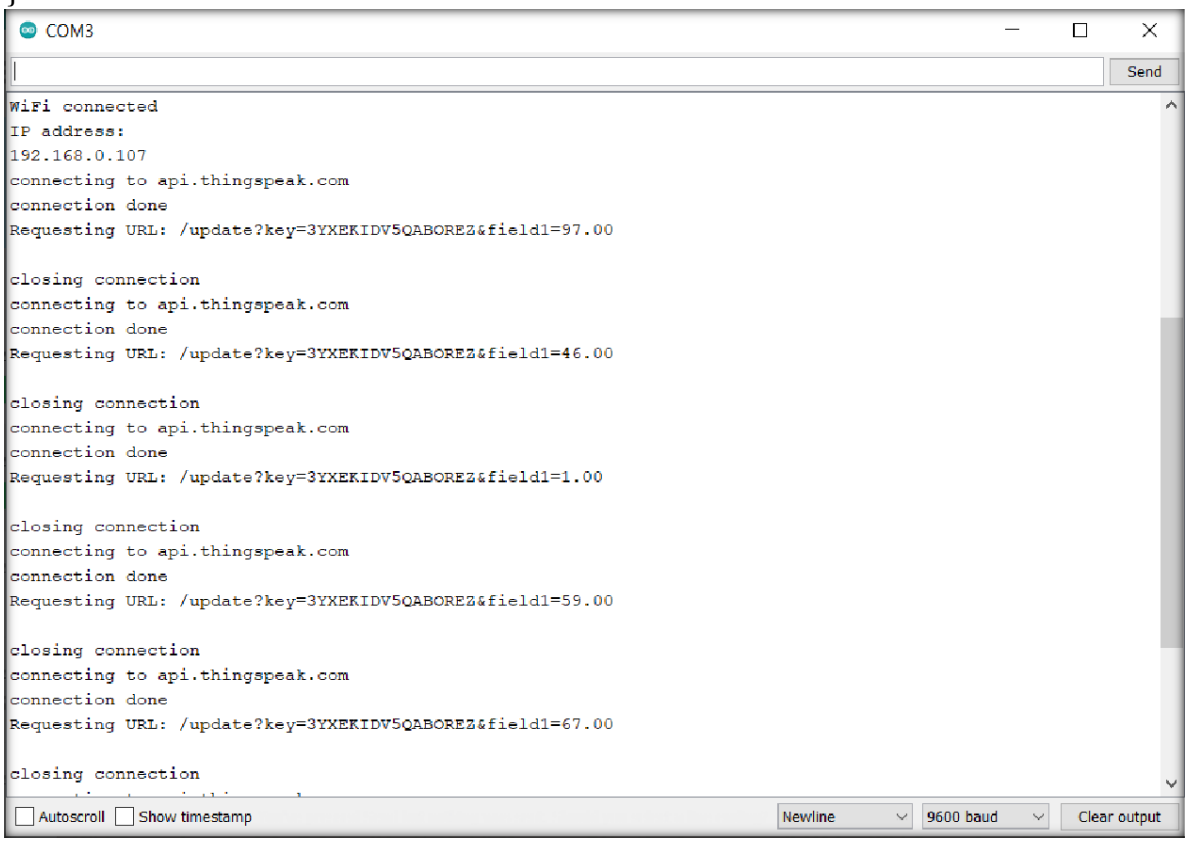

COM3
Send
WiFi connected, IP address: 192.168.0.107
Reconnecting client to vt51xg.messaging.internetofthings.ibmcloud.com
Bluemix connected
Sending payload: {"d":{"Name":"1234","temperature":13.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":9.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":29.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":4.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":31.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":11.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":13.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":23.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":0.00}}
Autoscroll
Show timestamp
Newline
9600 baud
Clear output

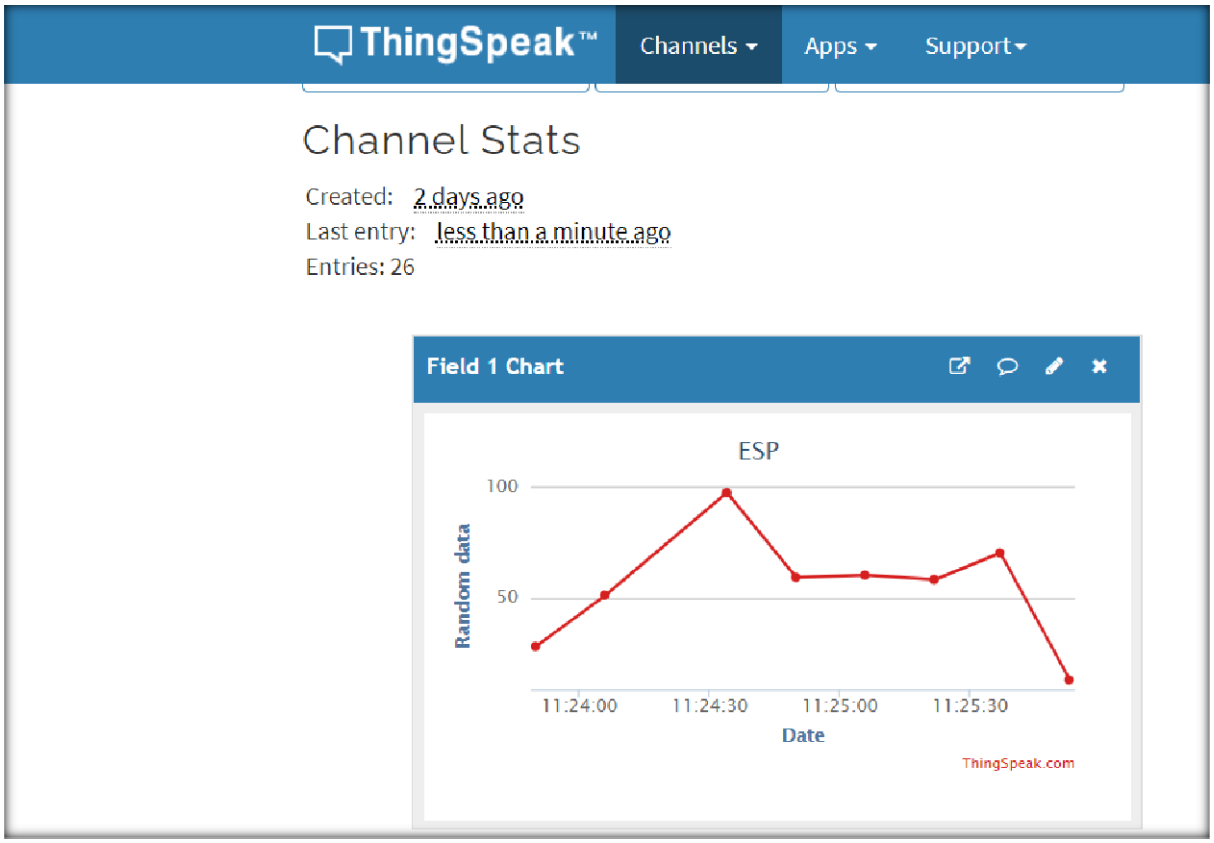

Line chart
30
20
10
0
18:28:30
18:29
18:29:30
18:30
18:30:30
18:31
18:31:30
18:32
18:32:30
18:33
5 minutes
now
temp
Line chart
20
15
10
5
0
18:28:30
18:29
18:29:30
18:30
18:30:30
18:31
18:31:30
18:32
18:32:30
18:33
5 minutes
now
humidity

Thingspeak:

```
#include <ESP8266WiFi.h>

#define SENSOR A0

const char* ssid     = "Rudrabarad"; //WiFiSSID
const char* password = "Rab@0412";  //PASSWORD

const char* host = "api.thingspeak.com";
// Data URL: https://thingspeak.com/channels/XXXXX/
const char* privateKey = "3YXEKIDV5QABOREZ"; //ENTER_YOUR_PRIVATE_KEY

void setup() {
  Serial.begin(9600);
  delay(10);
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

float value = 0;

void loop() {
  delay(5000);
  value = analogRead(A0);

  Serial.print("connecting to ");
  Serial.println(host);

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  const int httpPort = 80;
  if (!client.connect(host, httpPort)) {
    Serial.println("connection failed");
    return;
  }
```

```
  Serial.println("connection done");
  // We now create a URI for the request
  String url = "/update?";
  url += "key=";
  url += privateKey;
  url += "&field1=";
  url += value;


  Serial.print("Requesting URL: ");
  Serial.println(url);

  // This will send the request to the server
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  delay(10);

  // Read all the lines of the reply from server and print them to Serial
  while (client.available()) {
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");
}
```





CONCLUSION:

In this practical, we learned about IBM bluemix and Thingspeak platforms. We uploaded data to both the platforms and plotted the graph.

PRACTICAL - 9

AIM: Uses of Mishellium gateway for Zigbee Communication.

AICTE Project Demo and hardware study of Waspmote, Libellium Gateway, Zigbeemodule and various water sensors..

THEORY:

- **Waspmote:**

  - Waspmote hardware architecture has been specially designed to work with extremely low consumption. Digital switches allow to turn on and off any of the sensor interfaces as well as the radio modules. Three different sleep modes make Waspmote the lowest consumption IoT platform in the market (7 µA).
  - Features
    - Ultra low power (7µA)
    - 120+ sensors integrated on 8 Sensor Boards
    - 15 radio technologies:
    - Long range: 4G/NB-IoT/Cat-M/LoRaWAN/LoRa/Sigfox/ 868 MHz / 900 MHz
    - Medium range: ZigBee 3 / 802.15.4 / DigiMesh / WiFi
    - Short range: RFID-NFC / Bluetooth 2.1 / BLE
    - Over the Air Programming (OTA)
    - Open source SDK and API
    - Encryption libraries (AES, RSA, MD5, SHA Hash)
    - Certified encapsulated line (Plug & Sense!)
    - Industrial Protocols: RS-485, Modbus, CAN Bus, 4-20 mA

- **Libellium gateway:**

  - Meshlium is the IoT Gateway to connect any sensor to any cloud platform.
  - Meshlium is the most recommended option for outdoor networks since it is encased in a rugged, waterproof enclosure which protects it from the harshest conditions.
  - It can receive, parse and store frames in its local database. Meshlium can also forward sensor data directly to the Internet via Ethernet or GPRS/4G protocols, depending on the connectivity options available in the area.
  - Meshlium is the best Internet Gateway for Waspmote and Plug&Sense devices. It is a Linux-based router, totally modular and specially designed for harsh conditions without compromising flexibility in the installation. Meshlium can directly send sensor data from Waspmote to many 3rd party Cloud platforms.

    - Any scenario
    - Fast configuration
    - Easy installation
    - Easy maintenance
    - Fully certified

- **ZigBee module:**

  - Zigbee is a low-cost, low-power, wireless mesh network standard targeted at battery-powered devices in wireless control and monitoring applications.
  - Zigbee delivers low-latency communication.
  - Zigbee chips are typically integrated with radios and with microcontrollers.
  - Zigbee operates in the industrial, scientific and medical (ISM) radio bands: 2.4 GHz in most jurisdictions worldwide; though some devices also use 784 MHz in China, 868 MHz in Europe and 915 MHz in the US and Australia, however even those regions and countries still use 2.4 GHz for most commercial Zigbee devices for home use.
  - Data rates vary from 20 kbit/s (868 MHz band) to 250 kbit/s (2.4 GHz band).
  - Zigbee builds on the physical layer and media access control defined in IEEE standard 802.15.4 for low-rate wireless personal area networks (WPANs).
  - The specification includes four additional key components: network layer, application layer, Zigbee Device Objects (ZDOs) and manufacturer-defined application objects.
  - ZDOs are responsible for some tasks, including keeping track of device roles, managing requests to join a network, as well as device discovery and security.
  - The Zigbee network layer natively supports both star and tree networks, and generic mesh networking.
  - Every network must have one coordinator device. Within star networks, the coordinator must be the central node.
  - Both trees and meshes allow the use of Zigbee routers to extend communication at the network level.
  - It builds on the basic security framework defined in IEEE 802.15.4.
  - Zigbee protocols are intended for embedded applications requiring low power consumption and tolerating low data rates.
  - The resulting network will use very little power—individual devices must have a battery life of at least two years to pass certification.
  - Typical application areas include:
    - Home automation
    - Wireless sensor networks
    - Industrial control systems
    - Embedded sensing
    - Medical data collection
    - Smoke and intruder warning
    - Building automation
    - Remote wireless microphone configuration
  - Zigbee is not for situations with high mobility among nodes. Hence, it is not suitable for tactical ad hoc radio networks in the battlefield, where high data rate and high mobility is present and needed.

Water Sensors:

A water sensor is a device used in the detection of the water level for various applications. Water sensors can come in several variations that include ultrasonic sensors, pressure transducers, bubblers, and float sensors.

- **Chlorine residual sensor:**

  - Free chlorine is the most important disinfectant in water treatment due to its easy handling and strong disinfecting effect. Free chlorine sensors are applied in:

    - Drinking water - to ensure sufficient disinfection
    - Food - to provide hygienic bottling and packaging
    - Pool water - to dose disinfectant efficiently

  - Chlorine dioxide is more and more becoming a disinfectant of choice since it is less corrosive and independent from the pH value. Chlorine dioxide sensors are applied in:

    - Cooling systems or towers
    - Drinking water
    - Wash water for packed vegetables
    - Desalination plants to prevent ClO2 from disturbing reverse osmosis

  - Total chlorine is a good indicator of residual disinfectants in discharge water. The sensors are used in WWTPs:

    - To measure the effluent water's disinfection status
    - To control reuse of water

  - Sensors for chlorine dioxide measurement feature a working electrode, which is separated from the medium by a thin membrane.
  - Chlorine dioxide coming from the medium diffuses through this membrane and is reduced at the working electrode.
  - The circuit is completed by means of the counter electrode and the electrolyte.
  - The electron reduction at the working electrode is proportional to the concentration of chlorine dioxide in the medium.
  - This process works in a wide pH and temperature range.

- **Toc sensor:**

  - TOC refers to a Total Organic Carbon analyzer, which utilizes a catalytic oxidation combustion technique at high temperature (the temperature raises up to 720 ºC), to convert organic carbon into CO2.
  - The CO2 generated by oxidation is measured with a Non-dispersive Infra-Red (NDIR) sensor.
  - By using special kits and (dilution) methods the device can be applied to determine the carbon concentration over an extremely broad range (theoretically from 4µg/L to 30 000mg/L), from pure drinking water to sea water with sludge.
  - In addition, it is possible to indirectly determine the fraction of IC (= "inorganic carbon") arising from dissolved CO2 and acid salts containing carbon.

- **Turbidity Sensor:**

  - Global Water's Turbidity Sensor is a highly accurate submersible instrument for in-situ environmental or process monitoring.
  - Applications for the turbidity sensors include: water quality testing and management, river monitoring, stream measurement, reservoir water quality testing, groundwater testing, water and wastewater treatment, and effluent and industrial control.
  - In accordance with USEPA Method 180.1 for turbidity measurement, the Turbidity Sensors are a 90 degree scatter nephelometer.
  - The turbidity sensor directs a focused beam into the monitored water. The light beam reflects off particles in the water, and the resultant light intensity is measured by the turbidity sensor's photodetector positioned at 90 degrees to the light beam.
  - The light intensity detected by the turbidity sensor is directly proportional to the turbidity of the water.
  - The turbidity sensors utilize a second light detector to correct for light intensity variations, color changes, and minor lens fouling.
  - For environmental or process monitoring, simply place the turbidity sensor directly in the water and position it where the turbidity is to be monitored.
  - Since the turbidity sensor uses light to detect the water's turbidity ensure that the minimum amount of external light possible is exposed to the monitoring site.

- **Conductivity Sensor:**

  There are two basic sensor styles used for measuring Conductivity: Contacting and Inductive (Toroidal, Electrodeless).

  - When Contacting Sensors are used, the conductivity is measured by applying an alternating electrical current to the sensor electrodes (that together make up the cell constant) immersed in a solution and measuring the resulting voltage. The solution acts as the electrical conductor between the sensor electrodes.
  - With Inductive (also called Toroidal or Electrodeless), the sensing elements (electrode colis) of an inductive sensro do not come in direct contact with the process. These two matched (identical coils) are encapsulated in PEEK (or Teflon) protecting them from the adverse effects of the process.

- **PH sensor:**

  - A pH sensor is one of the most essential tools that’s typically used for water measurements.
  - This type of sensor is able to measure the amount of alkalinity and acidity in water and other solutions.
  - When used correctly, pH sensors are able to ensure the safety and quality of a product and the processes that occur within a wastewater or manufacturing plant.
  - In most cases, the standard pH scale is represented by a value that can range from 0-14.

- When a substance has a pH value of seven, this is considered to be neutral.
- Substances with a pH value above seven represent higher amounts of alkalinity whereas substances with a pH value that's lower than seven are believed to be more acidic.
- For instance, toothpaste typically comes with a pH value of 8-9. On the other hand, stomach acid has a pH value of two.
- The difference between an alkaline substance and an acidic substance is very important for any company that uses a cooling tower, boiler, manufacturing processes, swimming pool control, and various types of environmental monitoring.
- The human body has a standard pH level of 7.4, which is essential for the body to run effectively. If the composition of the body every becomes too acidic or overly alkaline, it will look to return to the neutral state.

- **Orp sensor:**
  - Oxidation-Reduction Potential (ORP) sensors measure the ability of a solution to act as an oxidizing or reducing agent.
  - To achieve accurate results, the correct combination of reference system, junction and shape are important. METTLER TOLEDO offers ORP sensors with smooth metal surfaces and unique reference junctions to ensure dependable measurements, even in dirty samples.

CONCLUSION

In this practical, we learned about different hardware like Waspmote, Libellium module and for communication Zigbee module. We also learned about various water sensors.

PRACTICAL - 10

AIM: Case study on TinyOS in IOT development.

THEORY:

TinyOS is an embedded, component-based operating system and platform for low-power wireless devices, such as those used in wireless sensor networks (WSNs), smartdust, ubiquitous computing, personal area networks, building automation, and smart meters. It is written in the programming language nesC, as a set of cooperating tasks and processes. It began as a collaboration between the University of California, Berkeley, Intel Research, and Crossbow Technology, was released as free and open-source software under a BSD license, and has since grown into an international consortium, the TinyOS Alliance.
TinyOS has been used in space, being implemented in ESTCube-1.
TinyOS is an open-source, BSD-based operating system which uses the nesC programming language to control and manage wireless sensor networks (WSN). The sensor devices (called motes) in such networks are characterized by low power, limited memory and very small form factor.

Why Is TinyOS Useful for Wireless Sensor Networks?

Low-power sensors, due to their limitations in scope, require efficient utilization of resources. TinyOS is essentially built on a "components-based architecture" to reduce code size to around 400 to 500 bytes and an "events-based design" which eliminates the need for even a command shell.
The components-based architecture uses "nesC," which is a C programming language designed for networking embedded systems. Each code snippet consists of simple functions placed within components and complex functions integrating all the components together.
TinyOS also uses an "events-based design" whose objective is to put the CPU to rest when there are no pending tasks. An event can be something such as the triggering of an alert when the temperature of a thermostat rises or falls above a certain value. As soon as the event is over, the sensor motes can go to sleep.
The need for a design like TinyOS is mandatory in applications such as smart transit and smart factories. Because of thousands of sensors, it is important to have a very small memory footprint to reduce power requirements.
Where Is TinyOS Being Used?
At the moment, TinyOS has over 35,000 downloads. Its main applications lie in all kinds of devices which utilize wireless sensor networks.

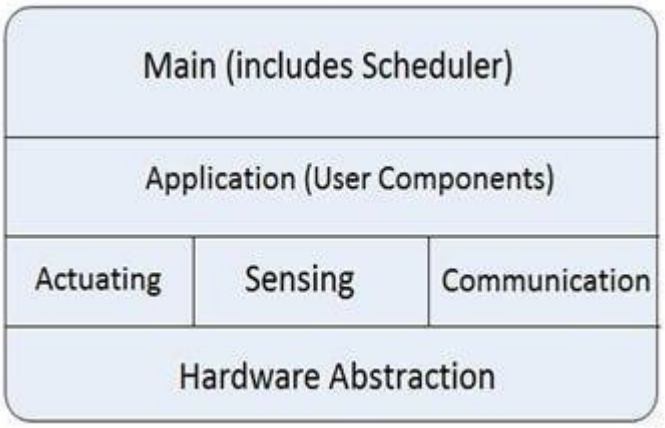
- **Environmental monitoring:**
  - Since each TinyOS system can be embedded in a small sensor, they are useful in monitoring air pollution, forest fires, and natural disaster prevention

- **Smart vehicles:**
  - Smart vehicles are autonomous and can be understood as a network of sensors. These sensors communicate through low-power wireless area networks (LPWAN) which makes TinyOS a perfect fit.

- **Smart cities:**
  - TinyOS is a viable solution for the low-power sensor requirements of smart cities' utilities, power grids, Internet infrastructure and other applications.

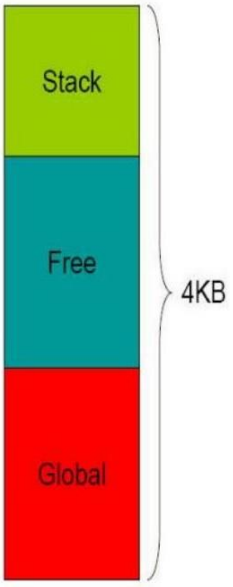
- **Machine condition monitoring:**
  - Machine-to-machine (M2M) applications have many sensor interfaces. It is impossible to assign a complete computing environment to each sensor. TinyOS can perform security, power management and debugging of the sensors.

Tiny OS Structure :



TinyOS models:

1. Data model:
   - **Static Memory Allocation**
     - No Heaps or any other dynamic structures used.
     - Memory requirements determined at compile time.
     - This increases the runtime efficiency.
   - **Global variables**
     - Allocated on per frame basis.
   - **Local Variables**
     - Saved on the stack
     - Defined in the function/method



2. Thread model:
   - **Power-Aware Two-levels Scheduling**
     - Long running tasks and interrupt events
     - Sleep unless tasks in queue, wakeup on event
   - **Tasks**
     - Time-flexible, background jobs
     - Atomic with respect to other tasks
     - Can be preempted by events
   - **Events**
     - Time-critical, shorter duration
     - Last-in first-out semantic (no priority)
     - Can post tasks for deferred execution

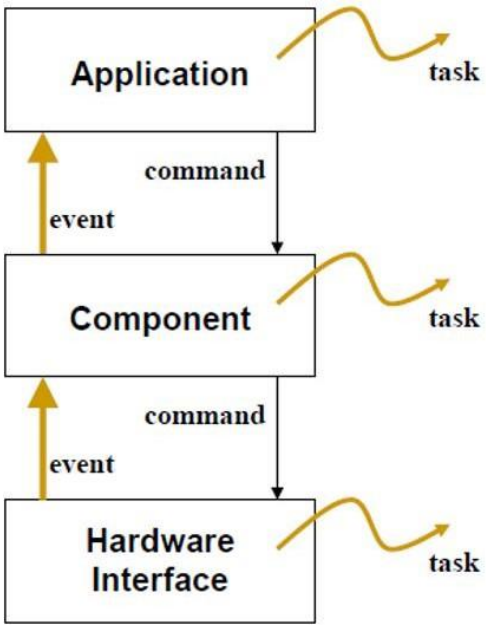
3. Programming model:
   - **Separation construction/composition**
   - **Construction of Modules**

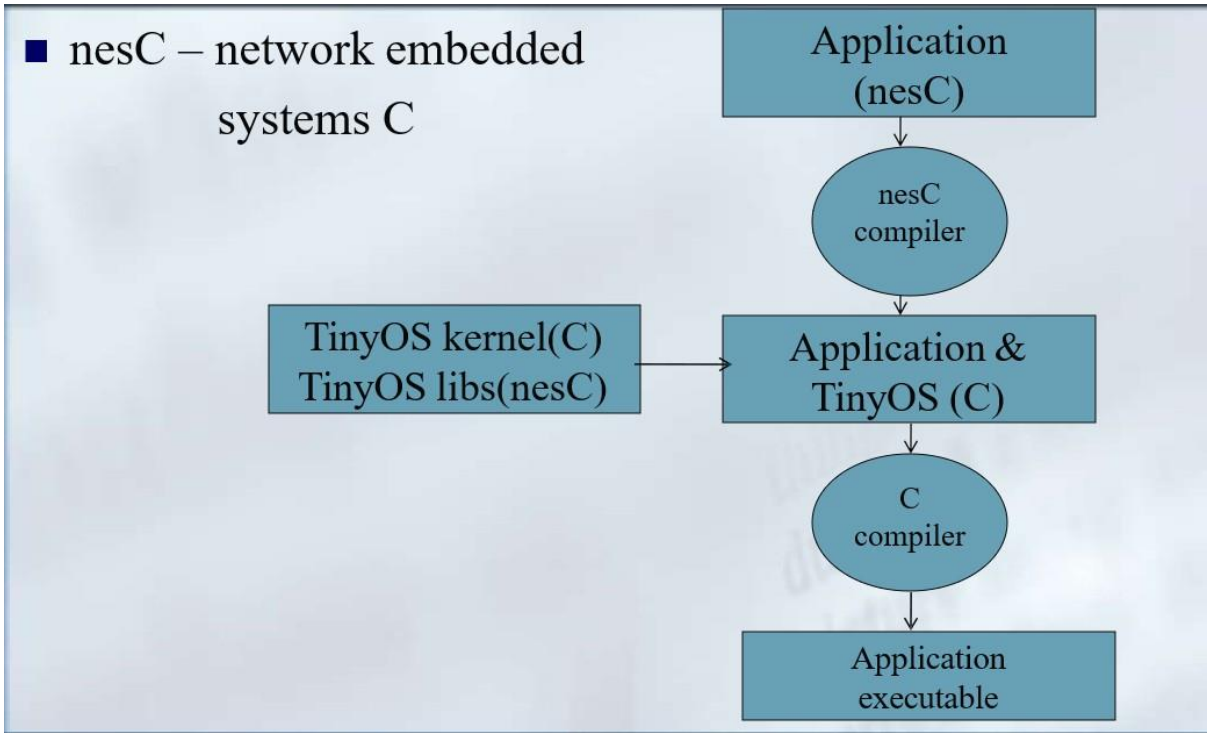
- Modules implementation similar to C coding
- Programs are built out of components
- Each component specifies an interface
- Interfaces are “hooks” for wiring components

- **Composition of Configurations**
  - Components are statically wired together
  - Increases programming efficiency (code reuse) a runtime efficiency.

4. Component model:
   - Components should use and provide bidirectional interfaces.
   - Components should call and implement commands and signal and handle events.
   - Components must handle events of used interfaces and also provide interfaces that must implement commands.

TinyOS basic constructs:





- An extension to the C programming language, to embody the concepts and execution model
- of TinyOS.
- Filename extension .nc
- Static language
- No dynamic memory(malloc)
- No function pointers
- No heap
- Includes task FIFO scheduler.
- Designed to encourage code reuse.

CONCLUSION

In this practical, we learned about features and application of TinyOS and we learned about its models and nesC language.