# PRACTICAL 1

## AIM:

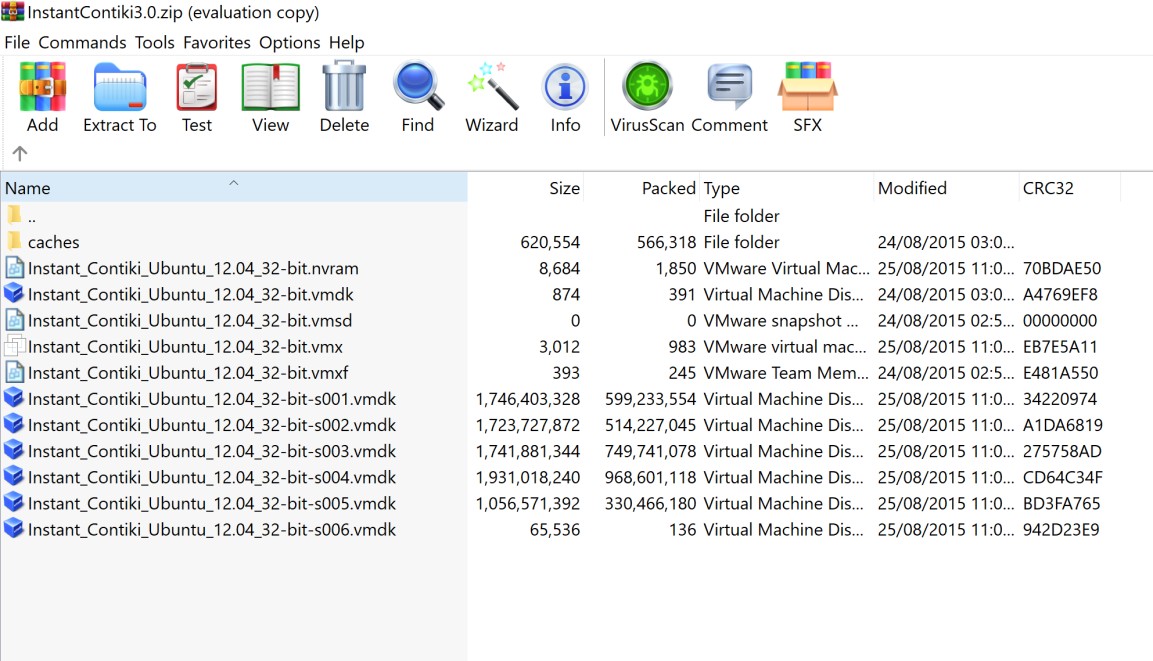
Installation and configuration of Instant Contiki OS with Cooja.

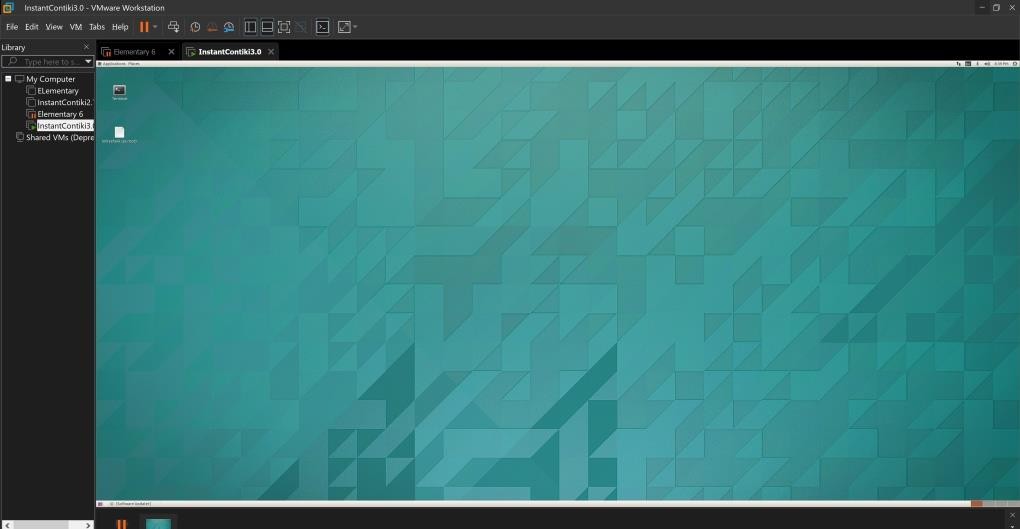
## Contiki OS and Cooja:

Contiki is an operating system for networked, memory-constrained systems with a focus on low-power wireless Internet of Things devices.

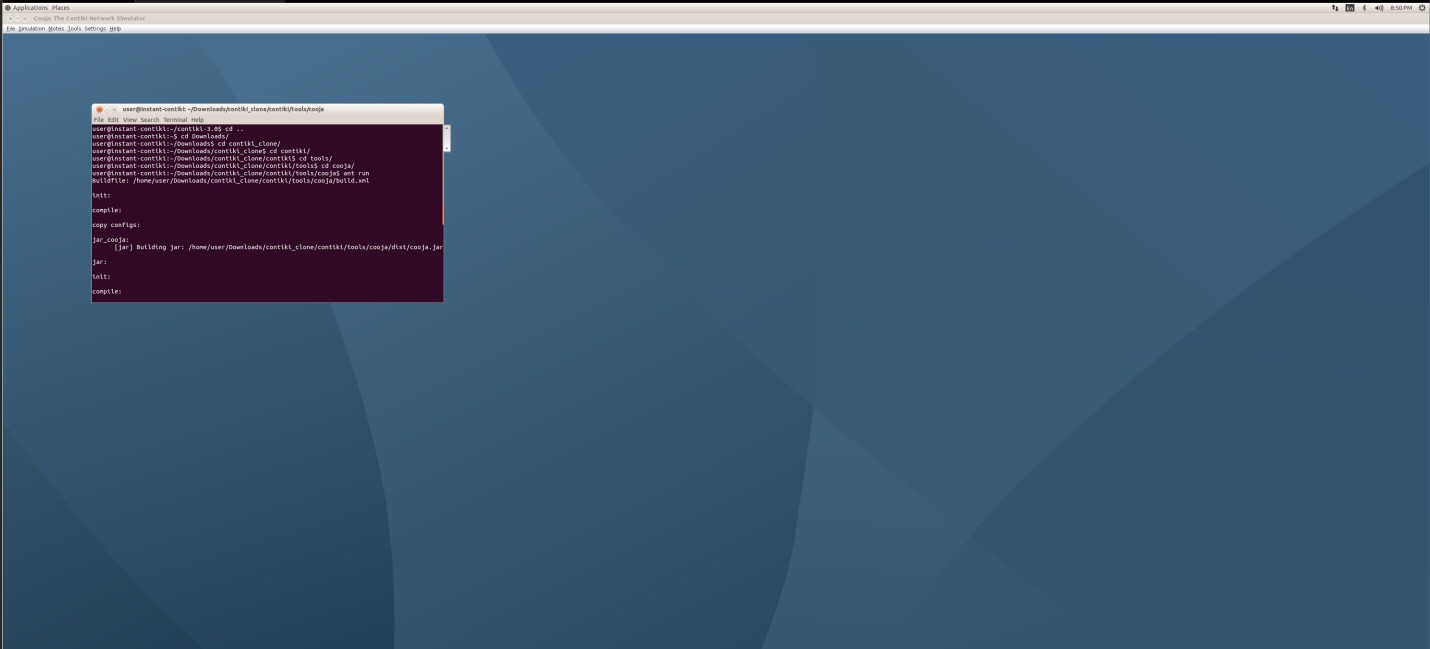
Cooja Simulator is a network simulator specifically designed for Wireless Sensor Networks.

## Steps to configure:

1. Contiki OS comes with cooja simulator and can be downloaded from this link: [https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%2](https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%203.0/InstantContiki3.0.zip/download) [03.0/InstantContiki3.0.zip/download](https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%203.0/InstantContiki3.0.zip/download) Here we get the .vmdk file that can be imported into the virtual machine platform to run the ready os.
2. After importing the .vmdk file into the virtual environment the machine is ready to

run.

1. Using the terminal we get into the path of Cooja and run the command ant run to start cooja simulator and hence we have configured the environment.



## CONCLUSION

In this practical we have successfully configured and installed Instant Contiki OS with Cooja.

# PRACTICAL 2

## AIM:

Study of different types of motes and deploy them using IoTarchitecture. Simulate Hello World program using Cooja.

## Notes:

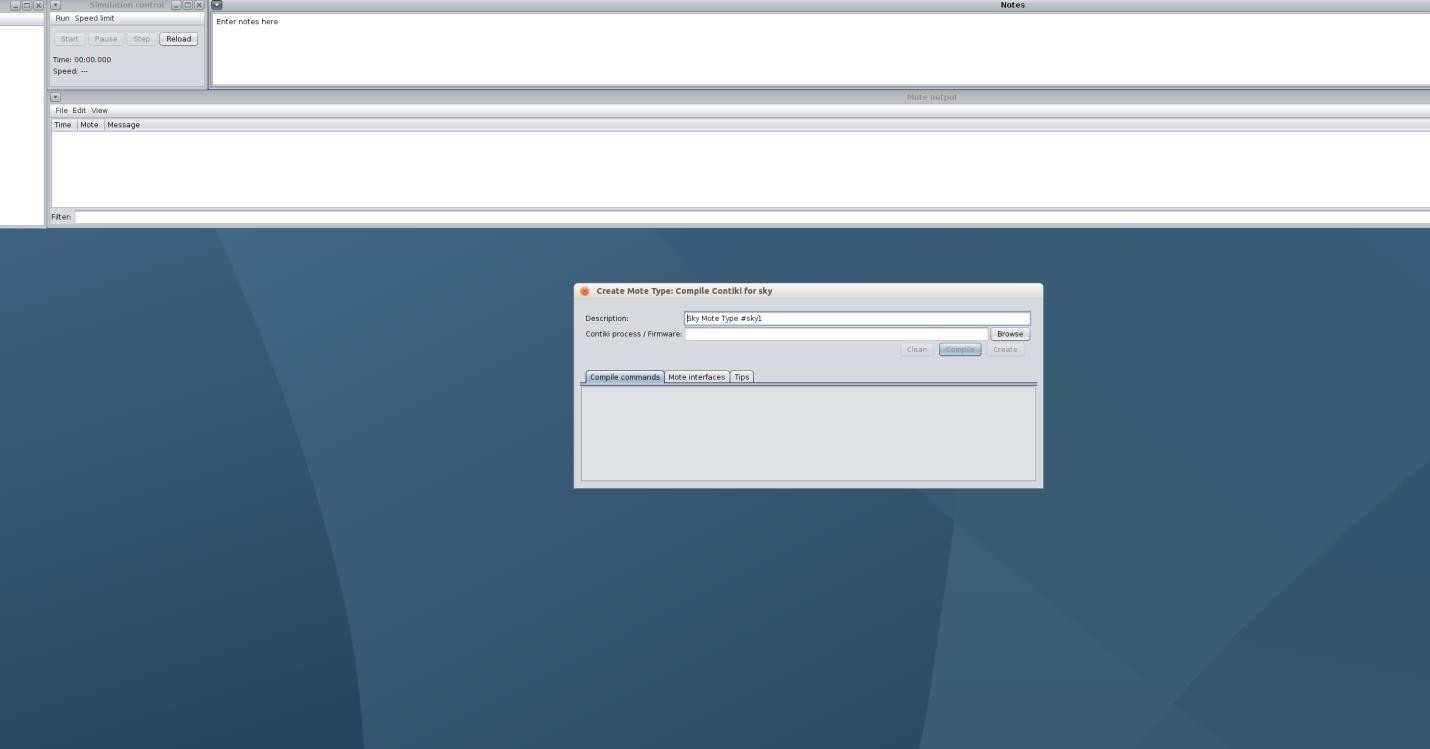
We need multiple sensors and depending on our project type the requirement of motes takes place for information exchange.

Cooja has multiple types of motes that we can select from which include Sky Mote, ESB Mote, Cooja Mote, Z1 Mote, CC430 Mote etc.

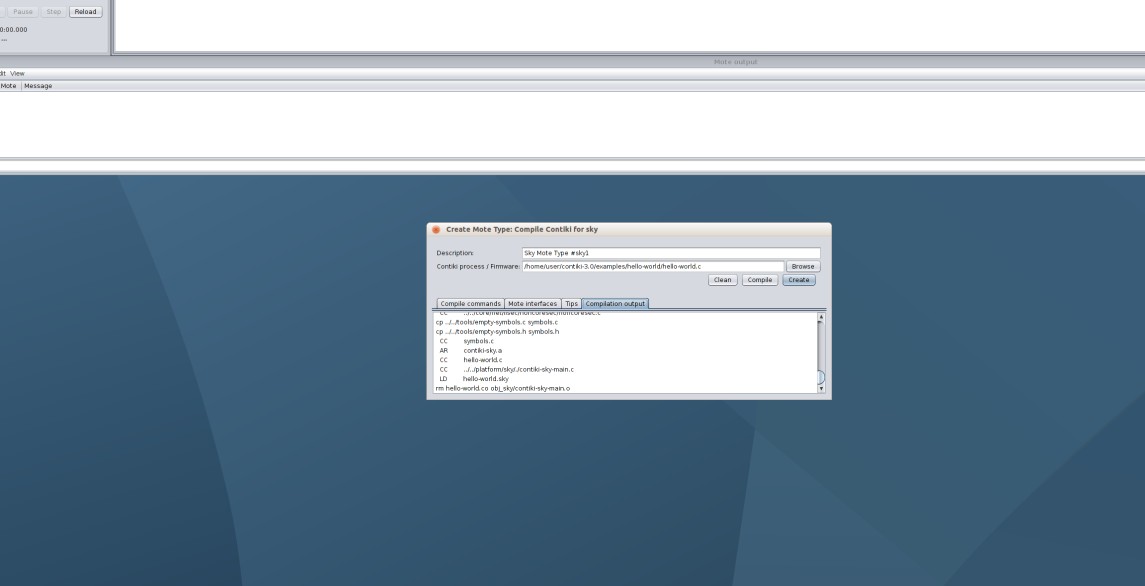
We will use the sky mote type to simulate the hello world program.

## Steps to configure:

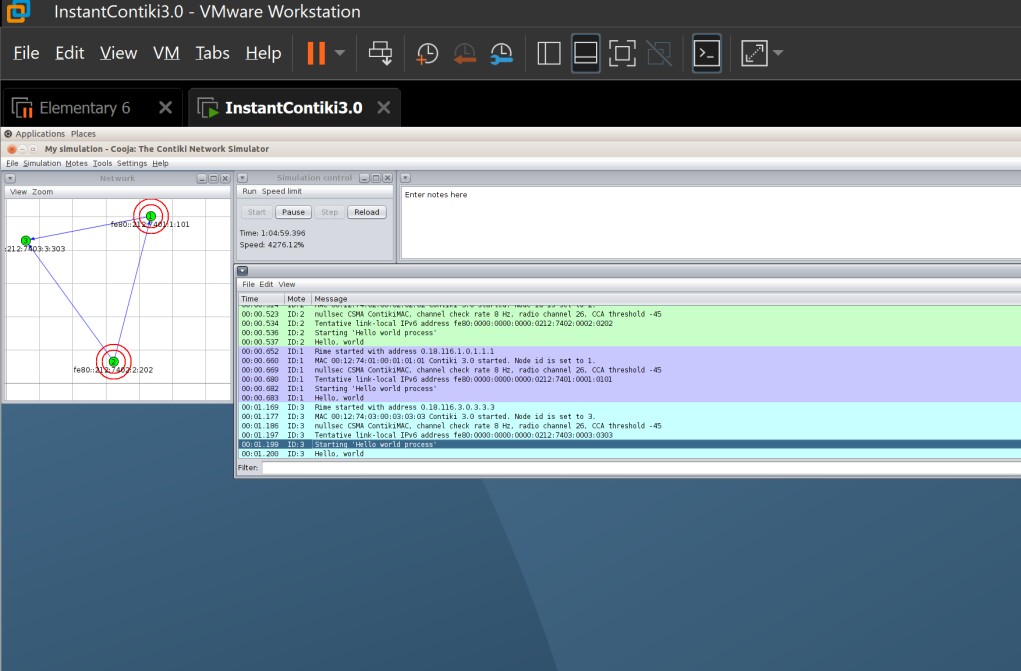
1. After starting cooja we will create a new simulation and to begin we will create a new mote group type i.e Sky Mote and 3 sky motes into our simulation.



1. To run the hello world simulation we will use the ready hello world.c file in the example section of cooja itself and we will build and compile our sky mote type using this file itself.



1. Now we will add 3 motes of the type we just build and run the simulation and see the output of the program we just built.



## CONCLUSION

In this practical we have seen the different types of motes and how they can function in order to communicate and transmit data and also did a simulation of the hello world program.

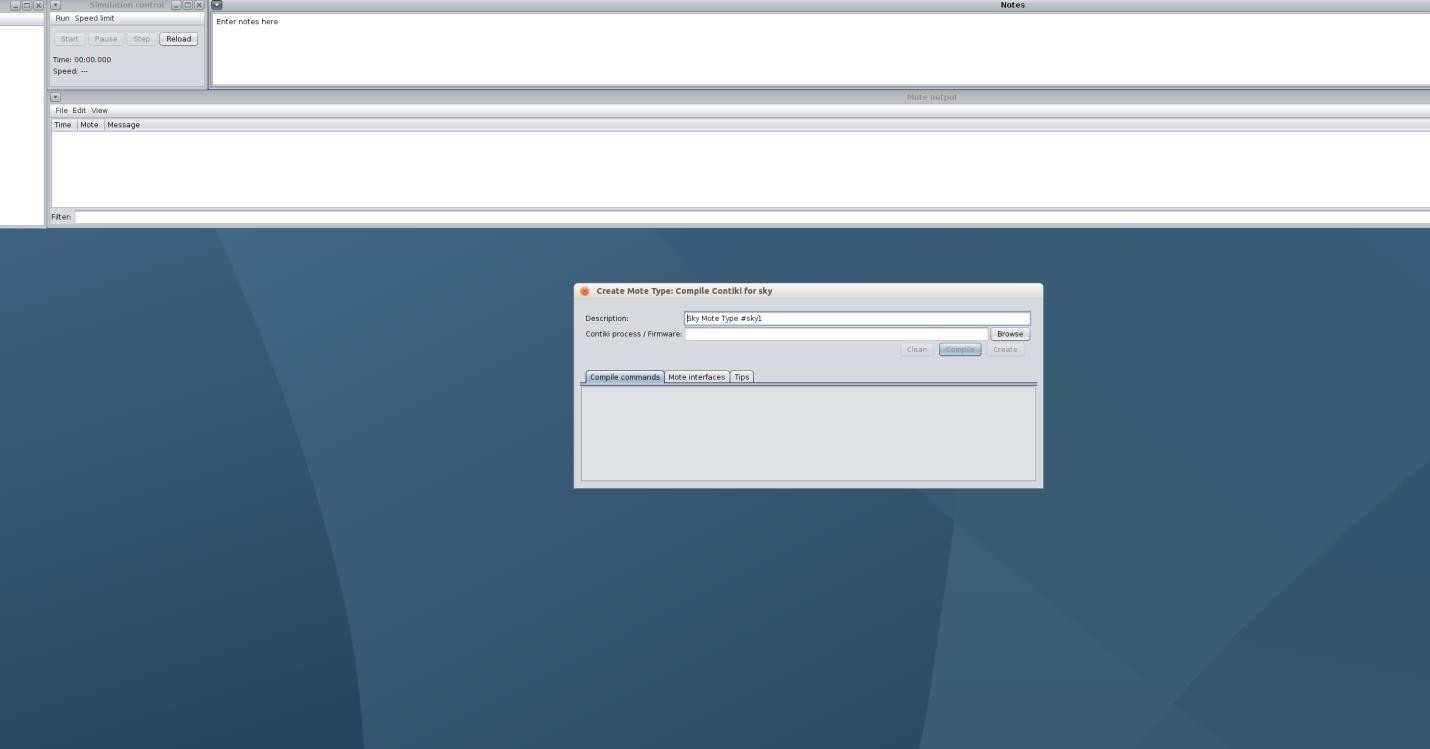
# PRACTICAL 3

## AIM:

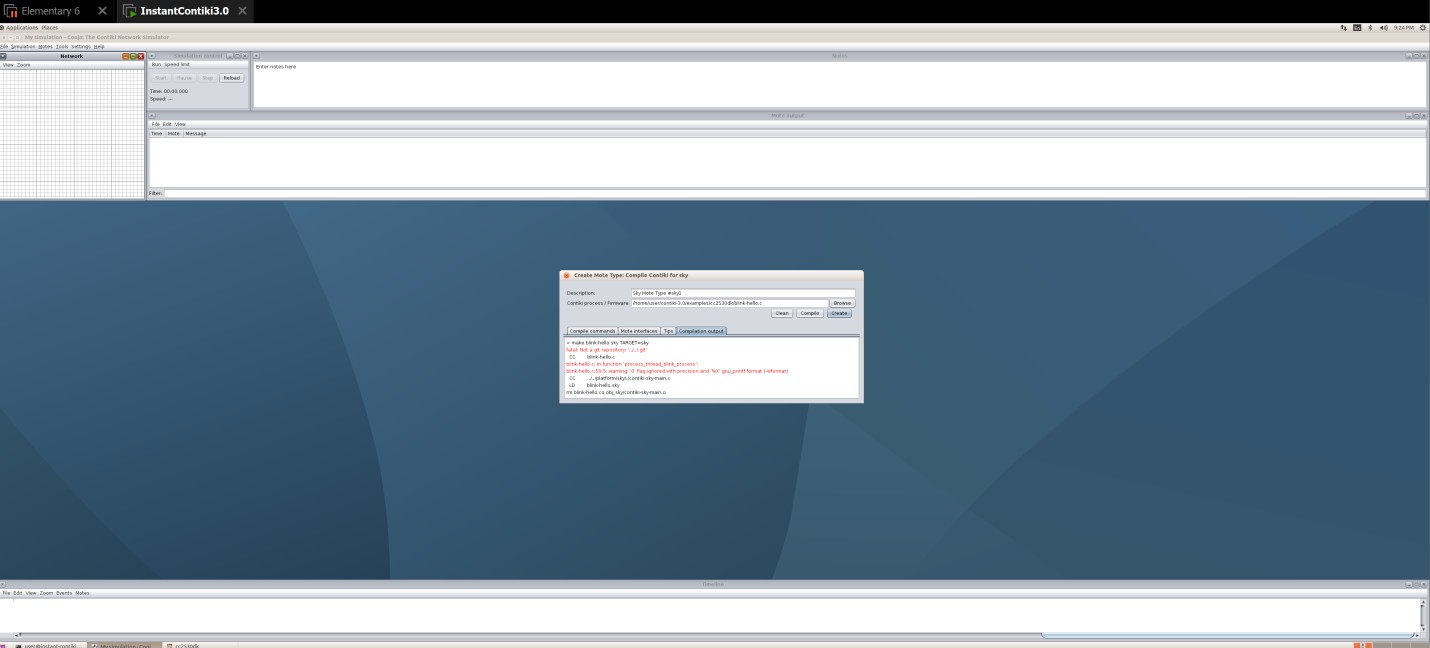
Create a scenario by adding some motes. Do the simulation for the same. Also observe the result for said scenario. When one mote sends the signal then led should turn green while one receives then it should show red color.

## Steps:

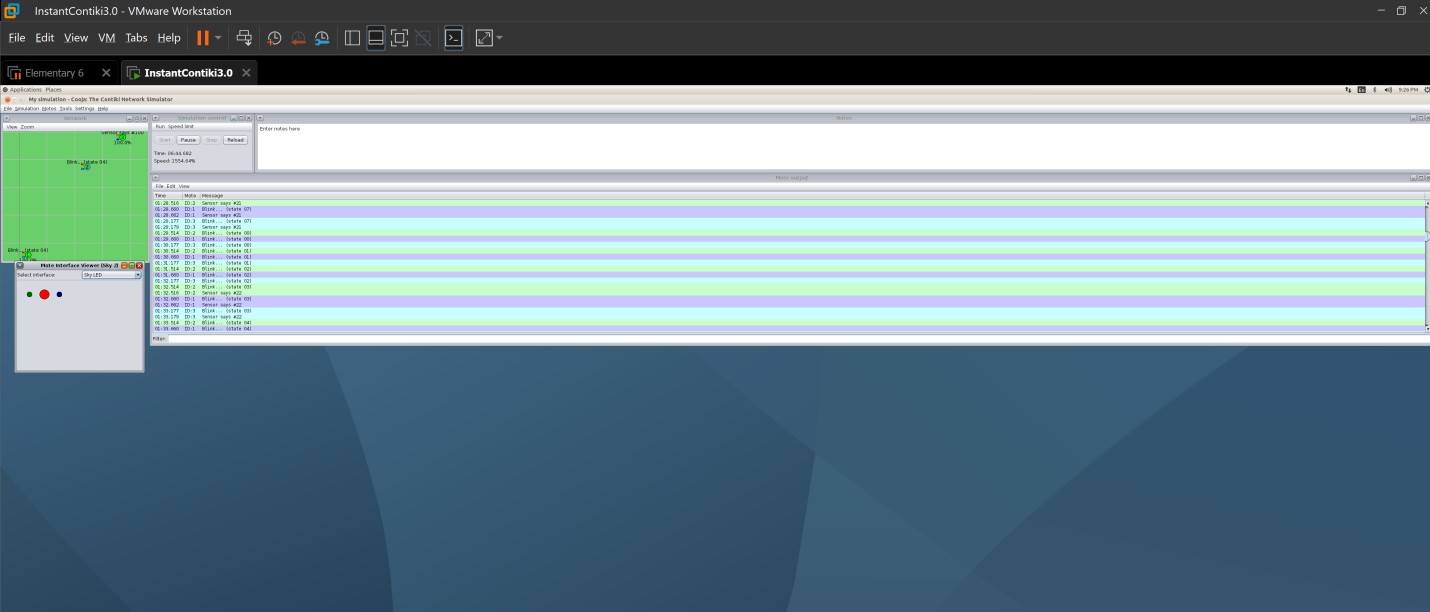
1. After starting cooja we will create a new simulation and to begin we will create a new mote group type i.e. Sky Mote and 3 sky motes into our simulation.



1. To perform led simulation we will use the ready blink hello.c file in the example section of cooja itself and we will build and compile our sky mote type using this file itself.



1. Now we will add 3 motes of the type we just build and run the simulation and see the console log and also the led’s that keep on changing as signal is received.



## CONCLUSION

In this practical we have seen how we can configure a led to behave on sending and receiving signals and this can be further helpful in project implementation.

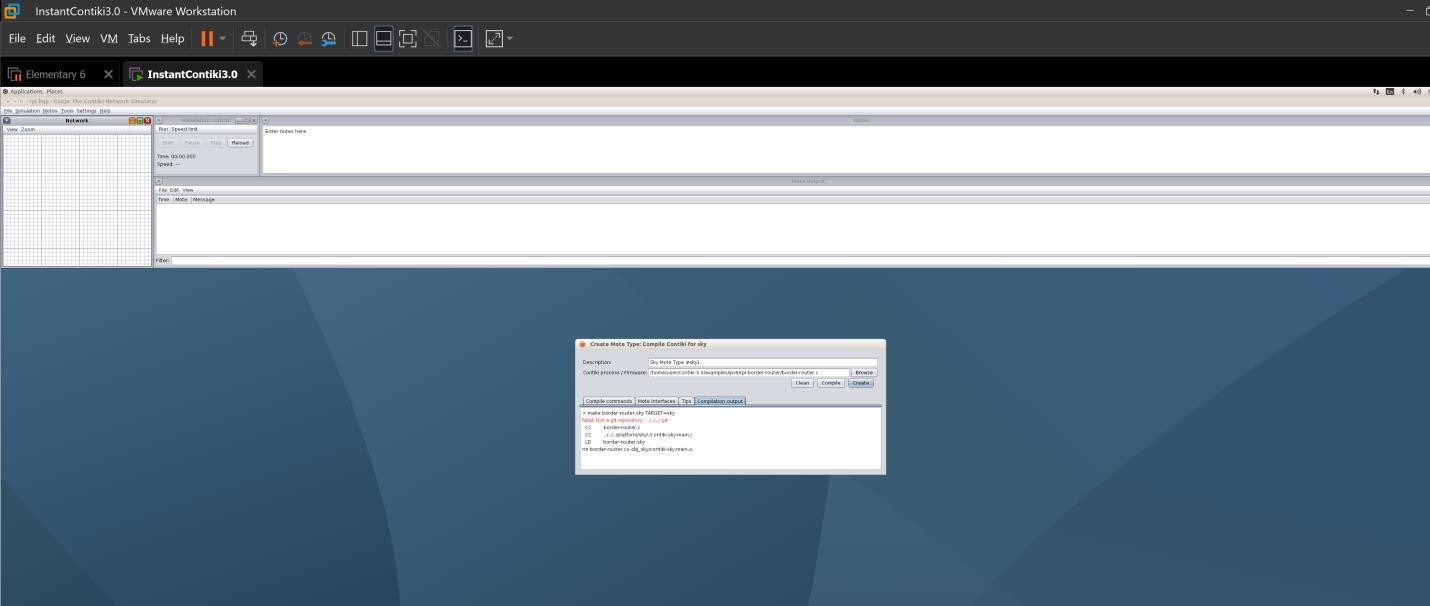
# PRACTICAL 4

## AIM:

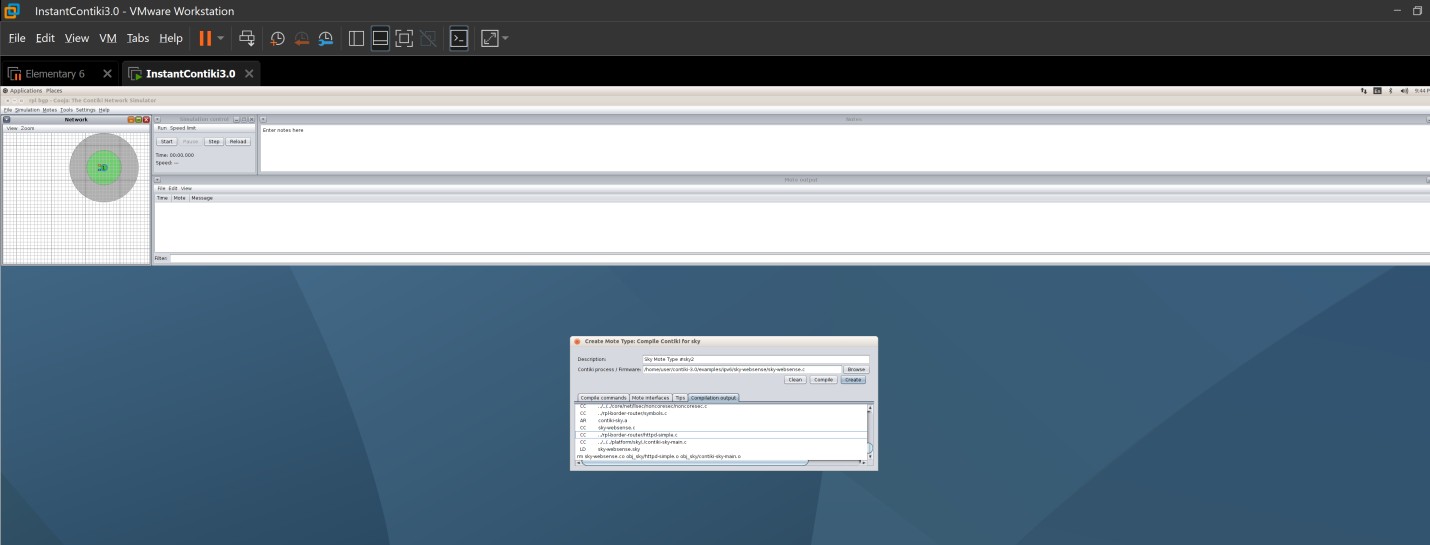
Simulate BGP and RPL protocol in Cooja.

## Steps:

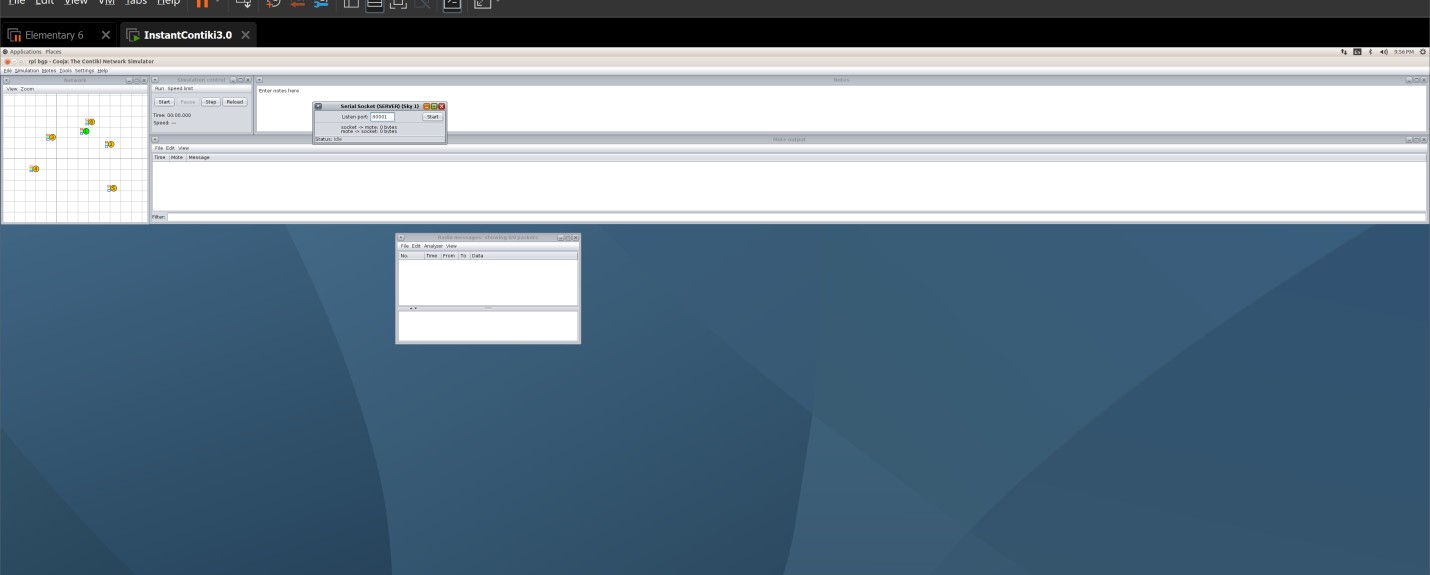
1. After starting cooja we will create a new simulation and to begin we will create a new Sky Mote group that will be the server mote and we will compile this using the rpl- border router.c file.



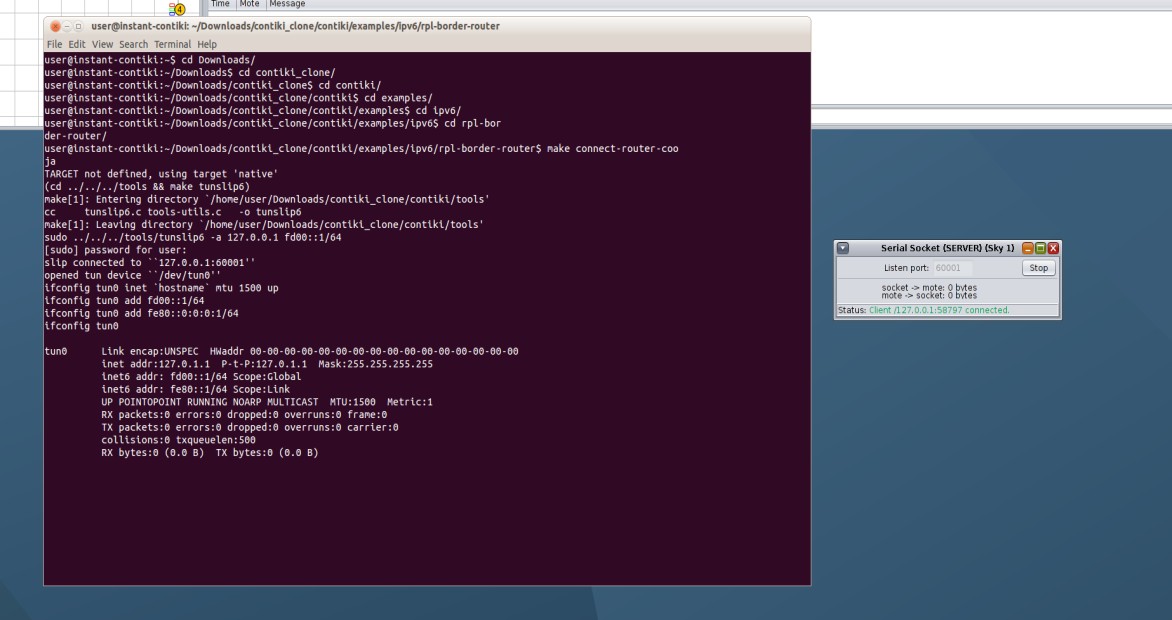
1. We will create another sky mote and this one would be compiled using the sky- websense.c file and we will add 5 motes of this type.



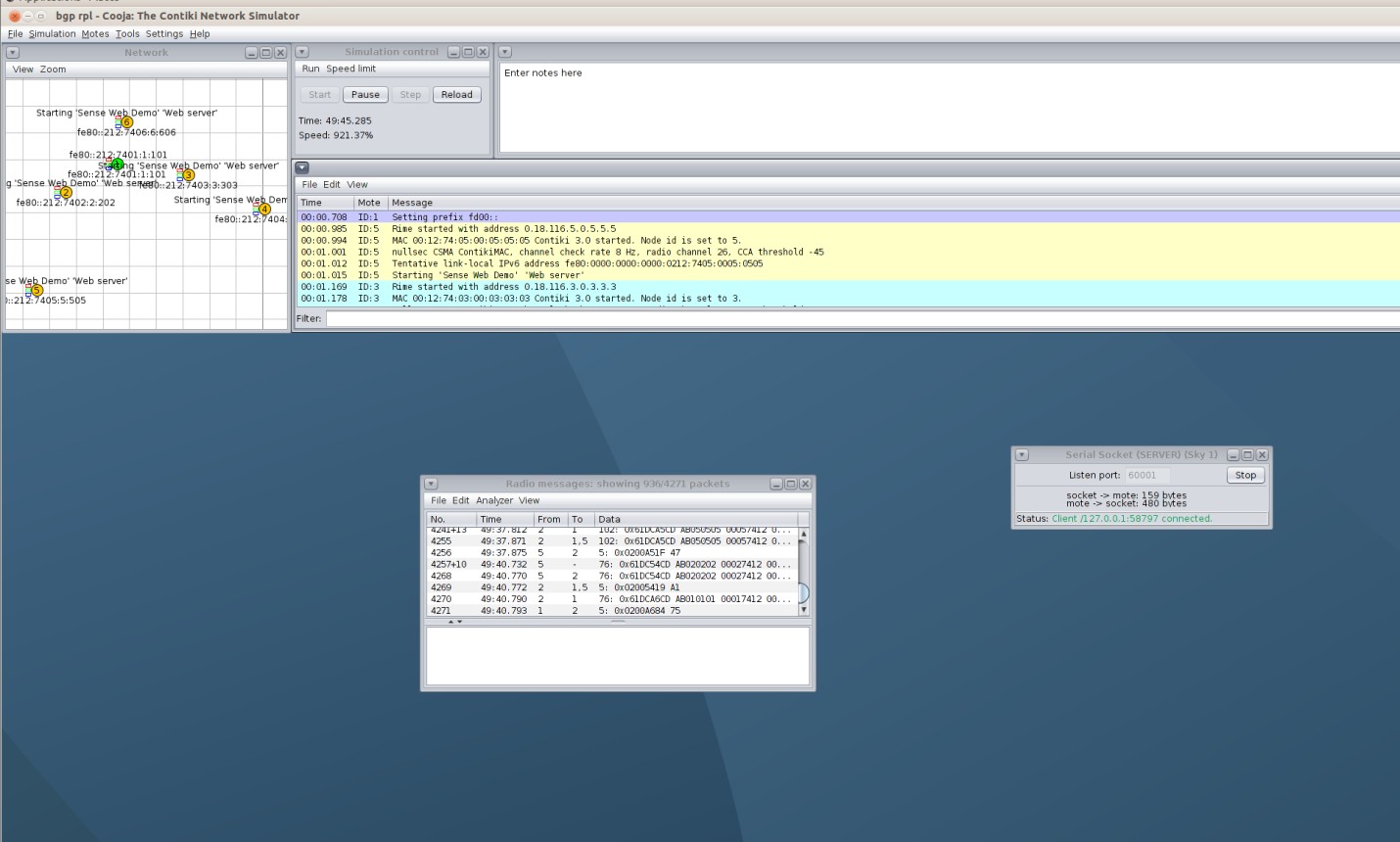
1. Now using the mote tools on mote 1(server) we will start the server socket on a port for it to listen. Also we will open the radio message tools to monitor traffic.



1. After starting the listening port we have to go to the example directory of router and type the command make connect router command to start the connection.



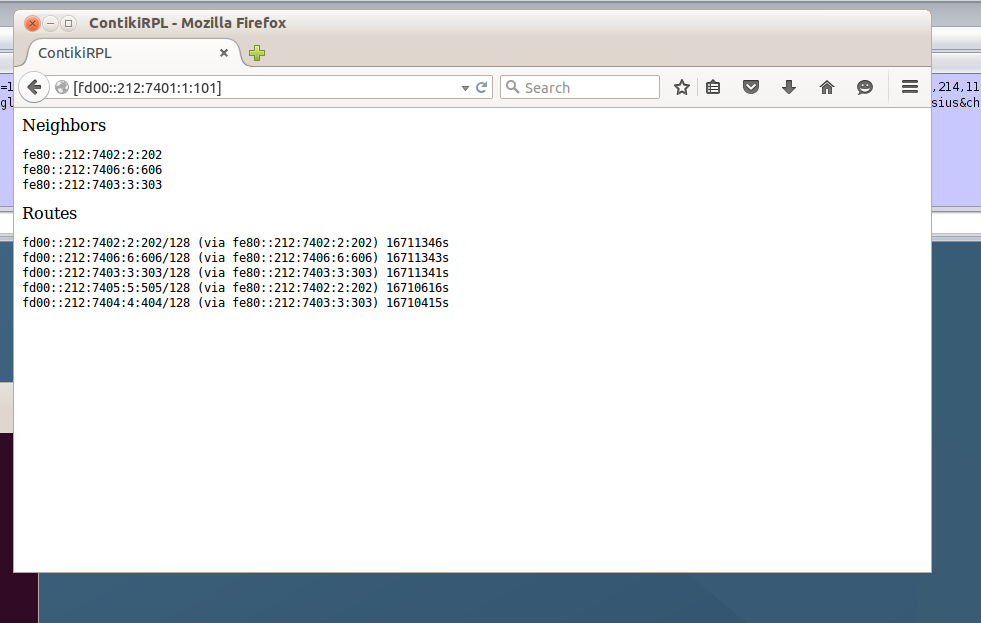
1. Now we will begin the simulation and the traffic will begin instantly and we can also observe all the motes will automatically have assigned ip addresses.



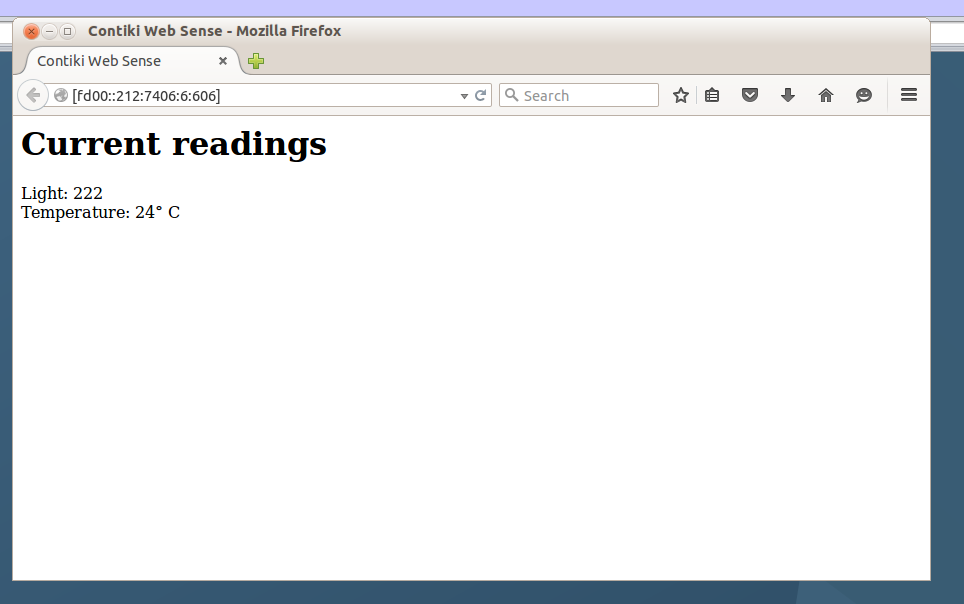
1. Also when we see the termial where we connected the router we would have the ip address of the border router.



1. Also using the various ip address of the border router we can get info when run in the browser. It works hence router is up and running and also displays all the routes

available to other motes and the motes it is directly connected to.

1. Also when we check the nearby sensor mote we can get to see the light and temperature readings.



## CONCLUSION

In this practical we have seen how we can implement BGP and RPL protocol in Cooja.

## 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.

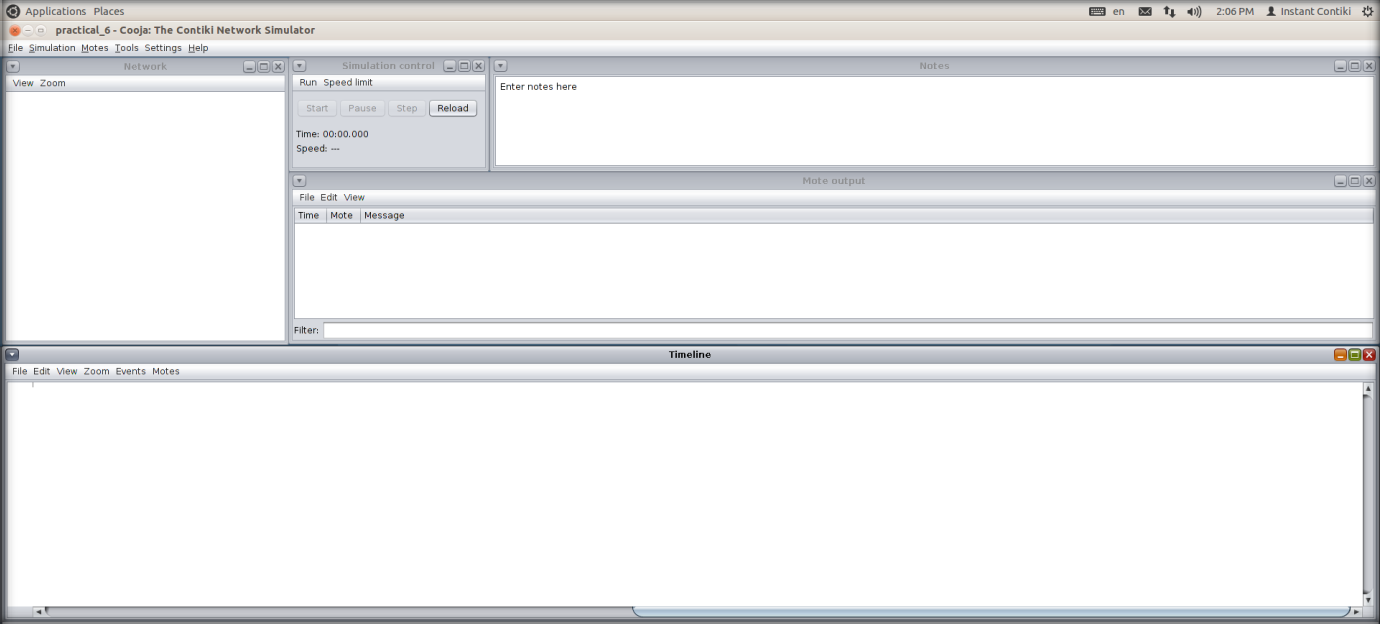
UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed 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.

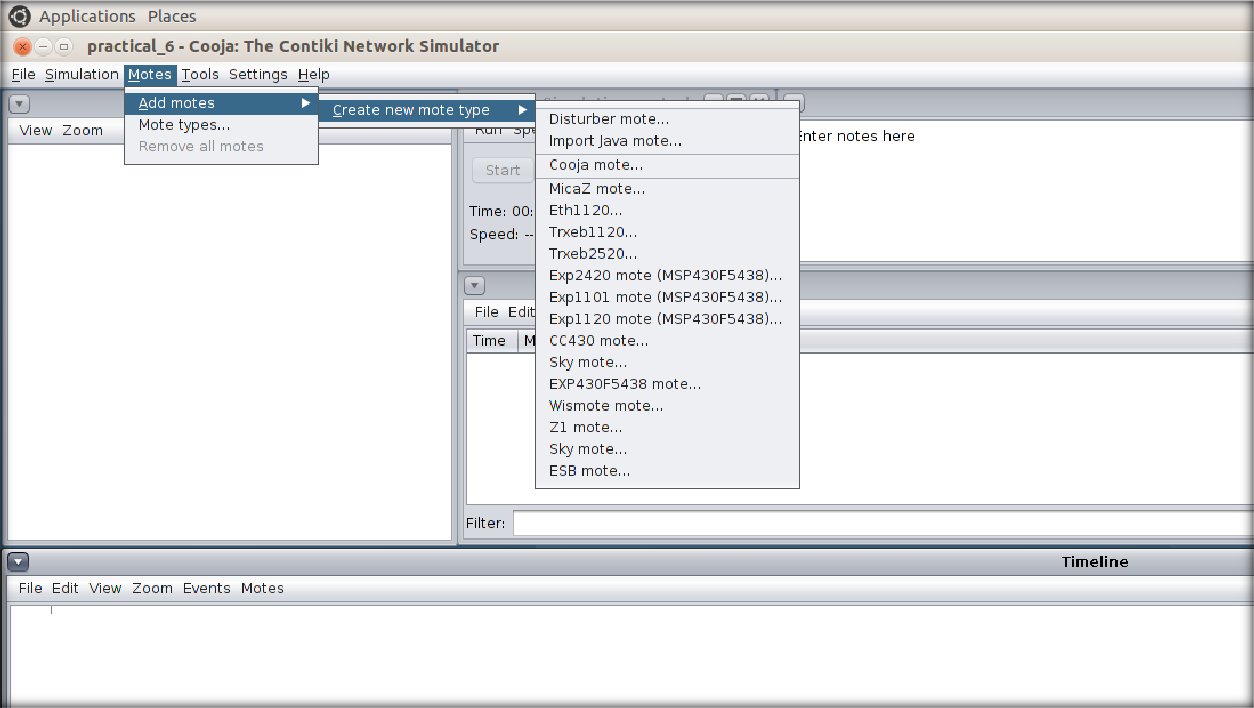
## PRACTICAL:

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.

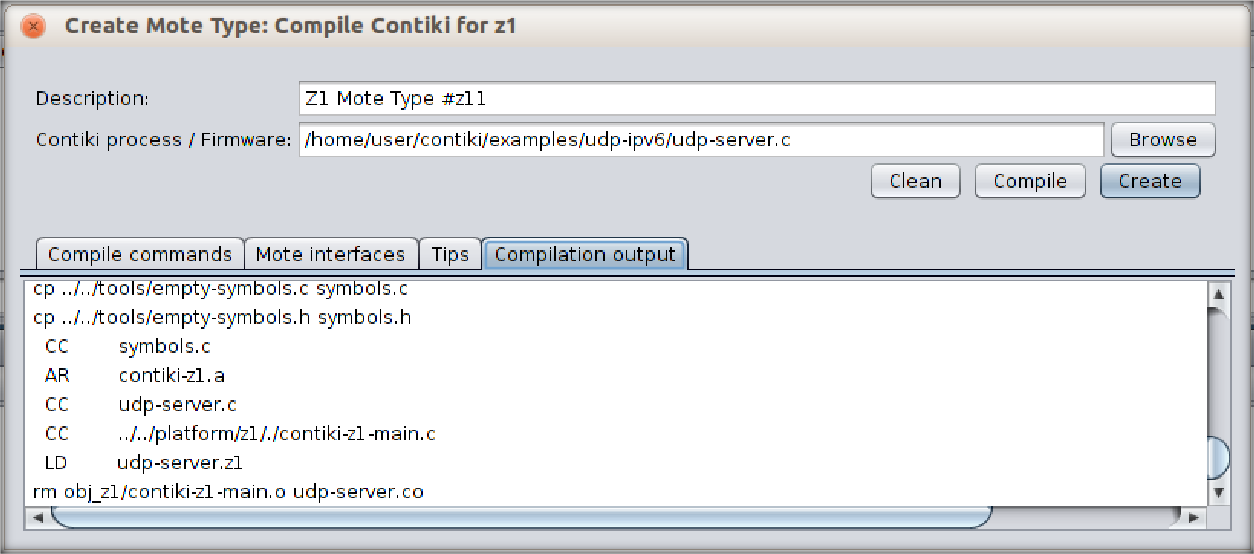
First of all, we will open Cooja simulator and create new simulator.



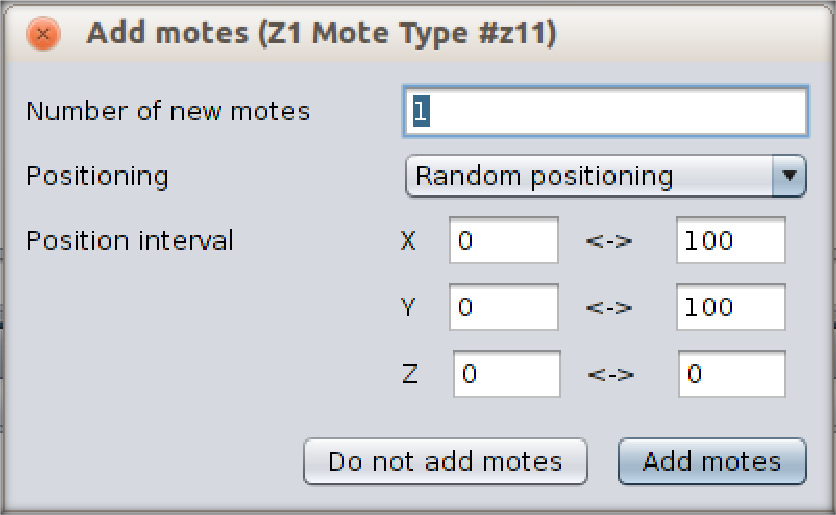
Then, we will first add the router mote, so we will go to Motes > Create new mote type >Z1 mote.



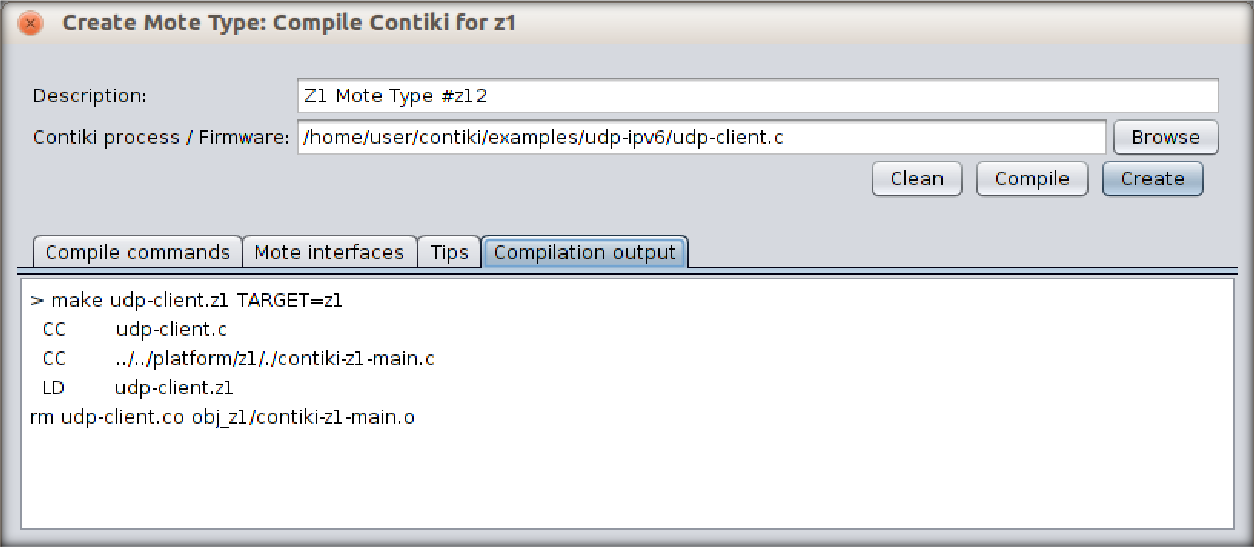
We will browse through example folder to find udp-ipv6 >udp-server.c. We will compile it and press create.



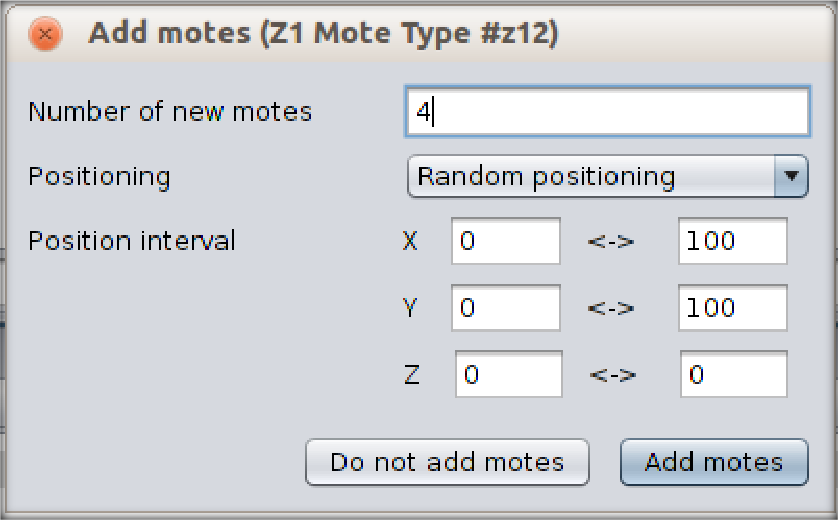
We will need only 1 router so number of new motes will be 1 only.



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.

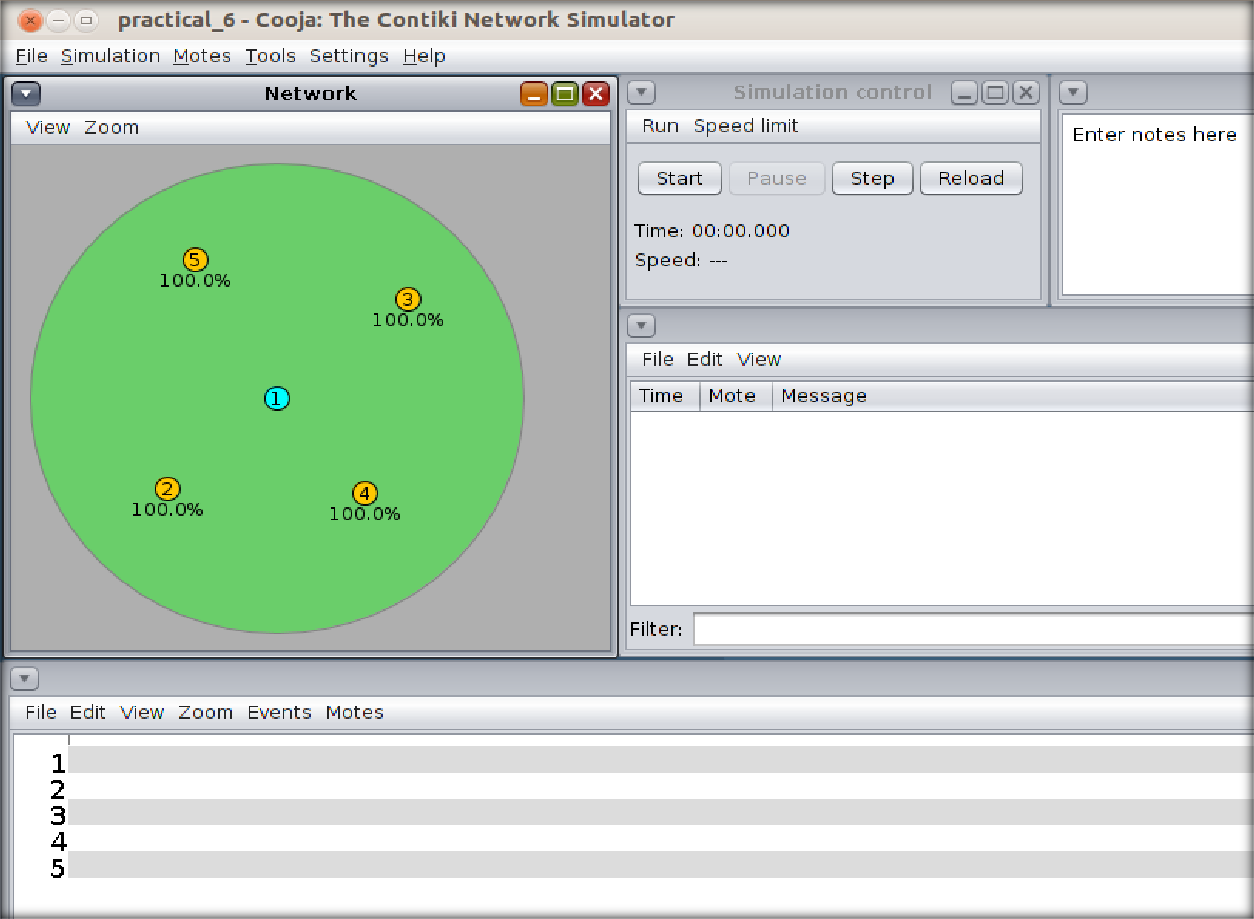


We will need more receivers for better visualization so we will add 4 motes.

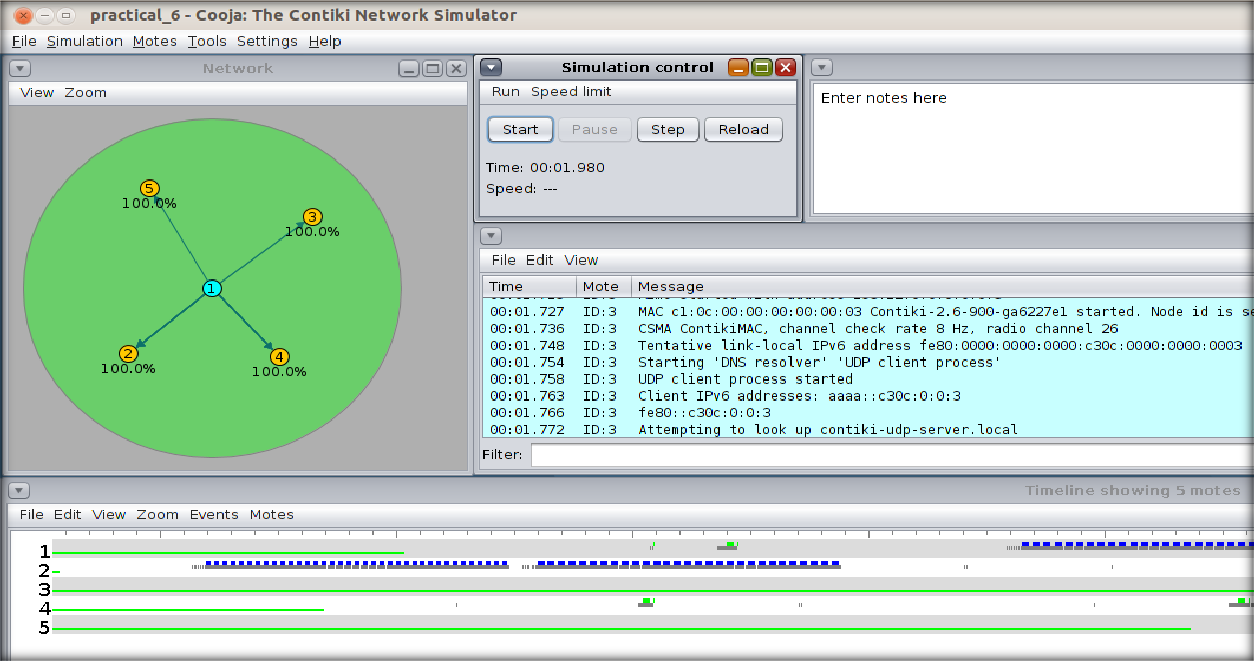


When we will add motes, we will be able to see a screen like this. By clicking on the mote, you will be able to see its range.

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.

## THOERY:

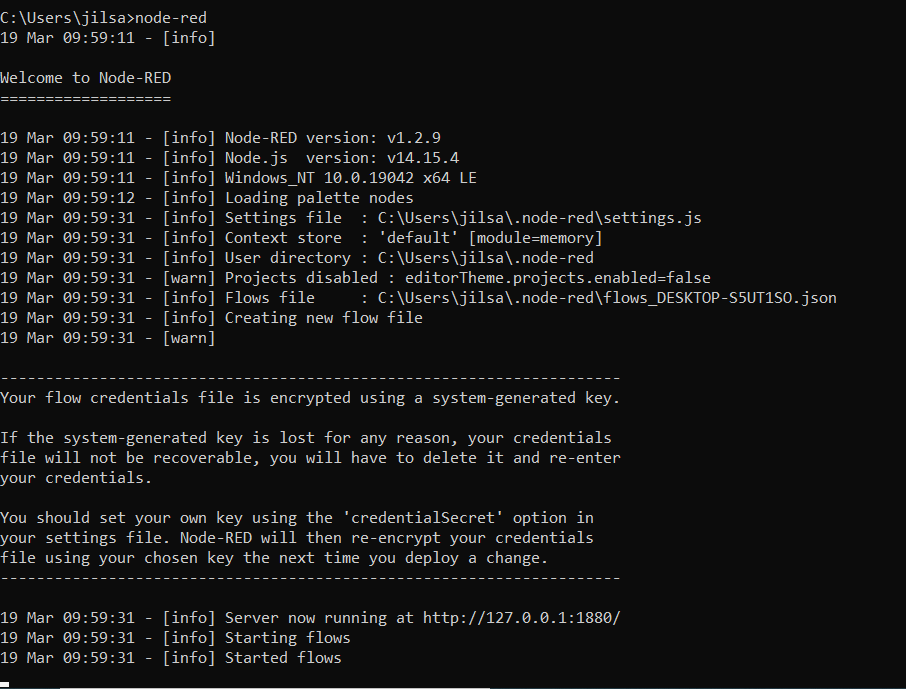
* Node Red:
  + Node-RED is a programming tool for wiring together hardware devices, APIs and online services in new and interesting ways.
  + It provides a browser-based editor that makes it easy to wire together flows using the wide range of nodes in the palette that can be deployed to its runtime 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.
    - Security Enabled
      * MQTT makes it easy to encrypt messages using TLS and authenticate clients using modern authentication protocols, such as OAuth.

## PRACTICAL:

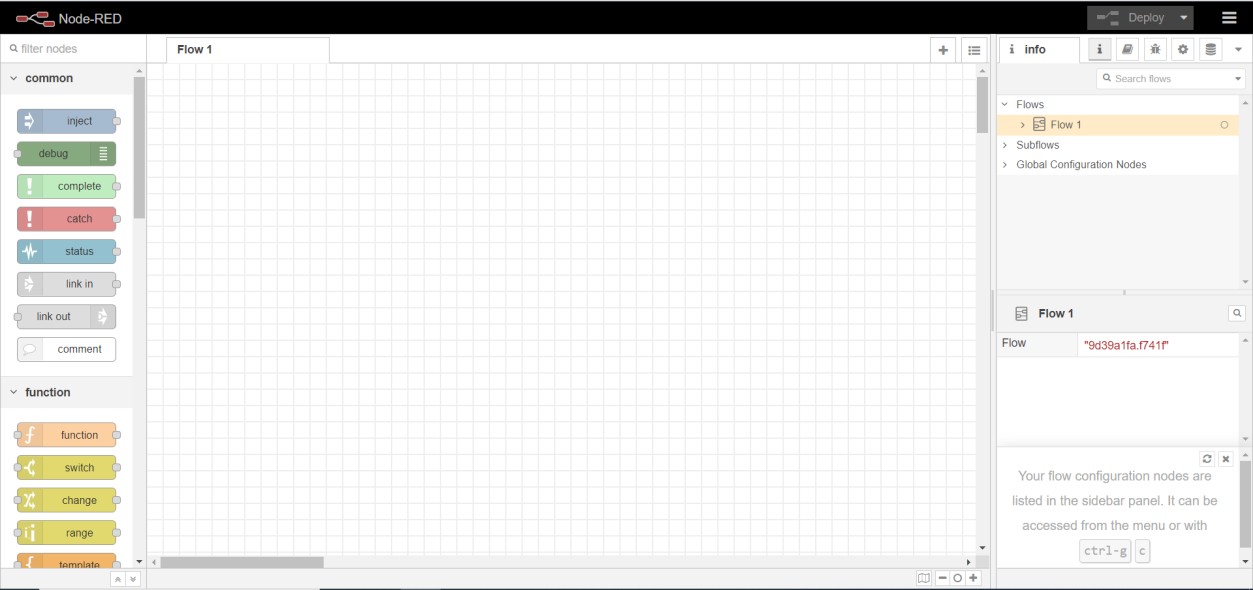
We can install node red on windows by following command.

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

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

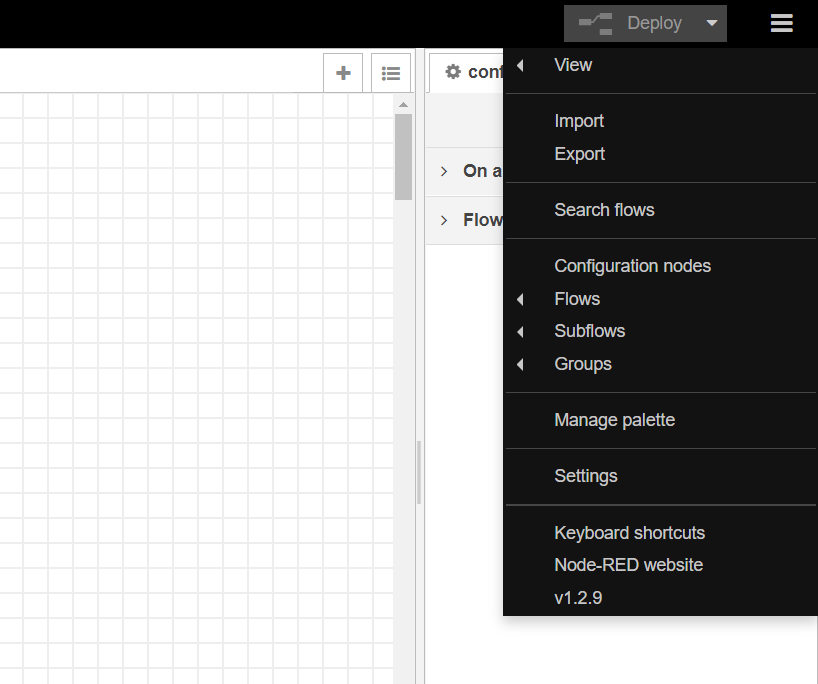


It will give us an IP address to use web-based node-red.

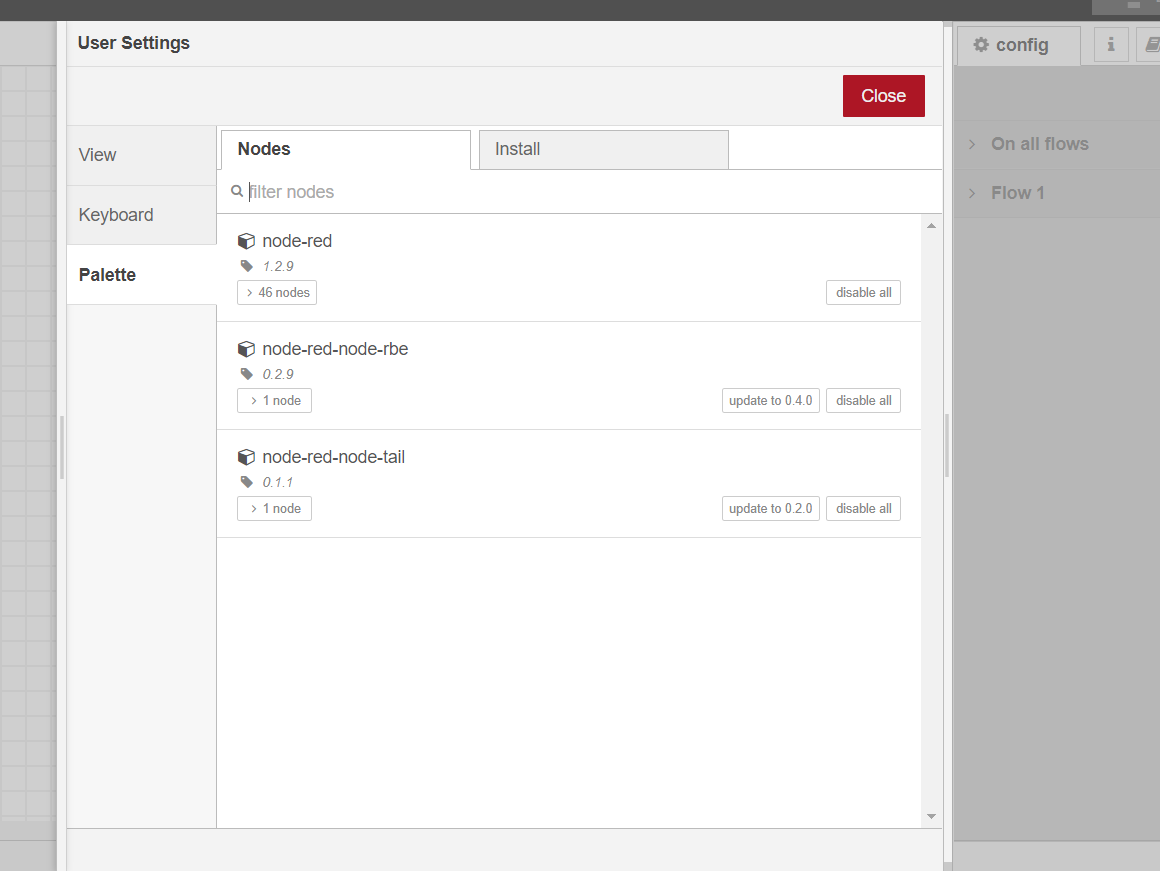


First, we will install broker for MQTT protocol.

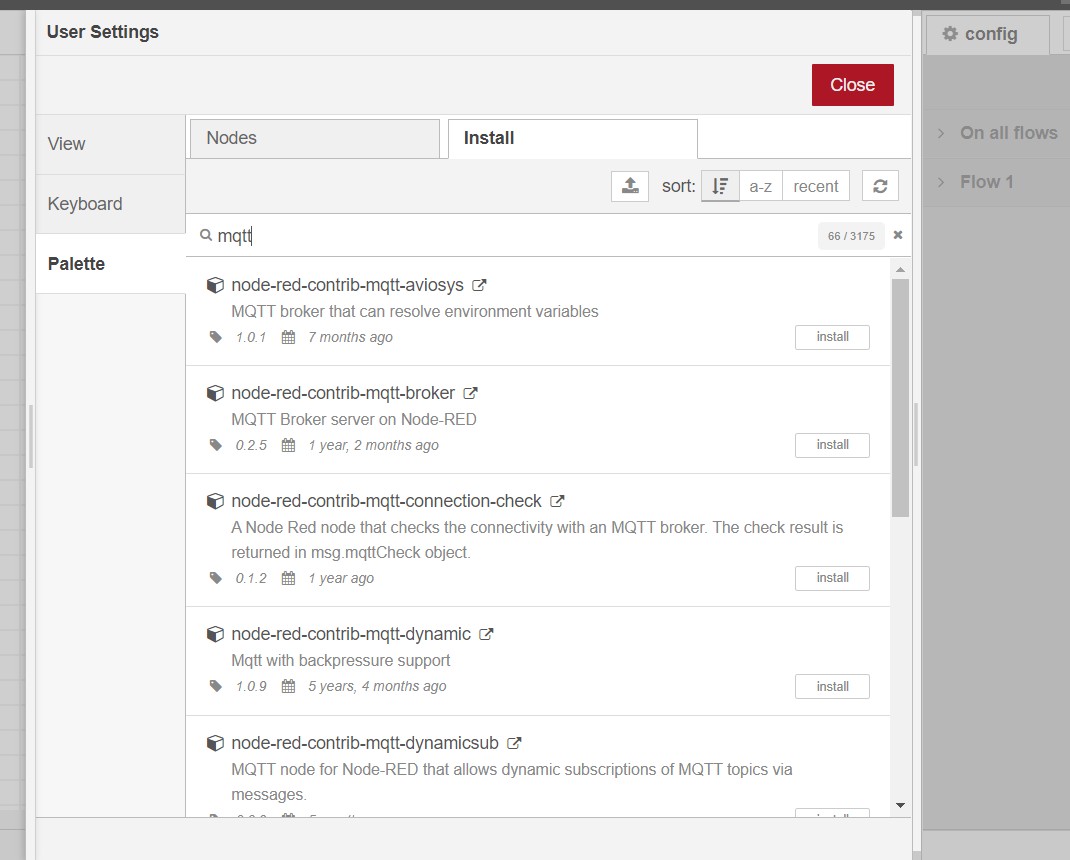
For that, we will go to menu situated at top right corner. We will see Manage Palette option there.



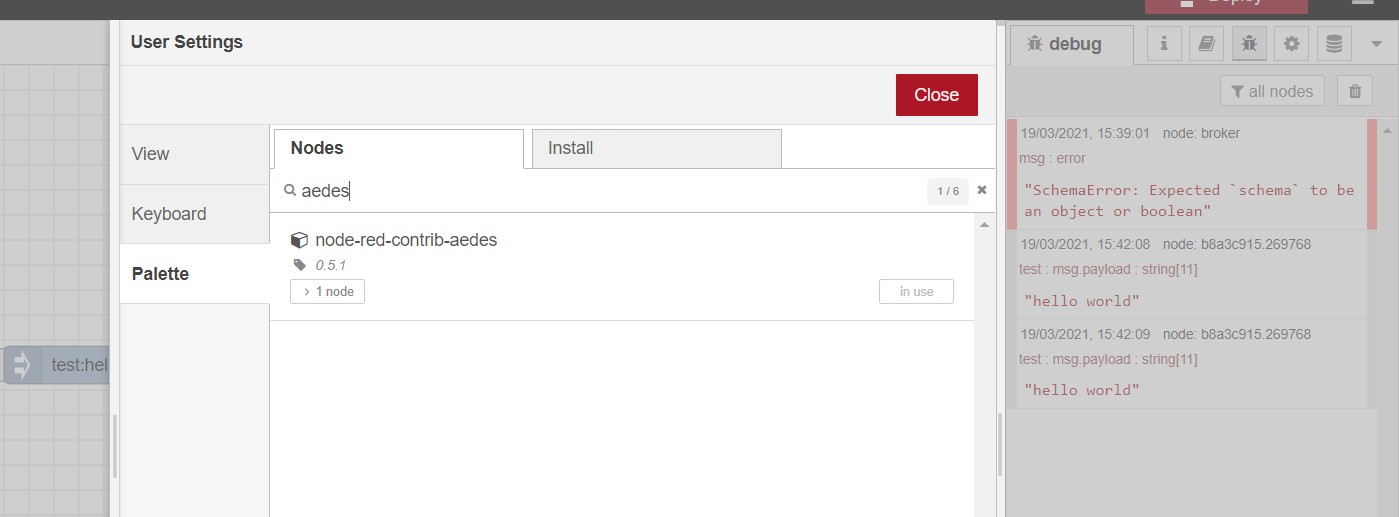
By clicking on it, User settings will be opened.

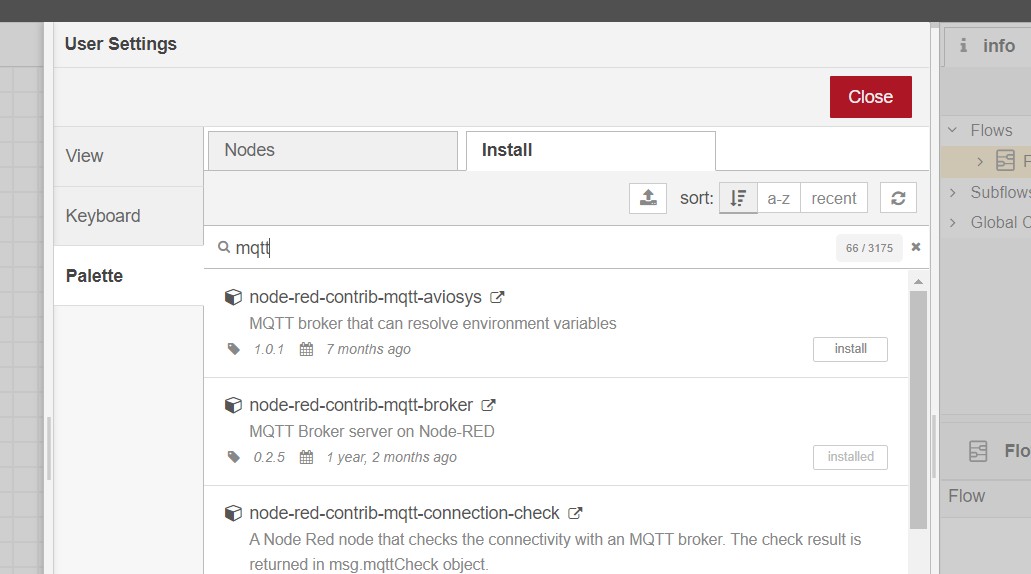


We go to Install tab and search MQTT.



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

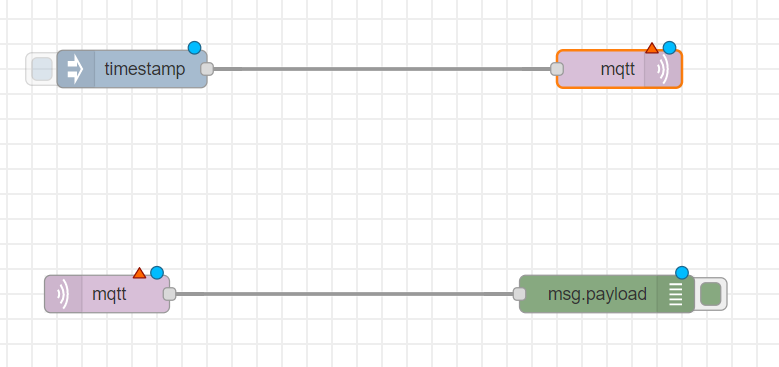




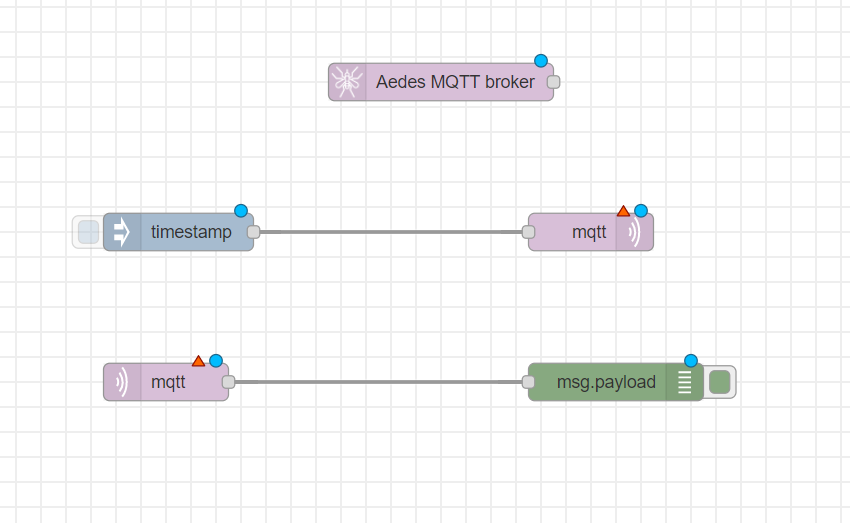
Then we will add 4 nodes.

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

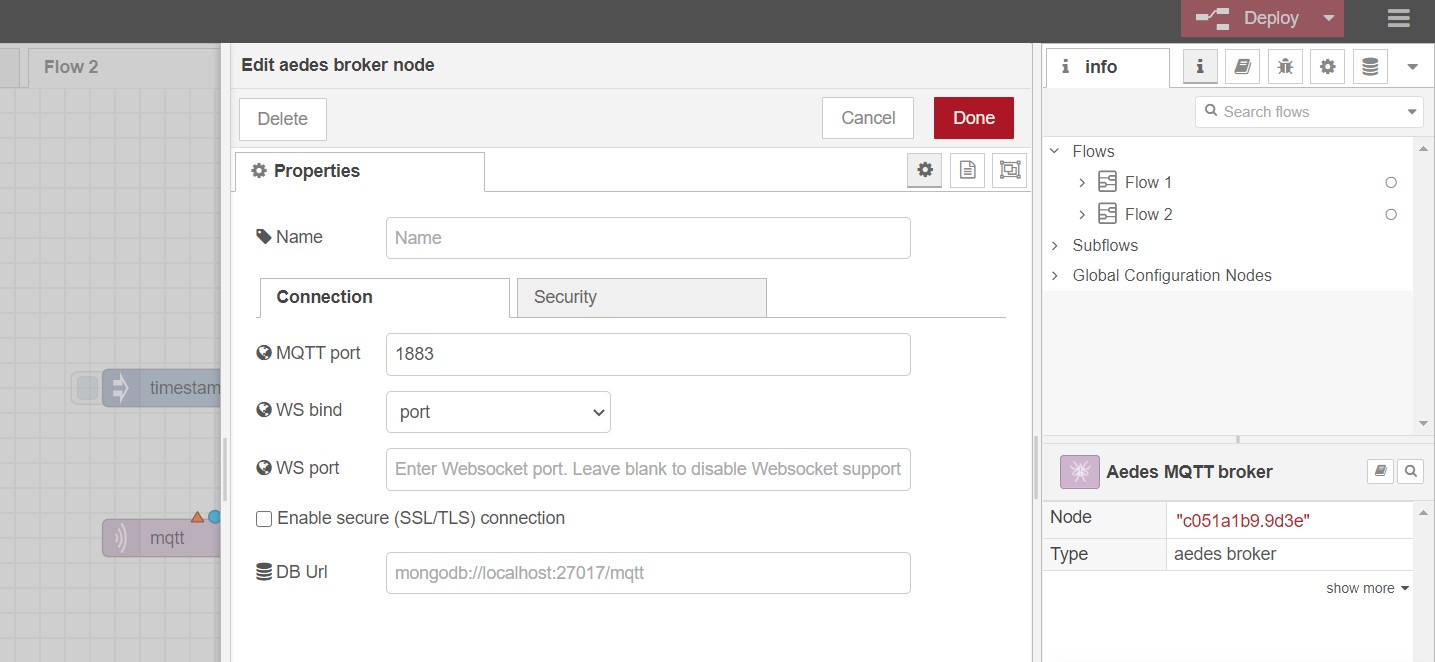
We will connect timestamp node to MQTT out and debug node to MQTT in.



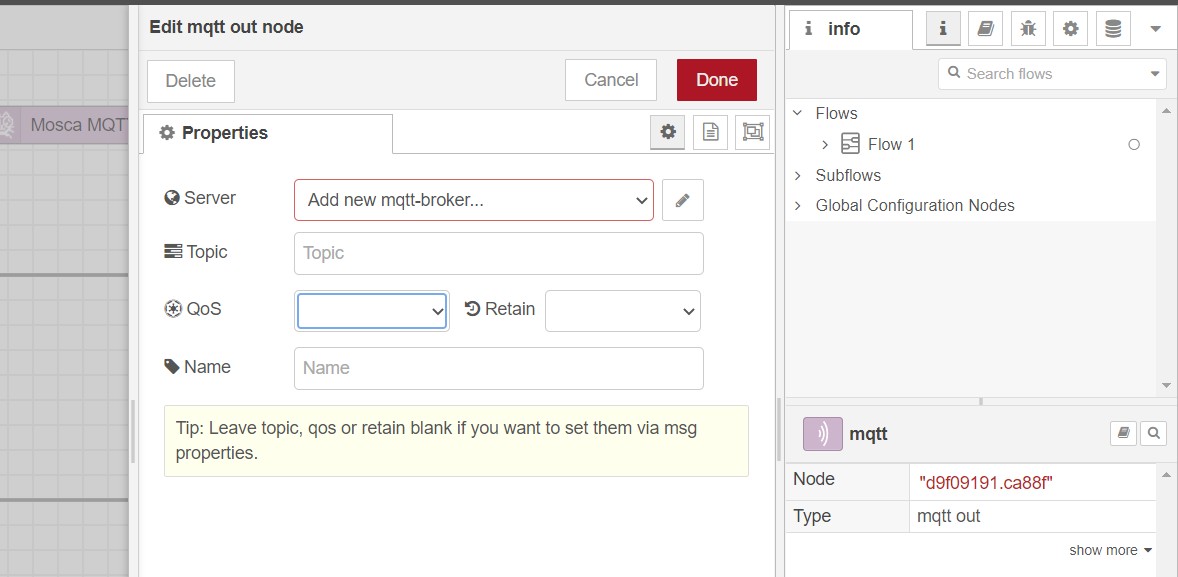
Then we will add MOSCA MQTT broker or AEDES MQTT broker.



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

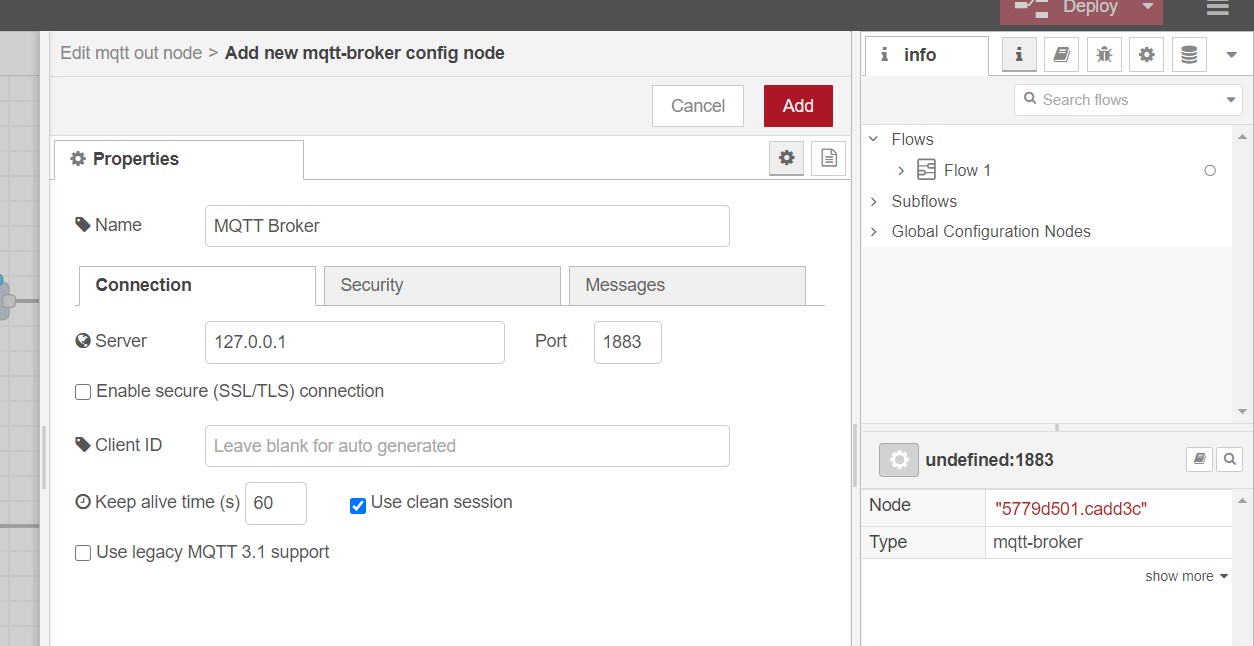


We will first configure MQTT out node.

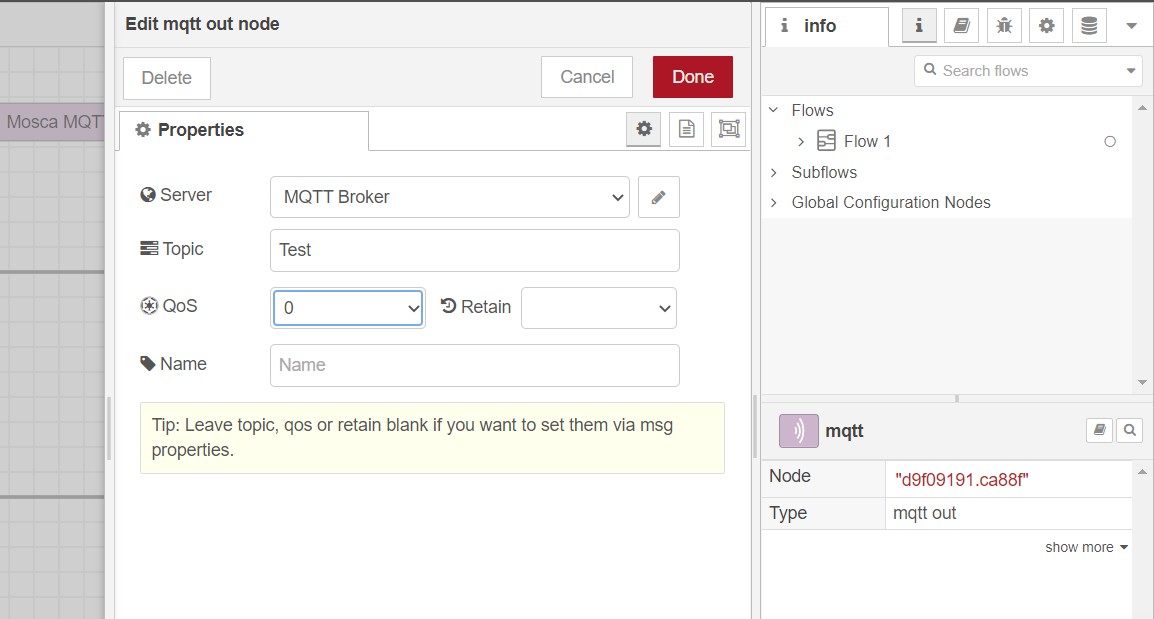


We will add new mttt-broker. So we click on the button beside it. We will add name and server IP address.

We can add additional settings like security too if we want.



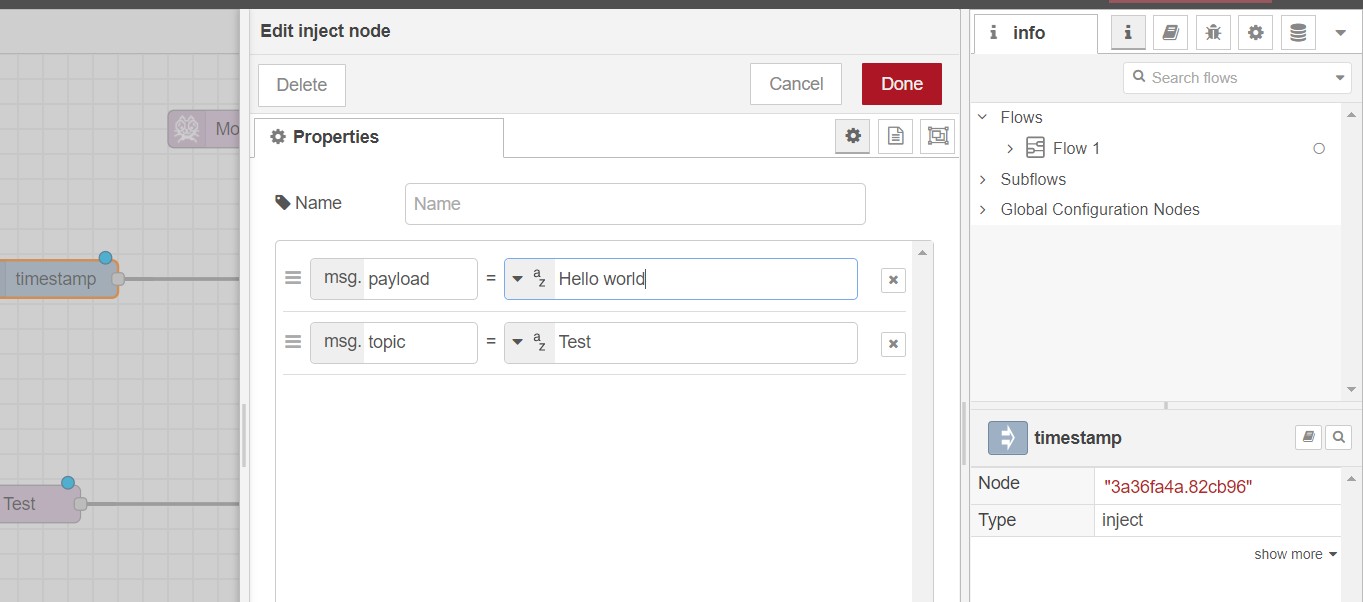
When we click add, the server will be created and we will fill a couple of fields there like topic and QoS.



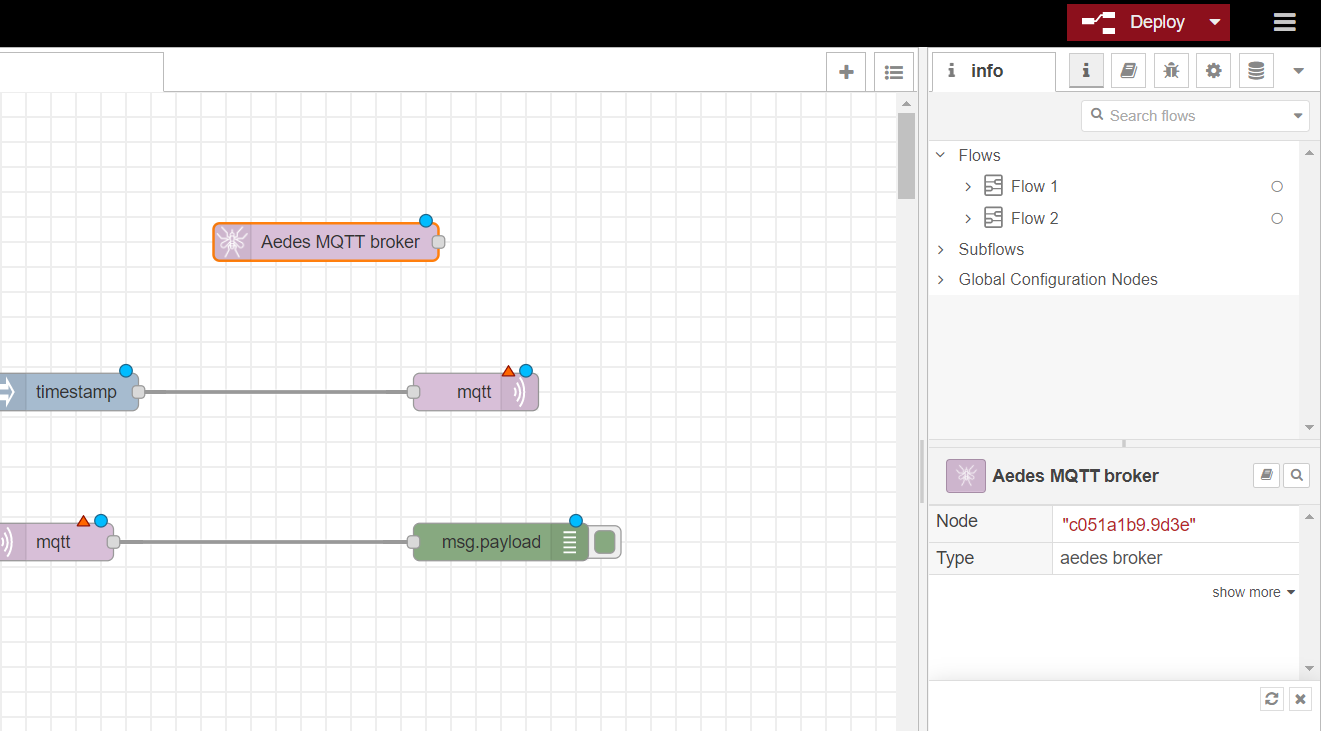
We will provide same configuration for MQTT in node but we don’t need to create MQTT Broker again.



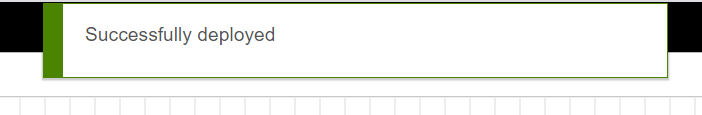
We will give input string in timestamp node. We change the timestamp to string of “Hello world” and give it the topic name same as MQTT nodes.



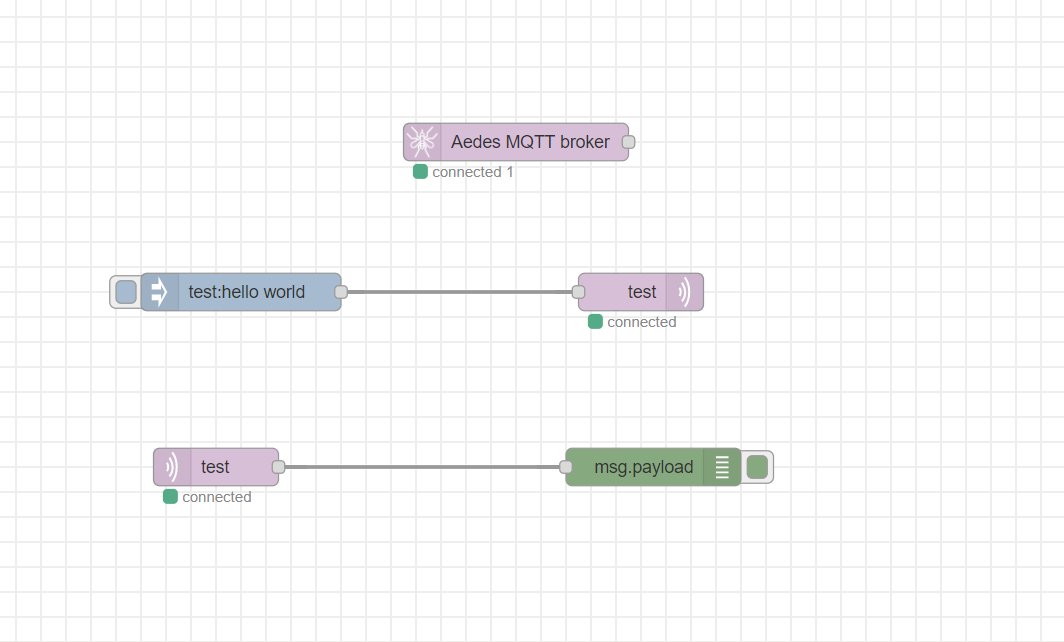
Now we can deploy our model by “Deploy” button on top right corner.



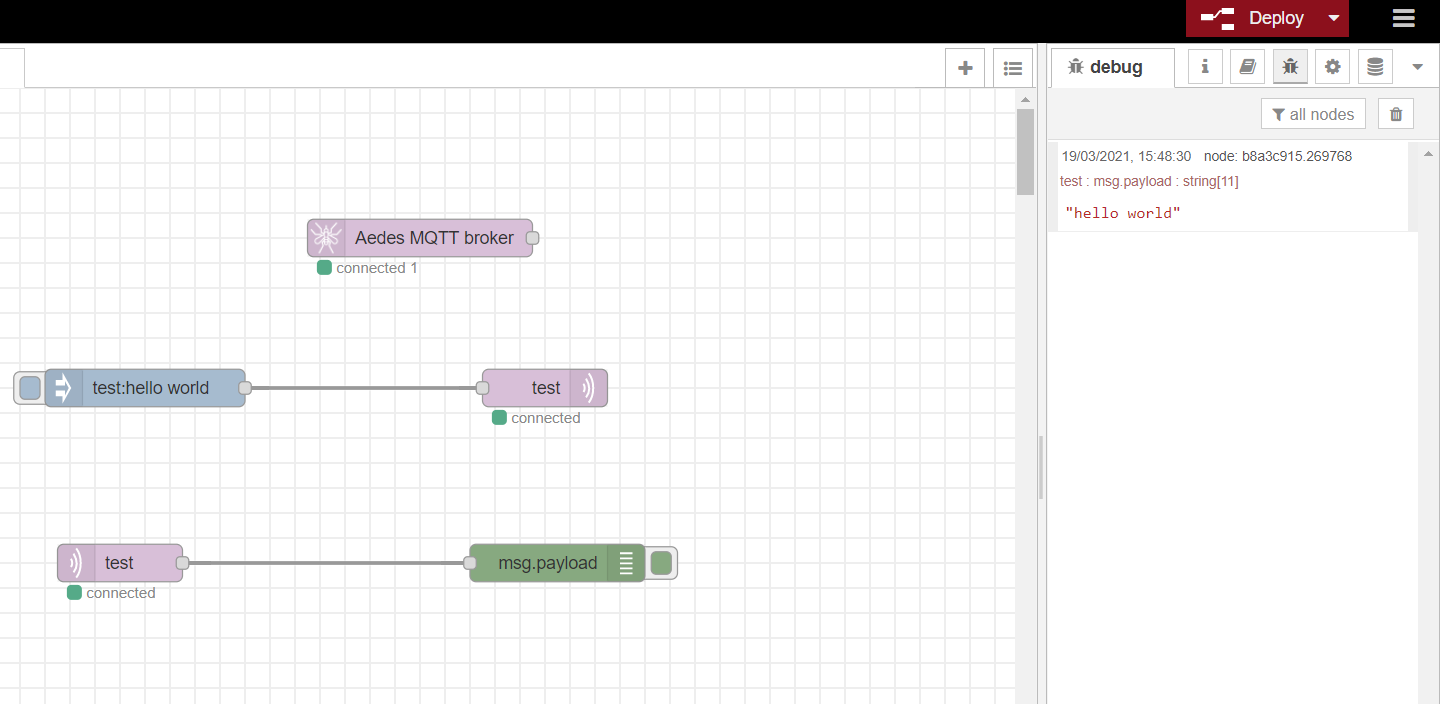
We can see “Successfully deployed” message.



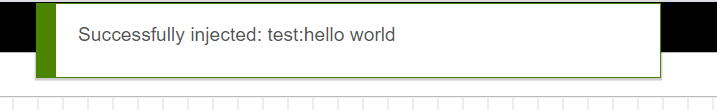
And in a moment, we will be able to see connected status if there is no error.



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.

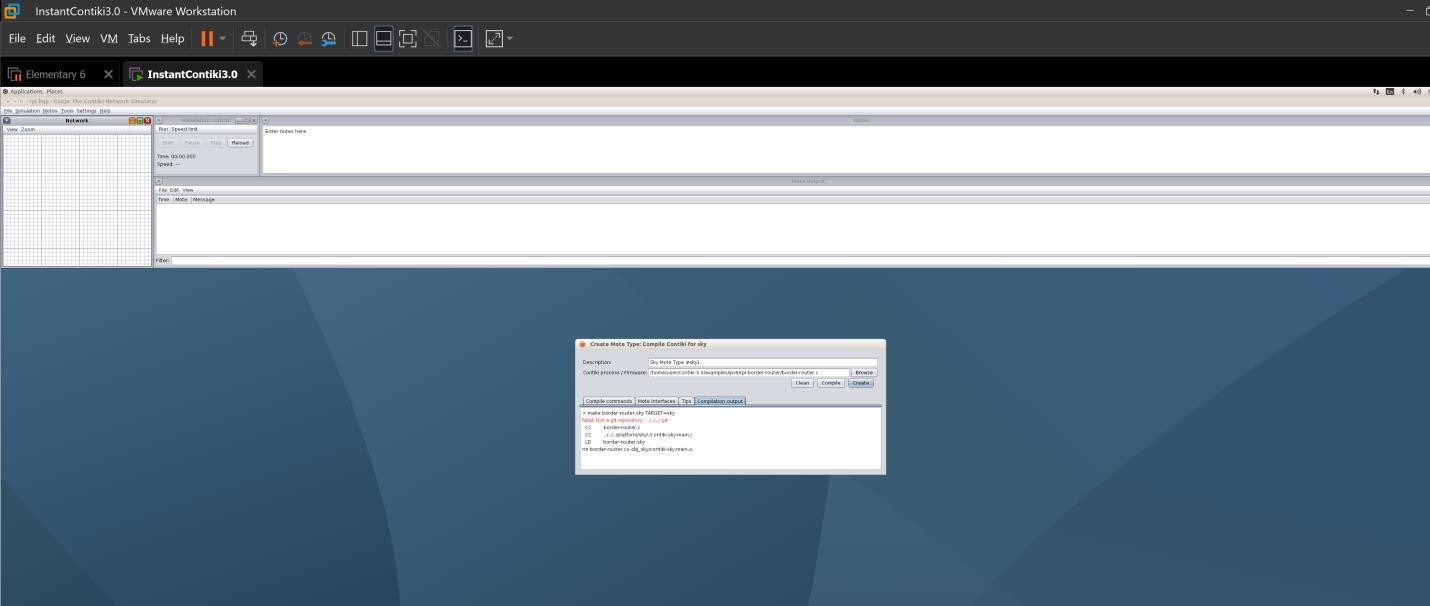
# PRACTICAL 7

## AIM:

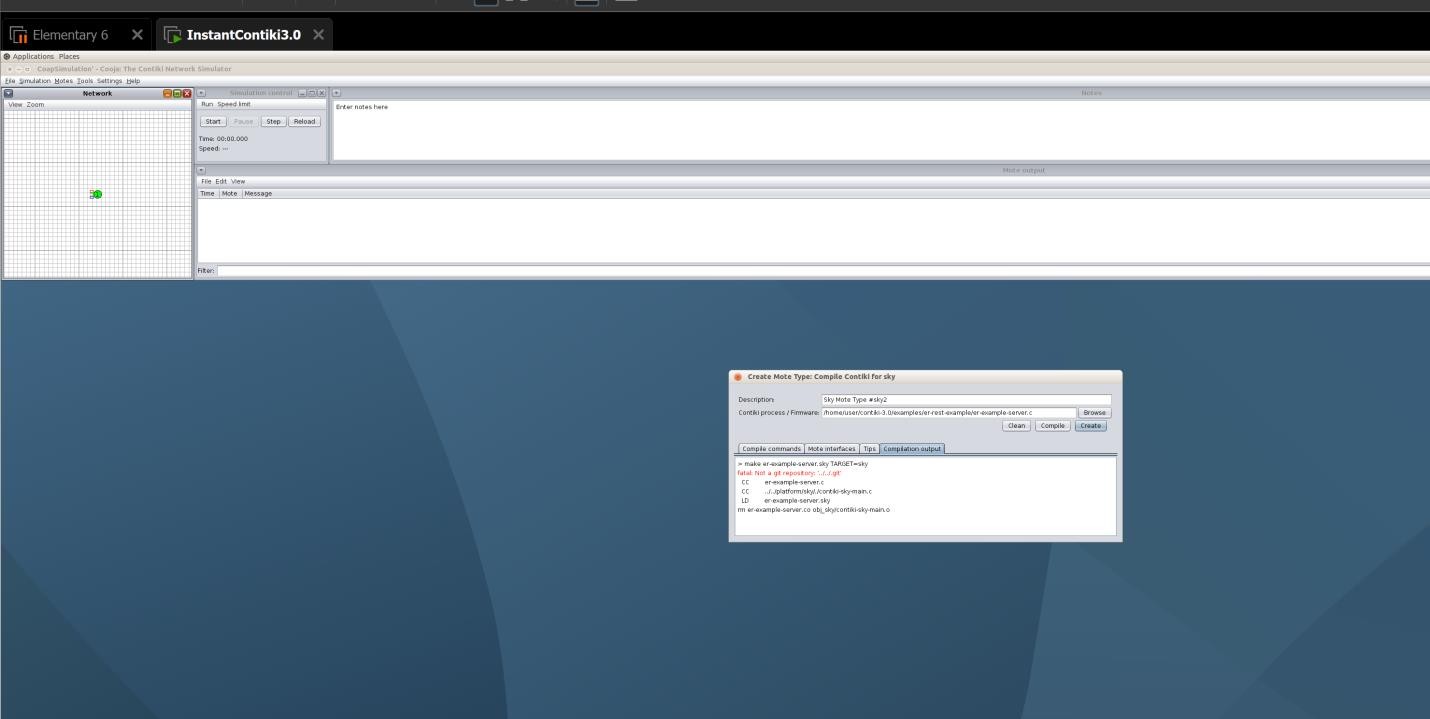
Simulate CoAP protocol in Contiki OS.

## Steps :

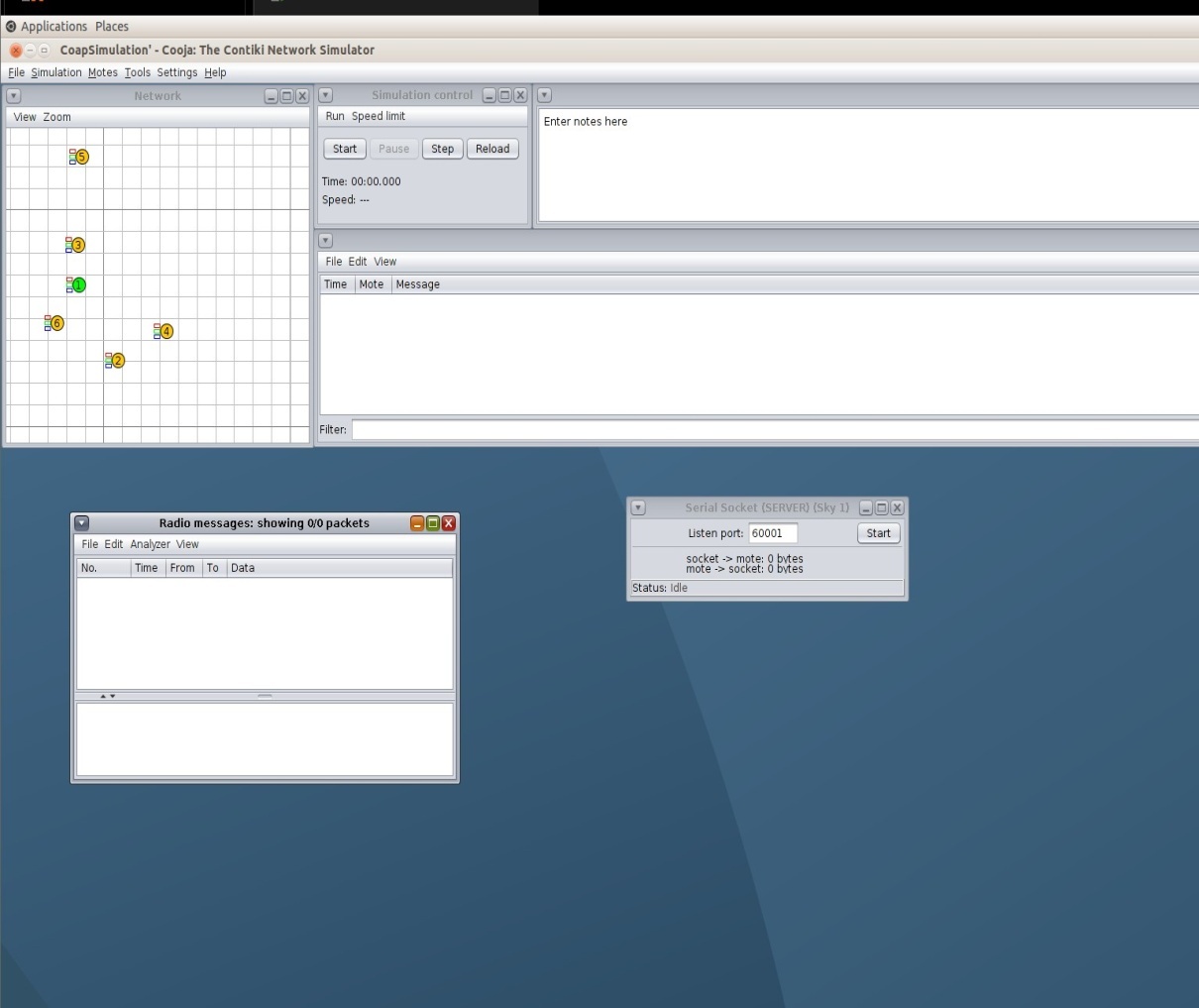
1. After starting cooja we will create a new simulation and to begin we will create a new Sky Mote group that will be the server mote and we will compile this using the rpl- border router.c file.



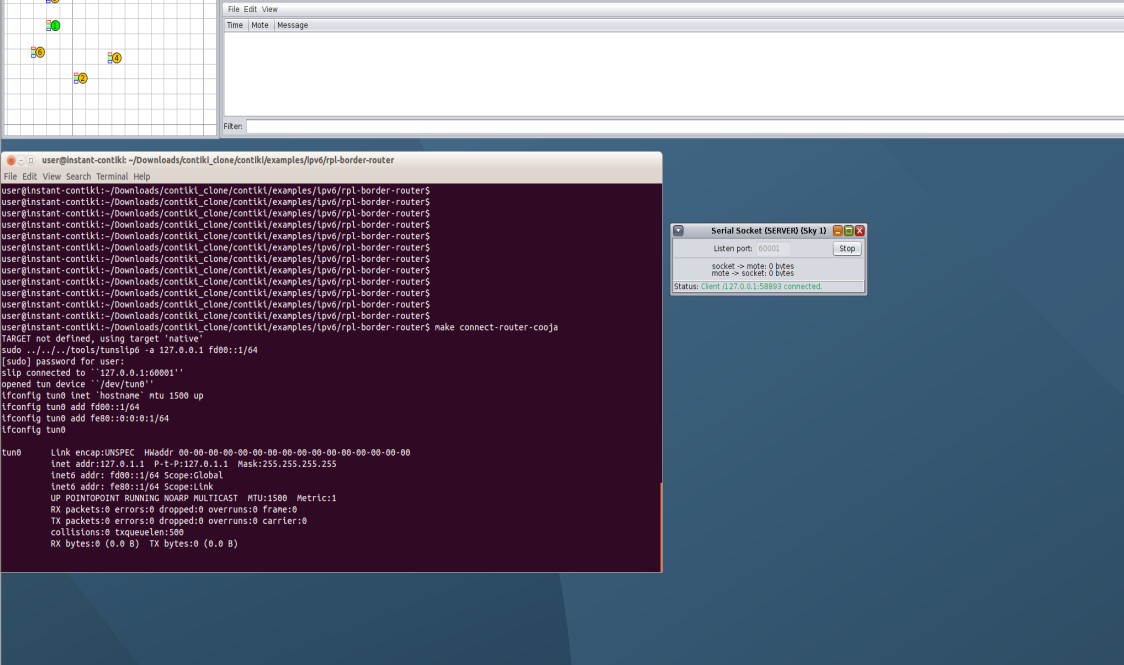
1. We will create another sky mote and this one would be compiled using the er rest example server.c file and we will add 5 motes of this type.



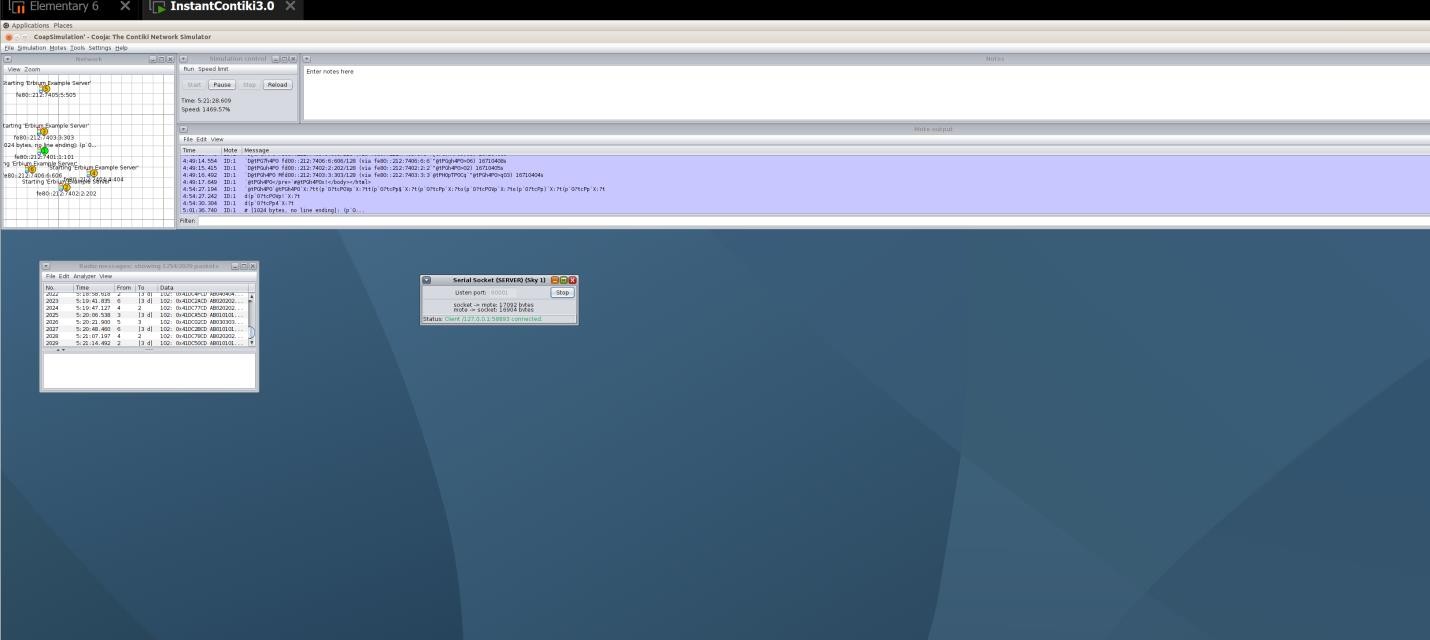
1. Now using the mote tools on mote 1(server) we will start the server socket on a port for it to listen. Also we will open the radio message tools to monitor traffic.



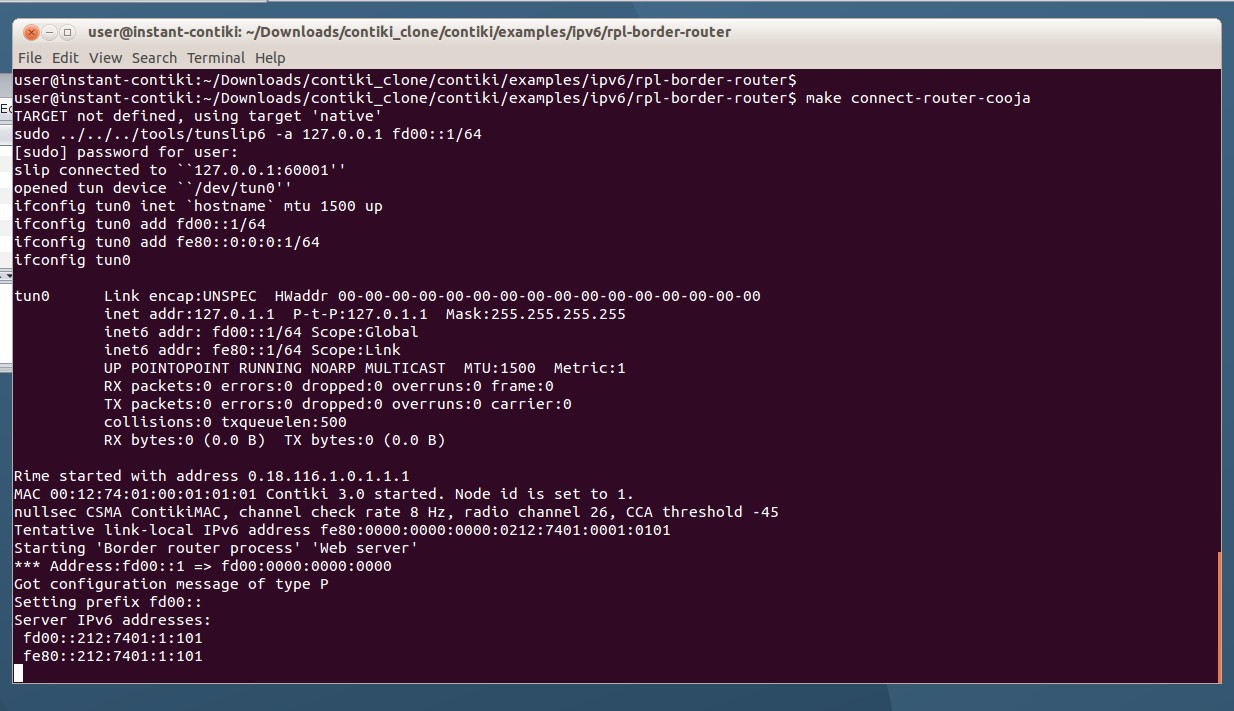
1. After starting the listening port we have to go to the example directory of router and type the command make connect router command to start the connection.



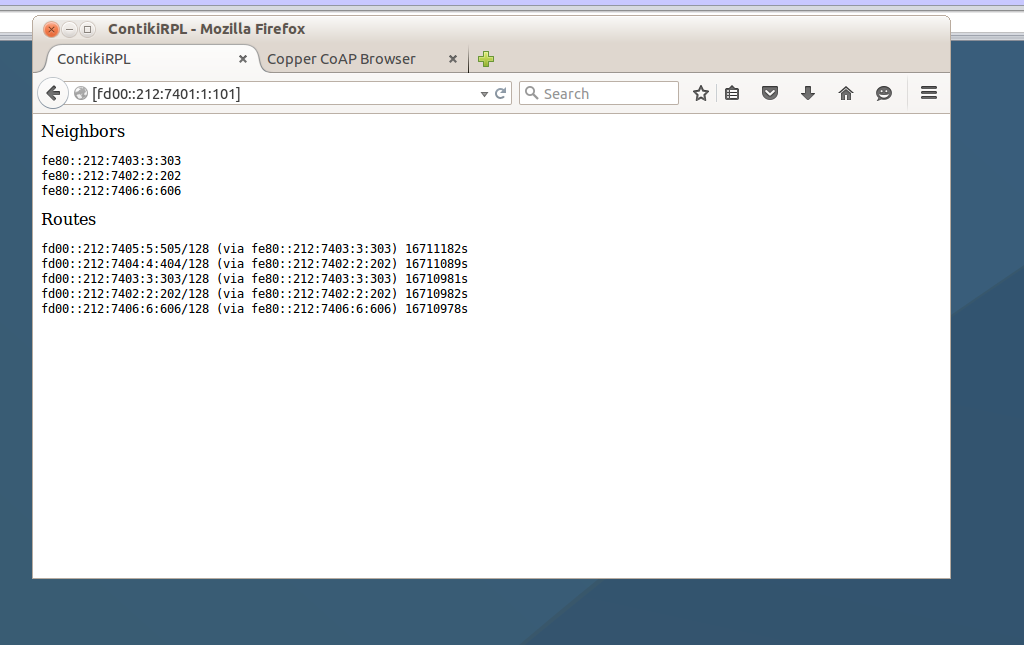
1. Now we will begin the simulation and the traffic will begin instantly and we can also observe all the motes will automatically have assigned ip addresses.



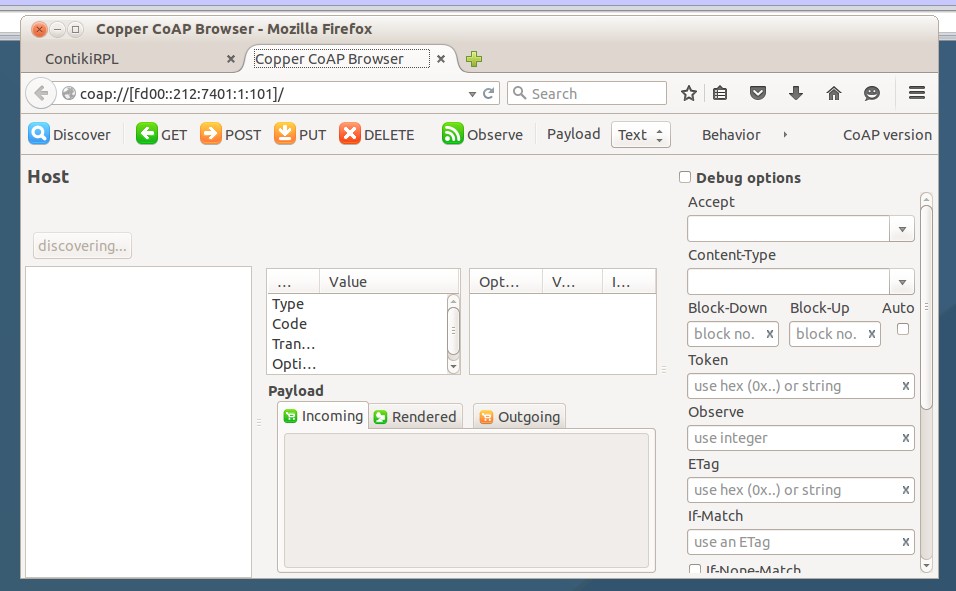
1. Also when we see the termial where we connected the router we would have the ip address of the border router.



1. Also using the various ip address of the border router we can get info when run in the browser. It works hence router is up and running and also displays all the routes

available to other motes and the motes it is directly connected to.

1. Also when we check the whole network and discover services and perform requests using Copper(Cu) which is a CoAp simulator agent however due to browser policies is not allowed and any longer supported hence a lot of errors and cannot get data.



## CONCLUSION

In this practical we have seen how we can Simulate CoAP protocol in Contiki OS.

## 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.

## THOERY:

* 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:
    - A range of services that enable you to build and extend web and mobile apps fast
    - Processing power for you to deliver app changes continuously
    - Fit-for-purpose programming models and services
    - Manageability of services and applications
    - Optimized and elastic workloads
    - 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:
    - Easily configure devices to send data to ThingSpeak using popular IoT protocols.
    - Visualize your sensor data in real-time.
    - Aggregate data on-demand from third-party sources.
    - Use the power of MATLAB to make sense of your IoT data.
    - Run your IoT analytics automatically based on schedules or events.
    - Prototype and build IoT systems without setting up servers or developing web software.
    - 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;

WiFiClientwifiClient;

PubSubClientclient(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 += "}}";

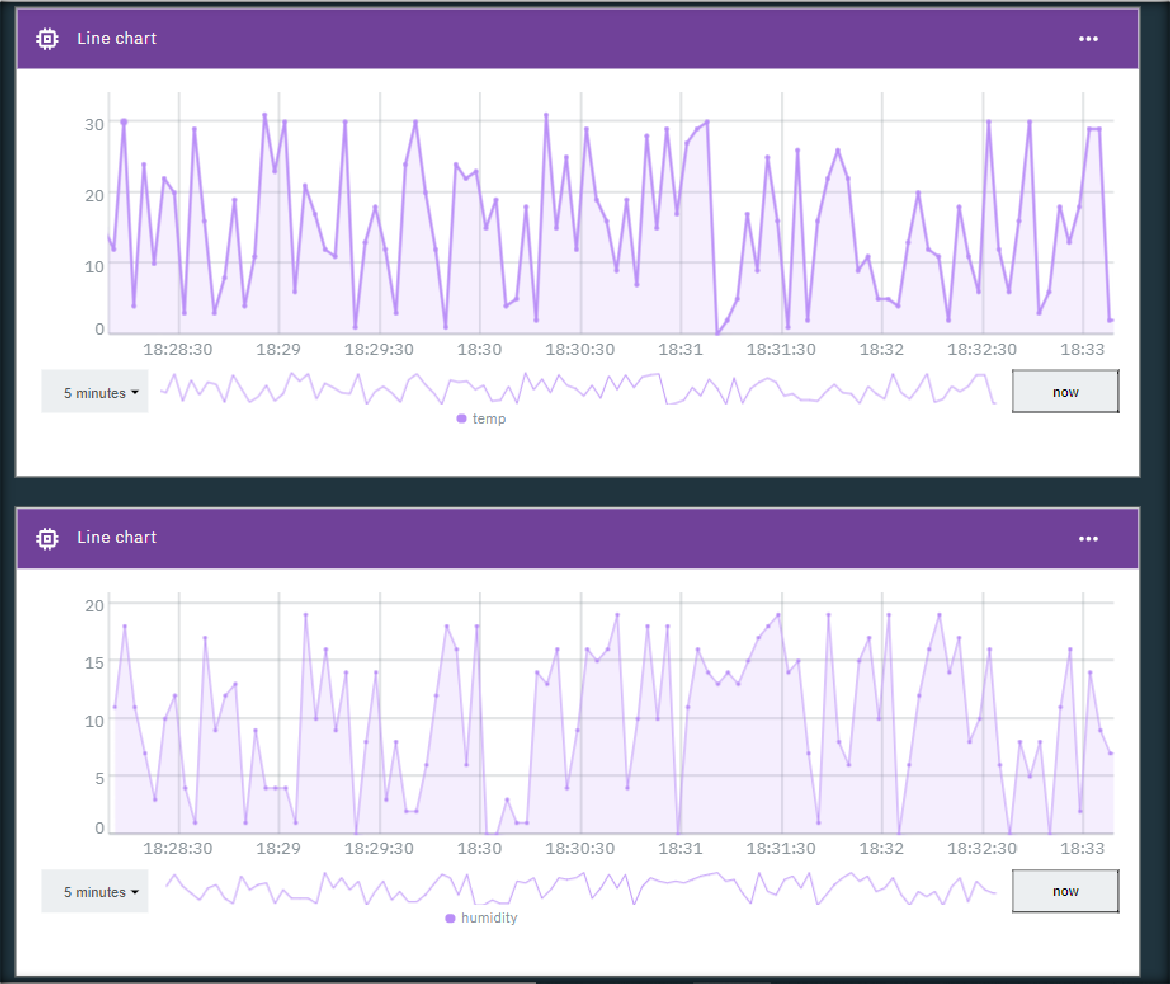
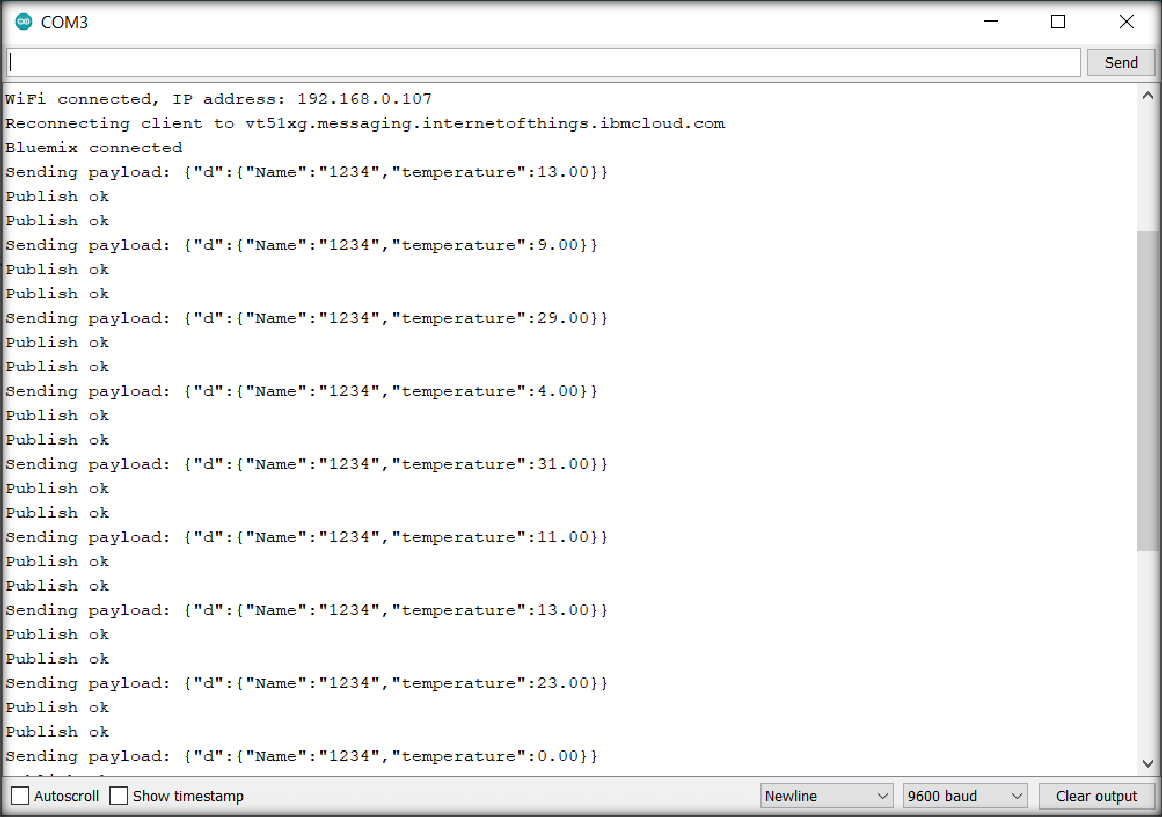
if (client.publish(pubTopic2, (char\*) payload1.c\_str())) { Serial.println("Publish ok");

} else { Serial.println("Publish failed");

}

}

}



*Thingspeak:*

#include <ESP8266WiFi.h>

#define SENSOR A0

const char\* ssid = "Chandaranas"; //WiFiSSID

const char\* password = "Cmv@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);

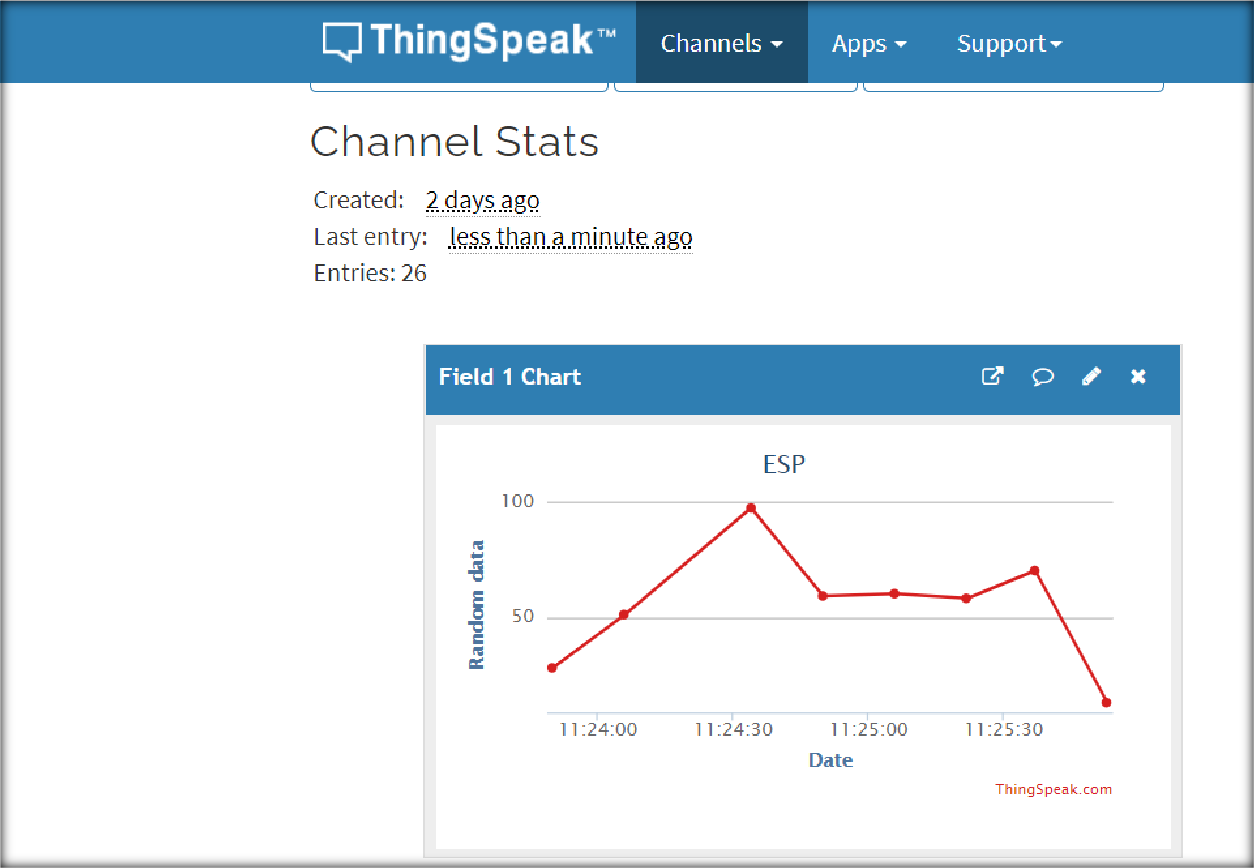
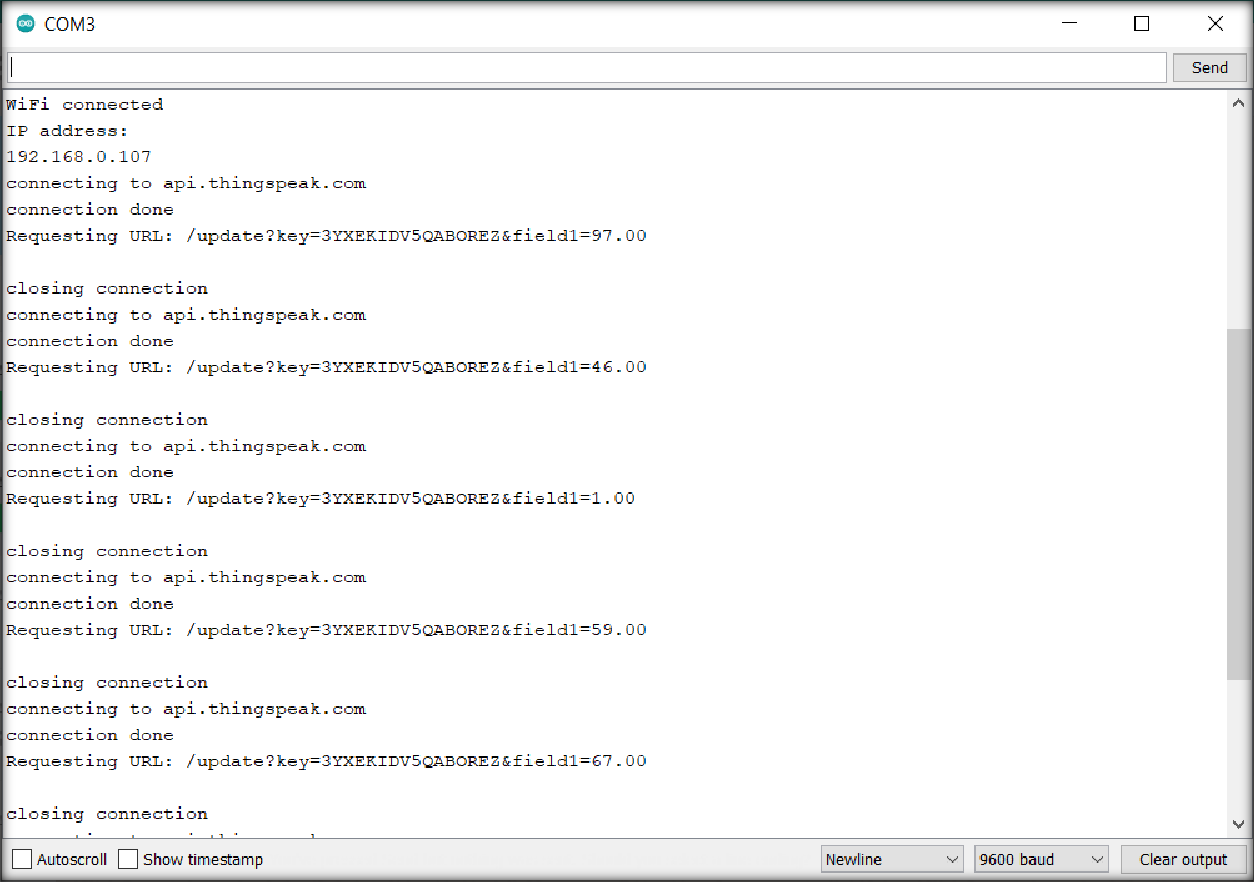
// Read all the lines of the reply from server and print them to Serial while (client.available()) {

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:**

AICTE Project Demo and hardware study of Waspmote, Libellium Gateway, Zigbee module 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.
  + Another defining feature of Zigbee is facilities for carrying out secure communications, protecting establishment and transport of cryptographic keys, ciphering frames, and controlling device.
  + 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:

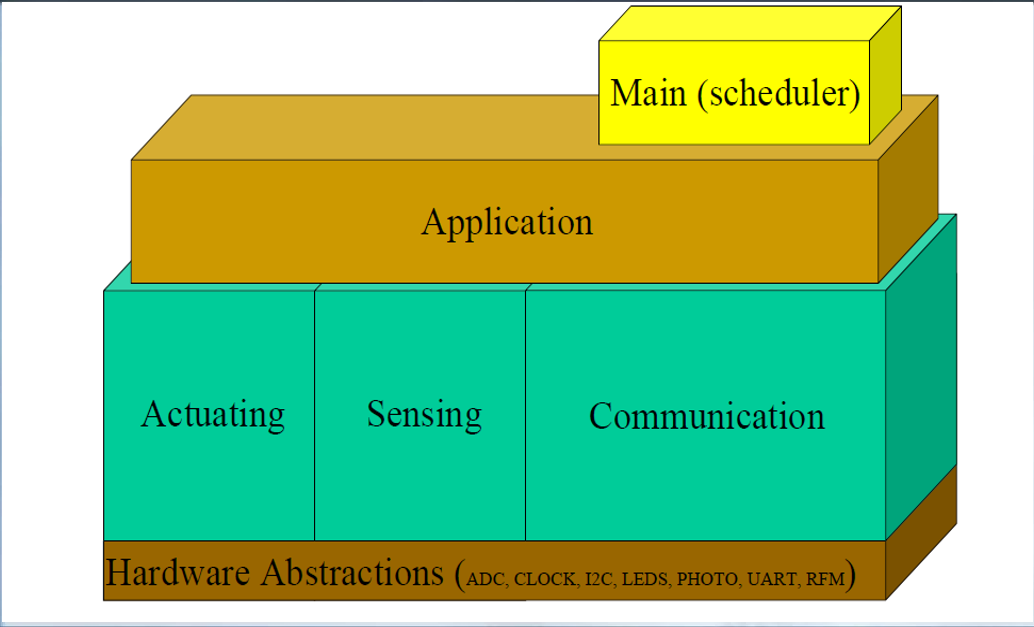
* + Smart vehicles are autonomous and can be understood as a network of sensors. These sensors communicate through low-power wireless area networks (LPWAN) which makes TinyOS a perfect fit.

### Smart cities:

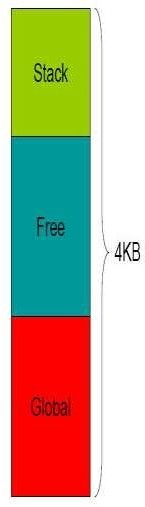
* + TinyOS is a viable solution for the low-power sensor requirements of smart cities’ utilities, power grids, Internet infrastructure and other applications.

### 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.



**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

1. 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

1. Programming model:

### Separation construction/composition

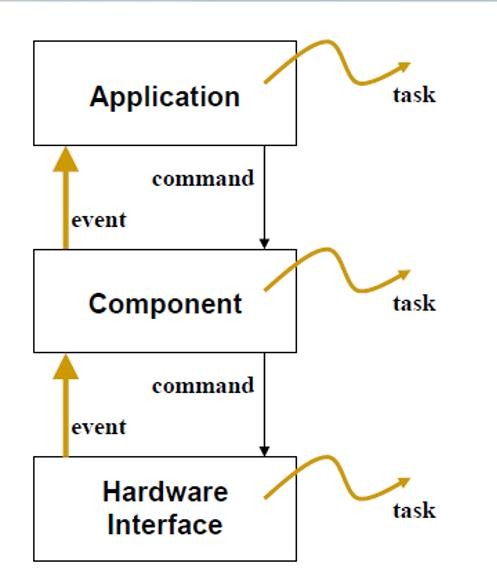
* + **Construction of Modules**
    - Modules implementation similar to C coding
    - Programs are built out of components
    - Each component specifies an interface
    - Interfaces are “hooks” for wiring components

### Composition of Configurations

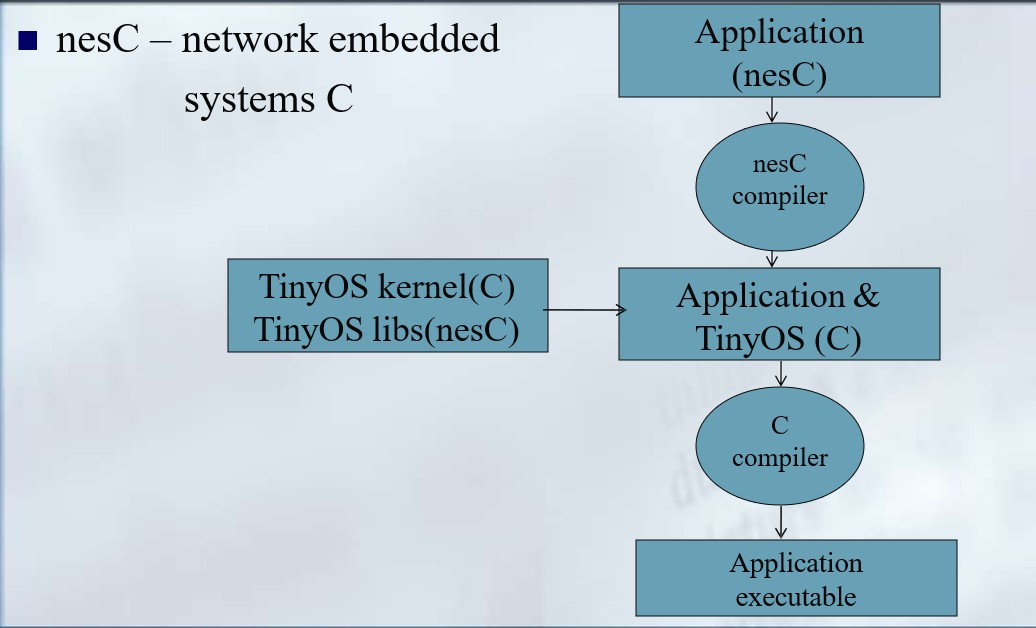
* + - Components are statically wired together
    - Increases programming efficiency (code reuse) a runtime efficiency.

1. 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:**



nesC language:



* 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.