PRACTICAL 1

AIM:

Installation and configuration of Instant Contiki OS with Cooja.

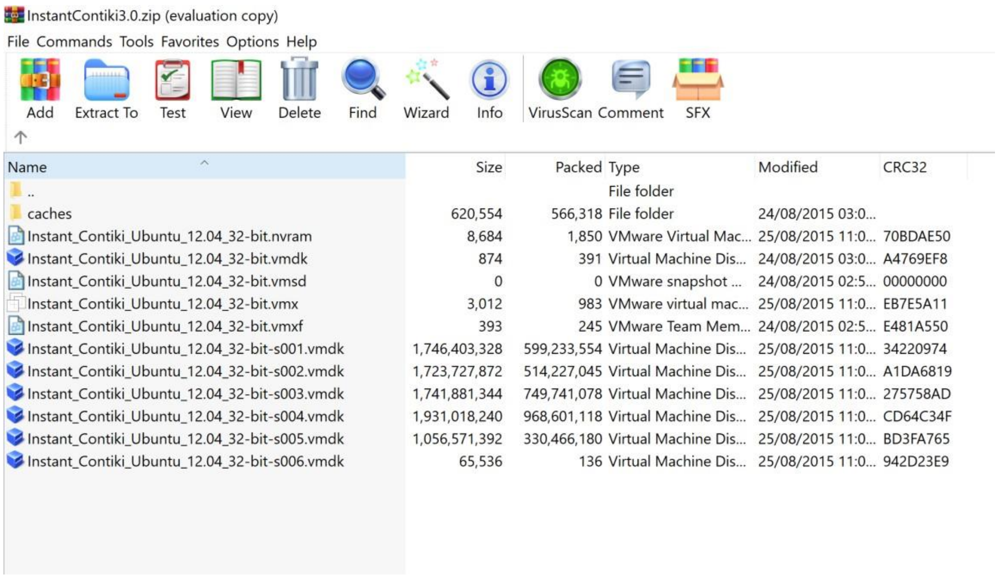
Contiki OS and Cooja:

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.

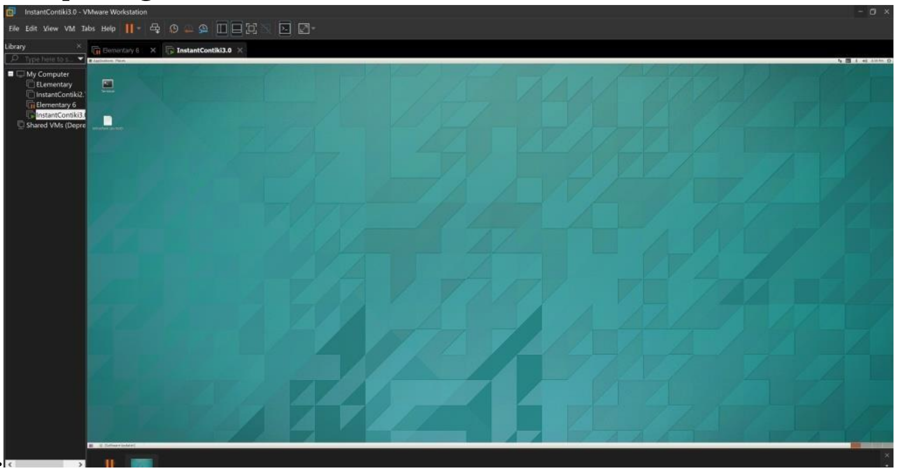
Steps to configure:

1. Contiki OS comes with cooja simulator and can be downloaded from this link: https://sourceforge.net/projects/contiki/files/Instant%20Contiki/Instant%20Contiki%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.

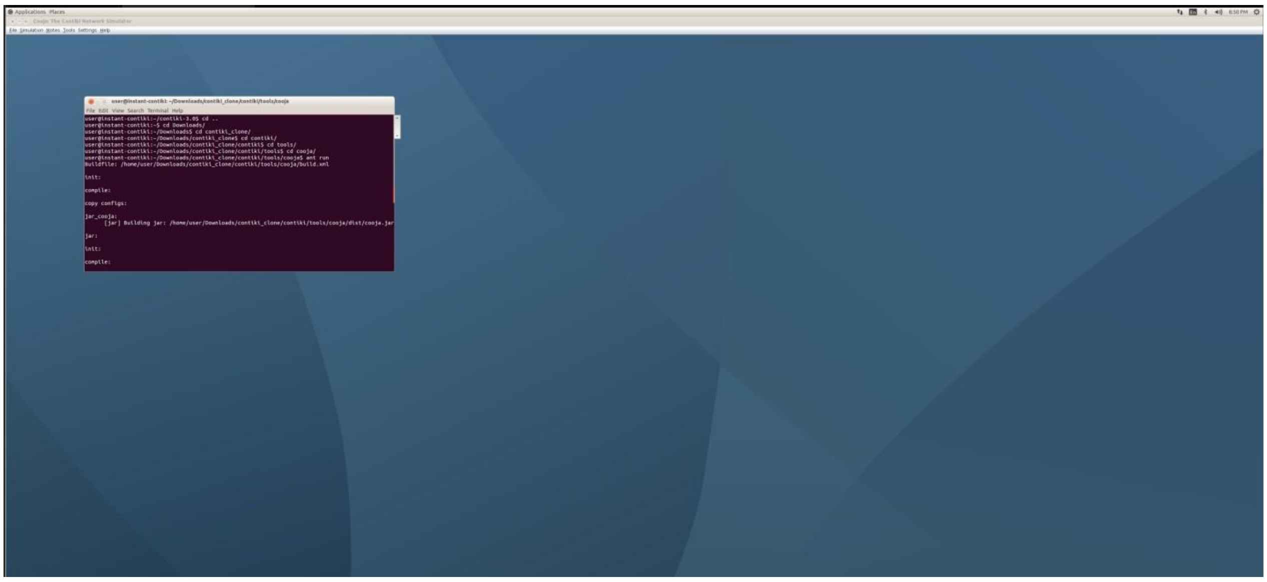

InstantContiki3.0.zip (evaluation copy)
File Commands Tools Favorites Options Help
Add
Extract To
Test
View
Delete
Find
Wizard
Info
VirusScan
Comment
SFX

Name	Size	Packed	Type	Modified	CRC32
..			File folder		
caches	620,554	566,318	File folder	24/08/2015 03:0...	
Instant_Contiki_Ubuntu_12.04_32-bit.nvram	8,684	1,850	VMware Virtual Mac...	25/08/2015 11:0...	70BDAE50
Instant_Contiki_Ubuntu_12.04_32-bit.vmdk	874	391	Virtual Machine Dis...	24/08/2015 03:0...	A4769EF8
Instant_Contiki_Ubuntu_12.04_32-bit.vmsd	0	0	VMware snapshot ...	24/08/2015 02:5...	00000000
Instant_Contiki_Ubuntu_12.04_32-bit.vmx	3,012	983	VMware virtual mac...	25/08/2015 11:0...	EB7E5A11
Instant_Contiki_Ubuntu_12.04_32-bit.vmxf	393	245	VMware Team Mem...	24/08/2015 02:5...	E481A550
Instant_Contiki_Ubuntu_12.04_32-bit-s001.vmdk	1,746,403,328	599,233,554	Virtual Machine Dis...	25/08/2015 11:0...	34220974
Instant_Contiki_Ubuntu_12.04_32-bit-s002.vmdk	1,723,727,872	514,227,045	Virtual Machine Dis...	25/08/2015 11:0...	A1DA6819
Instant_Contiki_Ubuntu_12.04_32-bit-s003.vmdk	1,741,881,344	749,741,078	Virtual Machine Dis...	25/08/2015 11:0...	275758AD
Instant_Contiki_Ubuntu_12.04_32-bit-s004.vmdk	1,931,018,240	968,601,118	Virtual Machine Dis...	25/08/2015 11:0...	CD64C34F
Instant_Contiki_Ubuntu_12.04_32-bit-s005.vmdk	1,056,571,392	330,466,180	Virtual Machine Dis...	25/08/2015 11:0...	BD3FA765
Instant_Contiki_Ubuntu_12.04_32-bit-s006.vmdk	65,536	136	Virtual Machine Dis...	25/08/2015 11:0...	942D23E9

2. After importing the .vmdk file into the virtual environment the machine is ready to run.



3. 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.


user@instant-contiki: ~/Downloads/contiki_clone/contiki/tools/cooja
File Edit View Search Terminal Help

```
user@instant-contiki:~/contiki-3.0$ cd ..
user@instant-contiki:~$ cd Downloads/
user@instant-contiki:~/Downloads$ cd contiki_clone/
user@instant-contiki:~/Downloads/contiki_clone$ cd contiki/
user@instant-contiki:~/Downloads/contiki_clone/contiki$ cd tools/
user@instant-contiki:~/Downloads/contiki_clone/contiki/tools$ cd cooja/
user@instant-contiki:~/Downloads/contiki_clone/contiki/tools/cooja$ ant run
Buildfile: /home/user/Downloads/contiki_clone/contiki/tools/cooja/build.xml

init:

compile:

copy configs:

jar_cooja:
      [jar] Building jar: /home/user/Downloads/contiki_clone/contiki/tools/cooja/dist/cooja.jar

jar:

init:

compile:
```

CONCLUSION

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

PRACTICAL 2

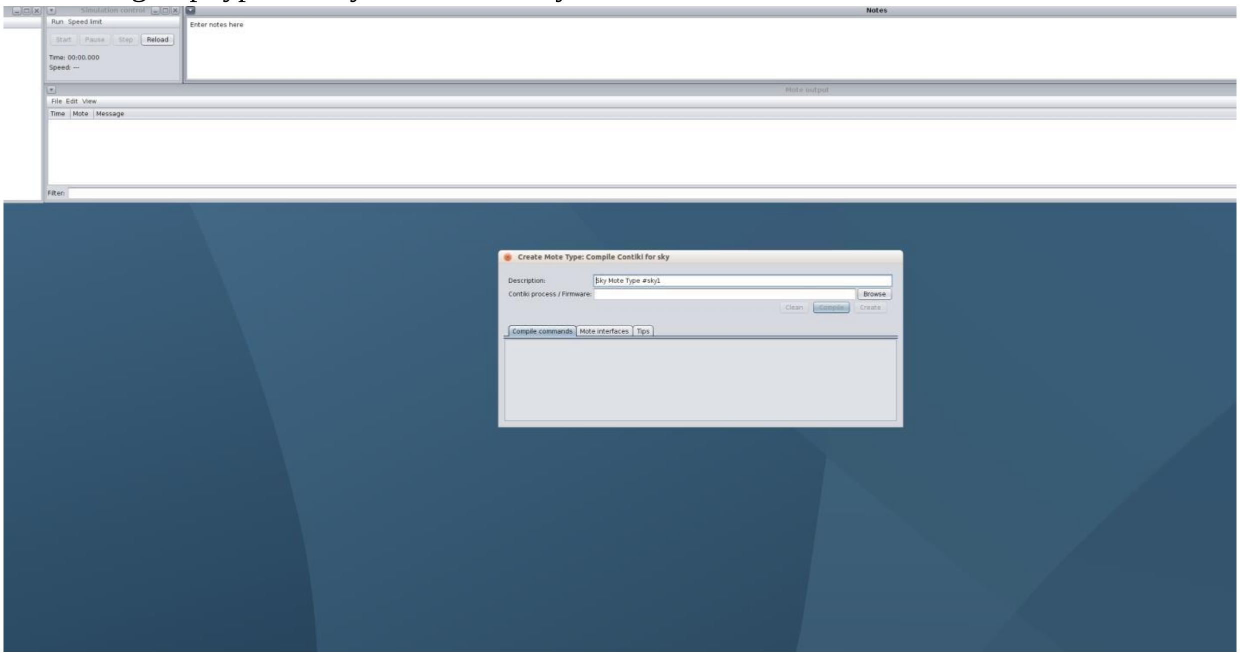
AIM:
Study of different types of motes and deploy them using IoTarchitecture. Simulate HelloWorld program using Cooja.

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.

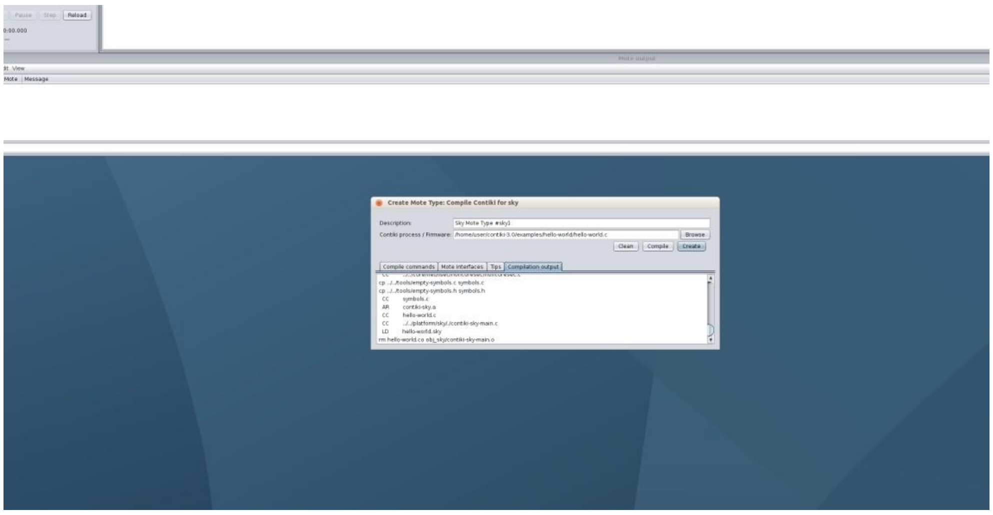
Steps to configure:
1. After starting cooja we will create a new simulation and to begin we will create a new mote group type i.e Sky Mote and 3 sky motes into our simulation.



Time: 00:00.000
Speed: ---
Enter notes here

Create Mote Type: Compile Contiki for sky
Description: Sky Mote Type #sky1
Contiki process / Firmware:

2. 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.

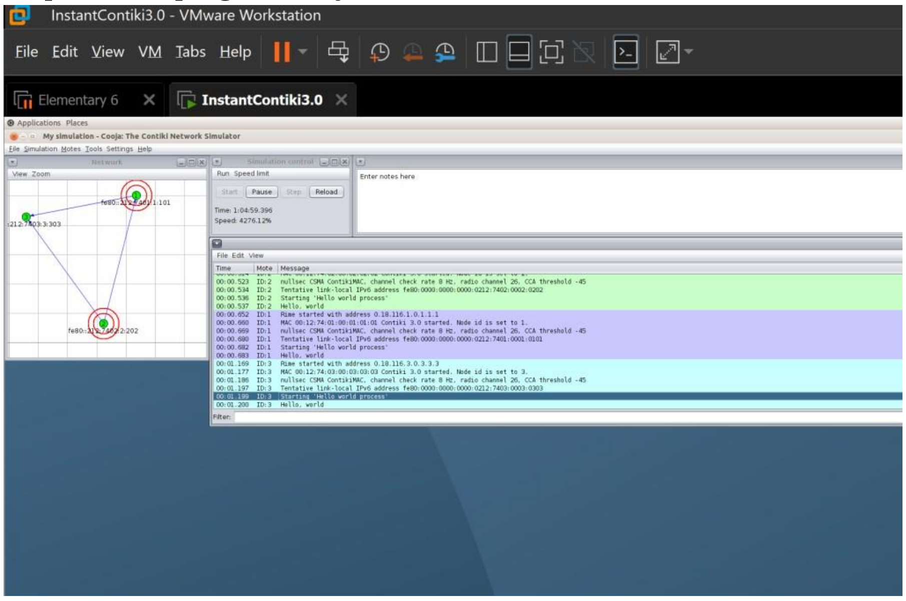


Create Mote Type: Compile Contiki for sky

Description: Sky Mote Type #sky1
Contiki process / Firmware: /home/user/contiki-3.0/examples/hello-world/hello-world.c

```
cp ../../tools/empty-symbols.c symbols.c
cp ../../tools/empty-symbols.h symbols.h
  CC        symbols.c
  AR        contiki-sky.a
  CC        hello-world.c
  CC        ../../platform/sky/./contiki-sky-main.c
  LD        hello-world.sky
rm hello-world.co obj_sky/contiki-sky-main.o
```

3. 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.


InstantContiki3.0 - VMware Workstation
My simulation - Cooja: The Contiki Network Simulator
fe80::212:7401:1:101
:212:7403:3:303
fe80::212:7402:2:202

Time: 1:04:59.396
Speed: 4276.12%

```
Time        Mote   Message
00:00.523   ID:2   nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26, CCA threshold -45
00:00.534   ID:2   Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7402:0002:0202
00:00.536   ID:2   Starting 'Hello world process'
00:00.537   ID:2   Hello, world
00:00.652   ID:1   Rime started with address 0.18.116.1.0.1.1.1
00:00.660   ID:1   MAC 00:12:74:01:00:01:01:01 Contiki 3.0 started. Node id is set to 1.
00:00.669   ID:1   nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26, CCA threshold -45
00:00.680   ID:1   Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7401:0001:0101
00:00.682   ID:1   Starting 'Hello world process'
00:00.683   ID:1   Hello, world
00:01.169   ID:3   Rime started with address 0.18.116.3.0.3.3.3
00:01.177   ID:3   MAC 00:12:74:03:00:03:03:03 Contiki 3.0 started. Node id is set to 3.
00:01.186   ID:3   nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26, CCA threshold -45
00:01.197   ID:3   Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7403:0003:0303
00:01.199   ID:3   Starting 'Hello world process'
00:01.200   ID:3   Hello, world
```

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.

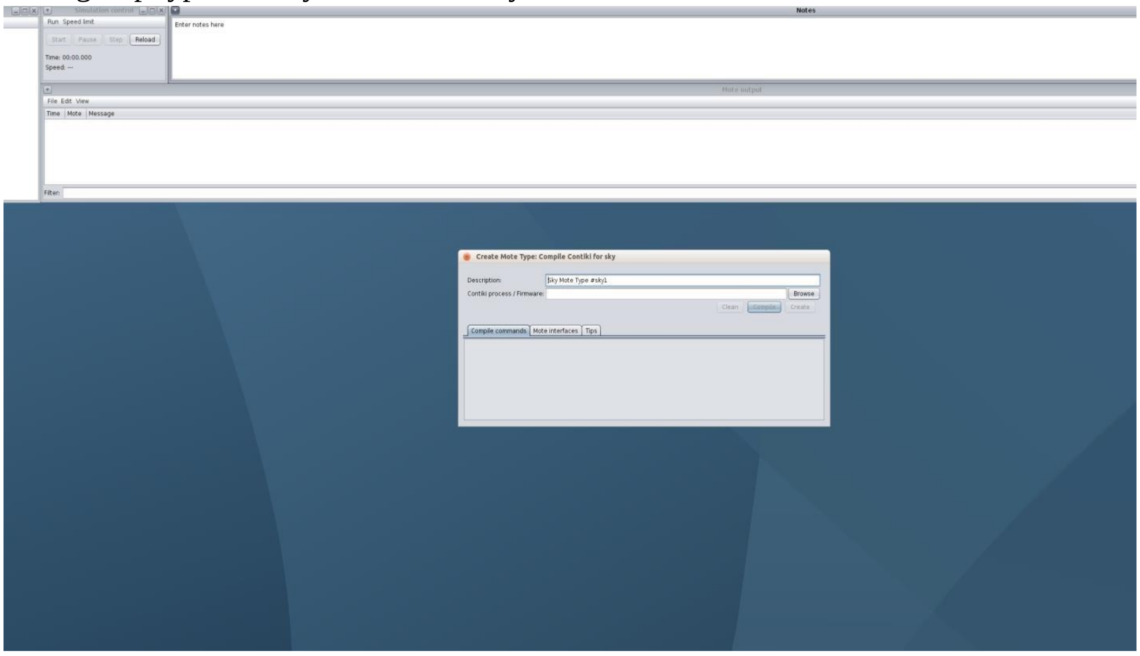
PRACTICAL 3

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.

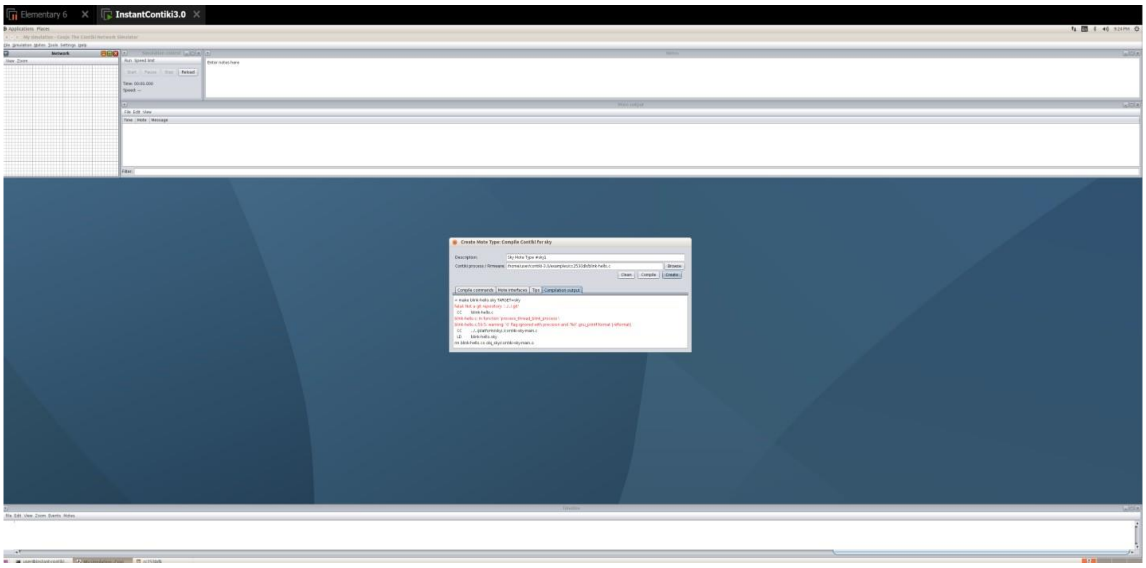


Create Mote Type: Compile Contiki for sky

Description: Sky Mote Type #sky1

Contiki process / Firmware:

2. 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.



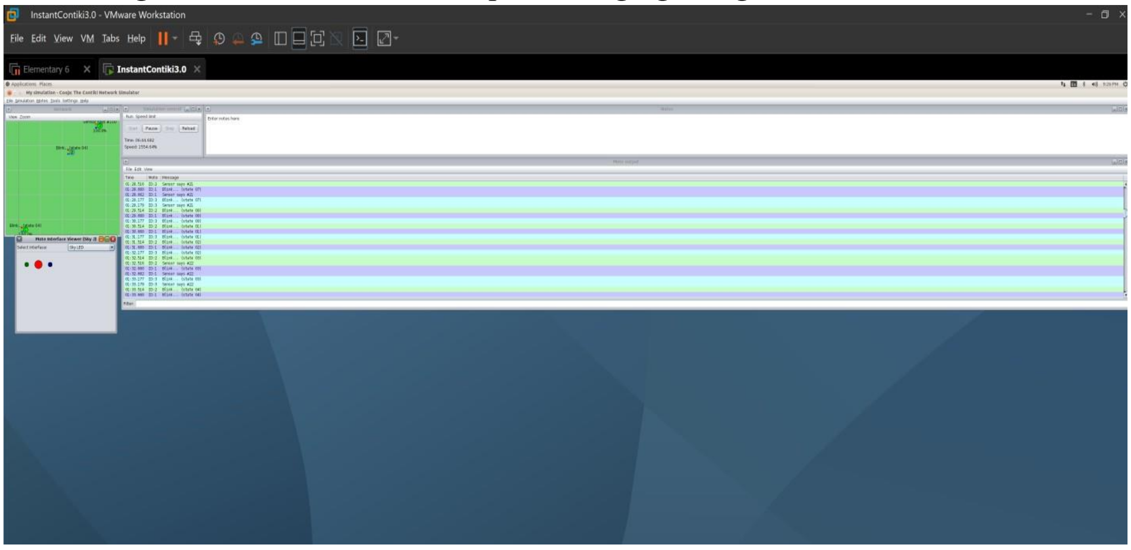
Create Mote Type: Compile Contiki for sky

Description: Sky Mote Type #sky1

Contiki process / Firmware: /home/user/contiki-3.0/examples/cc2530dk/blink-hello.c

```
> make blink-hello.sky TARGET=sky
fatal: Not a git repository: '../../.git'
  CC        blink-hello.c
blink-hello.c: In function 'process_thread_blink_process':
blink-hello.c:59:5: warning: '0' flag ignored with precision and '%X' gnu_printf format [-Wformat]
  CC        ../../platform/sky/./contiki-sky-main.c
  LD        blink-hello.sky
rm blink-hello.co obj_sky/contiki-sky-main.o
```

3. 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.



```
Time: 06:44.682
Speed: 1554.64%

Time       Mote  Message
01:28.516  ID:2  Sensor says #21
01:28.680  ID:1  Blink... (state 07)
01:28.682  ID:1  Sensor says #21
01:29.177  ID:3  Blink... (state 07)
01:29.179  ID:3  Sensor says #21
01:29.514  ID:2  Blink... (state 00)
01:29.680  ID:1  Blink... (state 00)
01:30.177  ID:3  Blink... (state 00)
01:30.514  ID:2  Blink... (state 01)
01:30.680  ID:1  Blink... (state 01)
01:31.177  ID:3  Blink... (state 01)
01:31.514  ID:2  Blink... (state 02)
01:31.680  ID:1  Blink... (state 02)
01:32.177  ID:3  Blink... (state 02)
01:32.514  ID:2  Blink... (state 03)
01:32.516  ID:2  Sensor says #22
01:32.680  ID:1  Blink... (state 03)
01:32.682  ID:1  Sensor says #22
01:33.177  ID:3  Blink... (state 03)
01:33.179  ID:3  Sensor says #22
01:33.514  ID:2  Blink... (state 04)
01:33.680  ID:1  Blink... (state 04)
```

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.

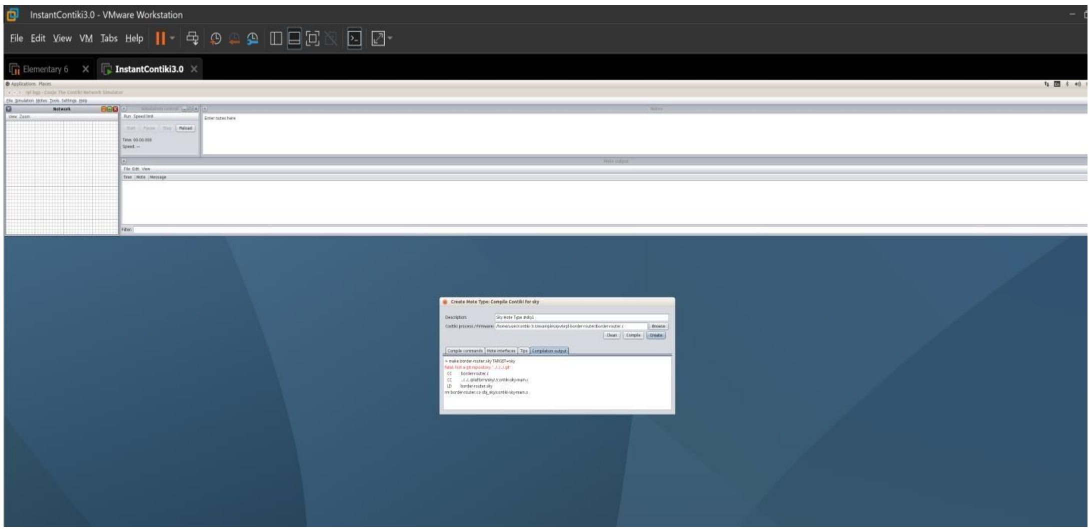
PRACTICAL 4

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.

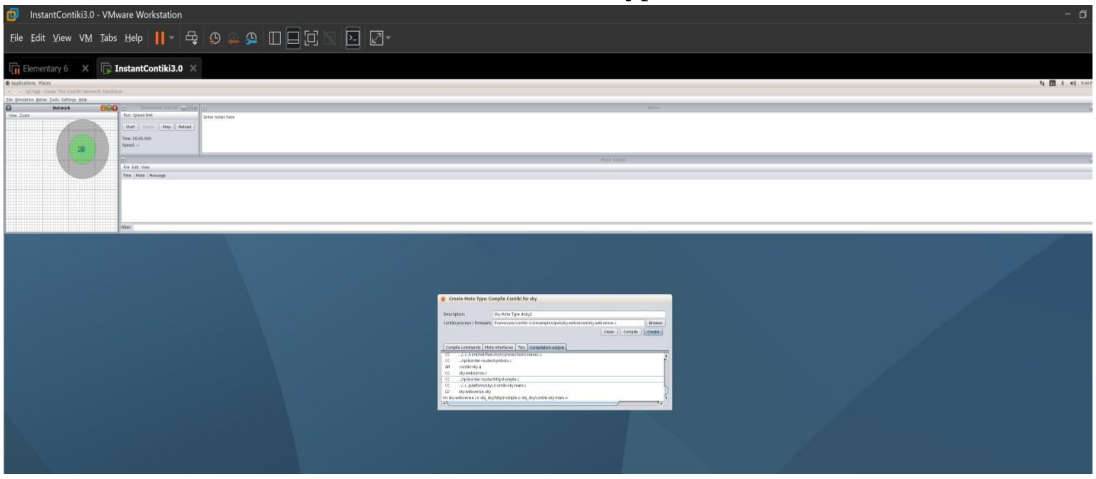


Description: Sky Mote Type #sky1

Contiki process / Firmware: /home/user/contiki-3.0/examples/ipv6/rpl-border-router/border-router.c

```
> make border-router.sky TARGET=sky
fatal: Not a git repository: '../../../.git'
  CC        border-router.c
  CC        ../../../platform/sky/./contiki-sky-main.c
  LD        border-router.sky
rm border-router.co obj_sky/contiki-sky-main.o
```

2. 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.

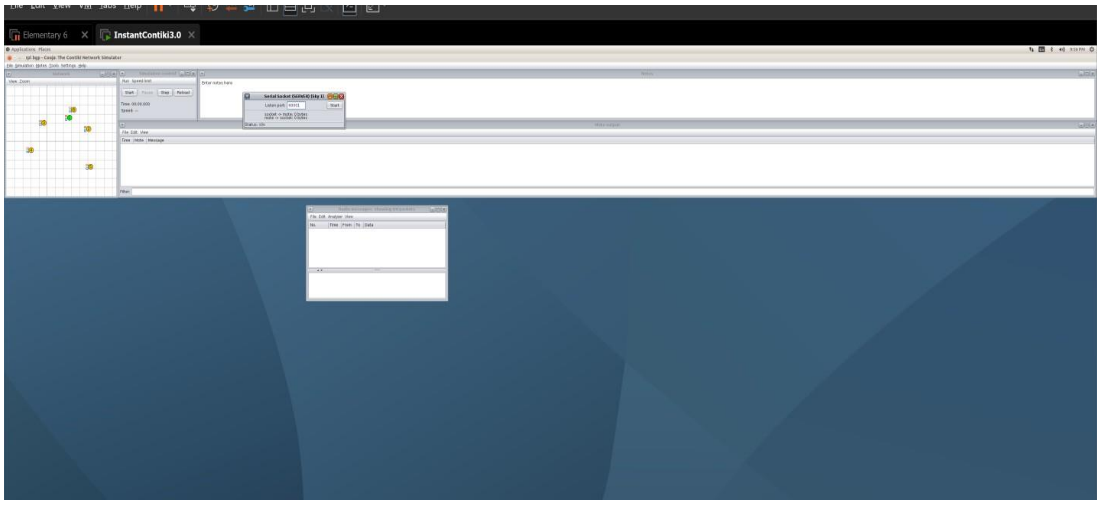


Description: Sky Mote Type #sky2

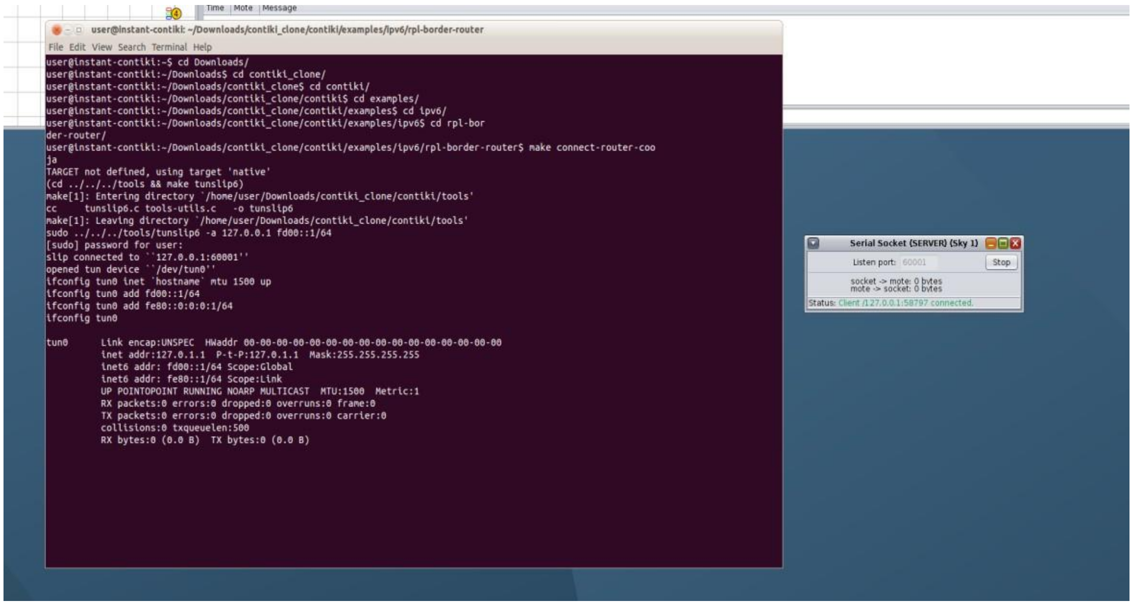
Contiki process / Firmware: /home/user/contiki-3.0/examples/ipv6/sky-websense/sky-websense.c

```
  CC        ../../../core/net/mac/contikimac/contikimac.c
  CC        ../rpl-border-router/symbols.c
  AR        contiki-sky.a
  CC        sky-websense.c
  CC        ../rpl-border-router/httpd-simple.c
  CC        ../../../platform/sky/./contiki-sky-main.c
  LD        sky-websense.sky
rm sky-websense.co obj_sky/httpd-simple.o obj_sky/contiki-sky-main.o
```

3. 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.



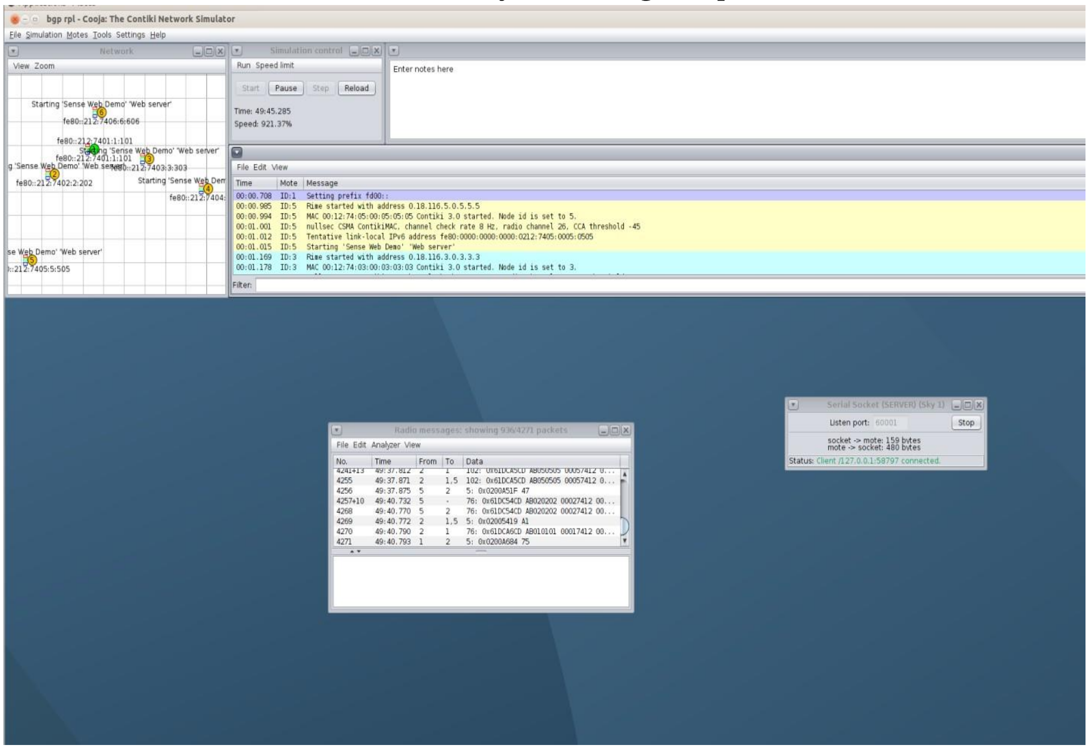
4. After starting the listening port we have to go to the example directory of router and type the command make connect router command to start the connection.



```
user@instant-contiki: ~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router
user@instant-contiki:~$ cd Downloads/
user@instant-contiki:~/Downloads$ cd contiki_clone/
user@instant-contiki:~/Downloads/contiki_clone$ cd contiki/
user@instant-contiki:~/Downloads/contiki_clone/contiki$ cd examples/
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples$ cd ipv6/
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6$ cd rpl-bor
der-router/
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$ make connect-router-coo
ja
TARGET not defined, using target 'native'
(cd ../../../tools && make tunslip6)
make[1]: Entering directory `/home/user/Downloads/contiki_clone/contiki/tools'
cc     tunslip6.c tools-utils.c   -o tunslip6
make[1]: Leaving directory `/home/user/Downloads/contiki_clone/contiki/tools'
sudo ../../../tools/tunslip6 -a 127.0.0.1 fd00::1/64
[sudo] password for user:
slip connected to ``127.0.0.1:60001''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::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: fd00::1/64 Scope:Global
          inet6 addr: fe80::1/64 Scope:Link
          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)
```

Serial Socket (SERVER) (Sky 1)
Listen port: 60001
socket -> mote: 0 bytes
mote -> socket: 0 bytes
Status: Client /127.0.0.1:58797 connected.

5. 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.

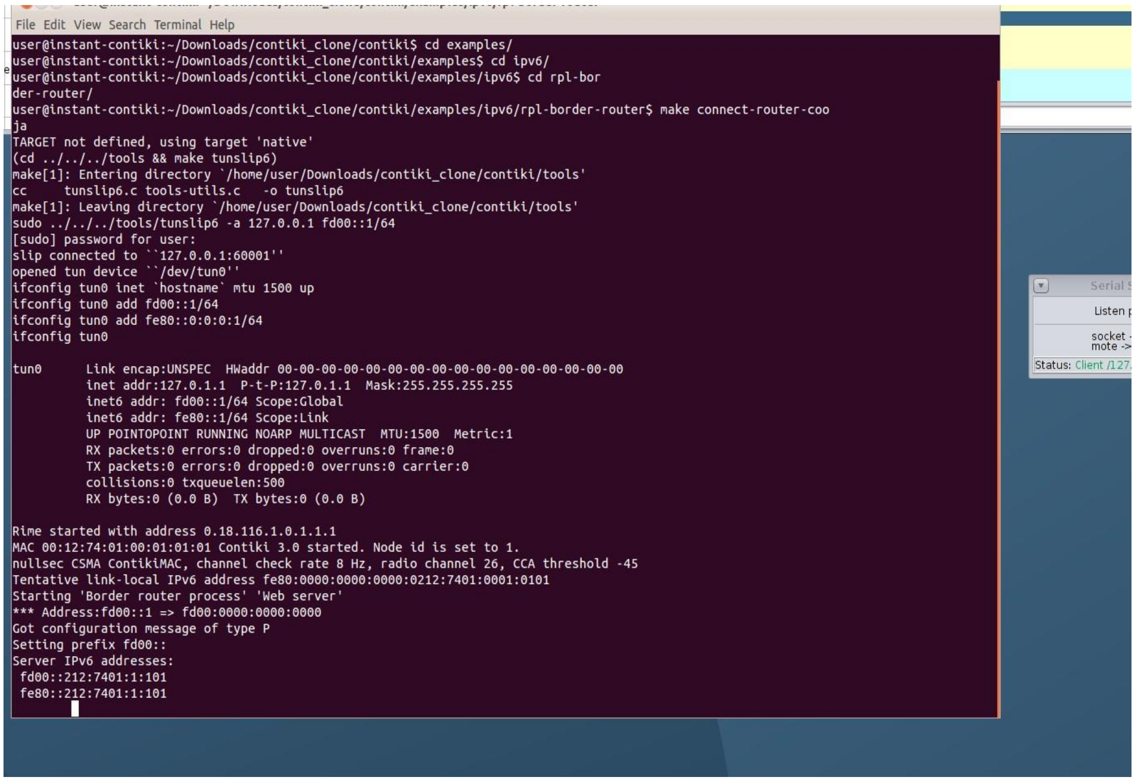

bgp rpl - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
Simulation control
Run Speed limit
Start
Pause
Step
Reload
Enter notes here
Serial Socket (SERVER) (Sky 1)
Listen port: 60001
Stop
socket -> mote: 159 bytes
mote -> socket: 480 bytes
Status: Client /127.0.0.1:58797 connected.

```
Time: 49:45.285
Speed: 921.37%
```

```
Time       Mote  Message
00:00.708  ID:1  Setting prefix fd00::
00:00.985  ID:5  Rime started with address 0.18.116.5.0.5.5.5
00:00.994  ID:5  MAC 00:12:74:05:00:05:05:05 Contiki 3.0 started. Node id is set to 5.
00:01.001  ID:5  nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26, CCA threshold -45
00:01.012  ID:5  Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7405:0005:0505
00:01.015  ID:5  Starting 'Sense Web Demo' 'Web server'
00:01.169  ID:3  Rime started with address 0.18.116.3.0.3.3.3
00:01.178  ID:3  MAC 00:12:74:03:00:03:03:03 Contiki 3.0 started. Node id is set to 3.
```

```
Radio messages: showing 936/4271 packets
No.      Time       From  To   Data
4241+13  49:37.812  2     1    102: 0x61DCA5CD AB050505 00057412 0...
4255     49:37.871  2     1,5  102: 0x61DCA5CD AB050505 00057412 0...
4256     49:37.875  5     2    5: 0x0200A51F 47
4257+10  49:40.732  5     -    76: 0x61DC54CD AB020202 00027412 00...
4268     49:40.770  5     2    76: 0x61DC54CD AB020202 00027412 00...
4269     49:40.772  2     1,5  5: 0x02005419 A1
4270     49:40.790  2     1    76: 0x61DCA6CD AB010101 00017412 00...
4271     49:40.793  1     2    5: 0x0200A684 75
```

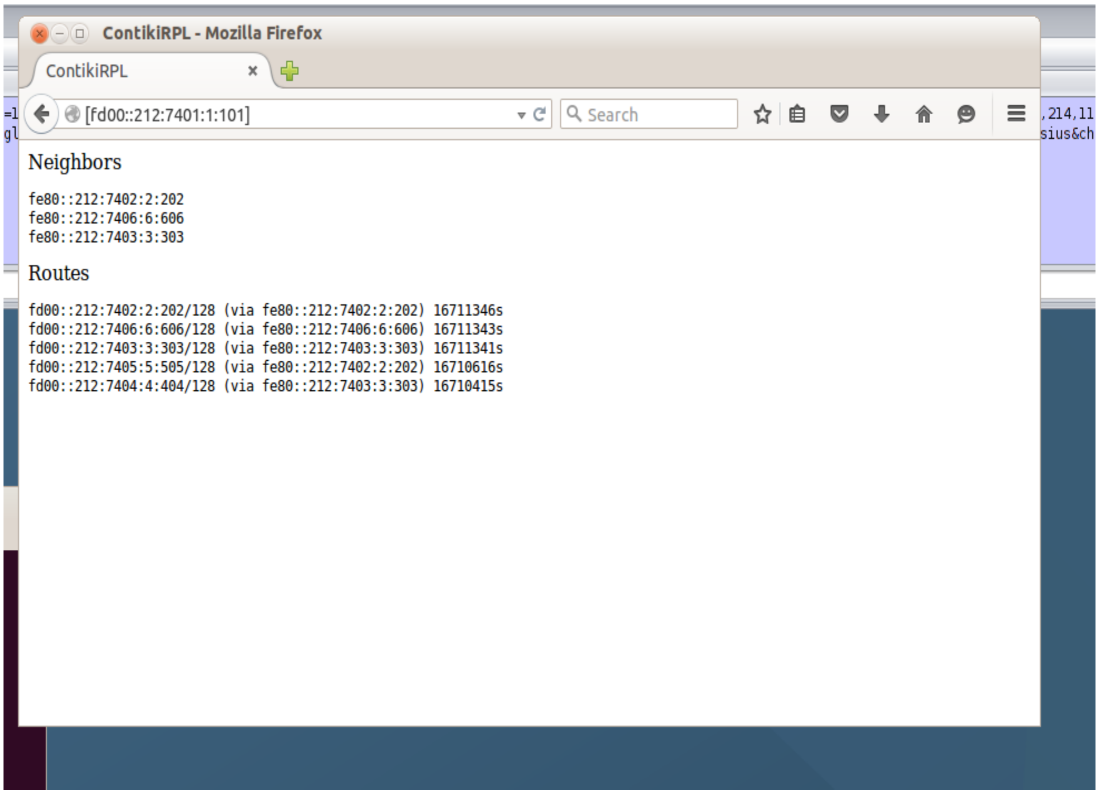
6. Also when we see the termial where we connected the router we would have the ip address of the border router.



```
user@instant-contiki:~/Downloads/contiki_clone/contiki$ cd examples/
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples$ cd ipv6/
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6$ cd rpl-bor
der-router/
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$ make connect-router-coo
ja
TARGET not defined, using target 'native'
(cd ../../../tools && make tunslip6)
make[1]: Entering directory `/home/user/Downloads/contiki_clone/contiki/tools'
cc     tunslip6.c tools-utils.c   -o tunslip6
make[1]: Leaving directory `/home/user/Downloads/contiki_clone/contiki/tools'
sudo ../../../tools/tunslip6 -a 127.0.0.1 fd00::1/64
[sudo] password for user:
slip connected to ``127.0.0.1:60001''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::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: fd00::1/64 Scope:Global
          inet6 addr: fe80::1/64 Scope:Link
          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)

Rime started with address 0.18.116.1.0.1.1.1
MAC 00:12:74:01:00:01:01:01 Contiki 3.0 started. Node id is set to 1.
nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26, CCA threshold -45
Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7401:0001:0101
Starting 'Border router process' 'Web server'
*** Address:fd00::1 => fd00:0000:0000:0000
Got configuration message of type P
Setting prefix fd00::
Server IPv6 addresses:
 fd00::212:7401:1:101
 fe80::212:7401:1:101
```

7. 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.



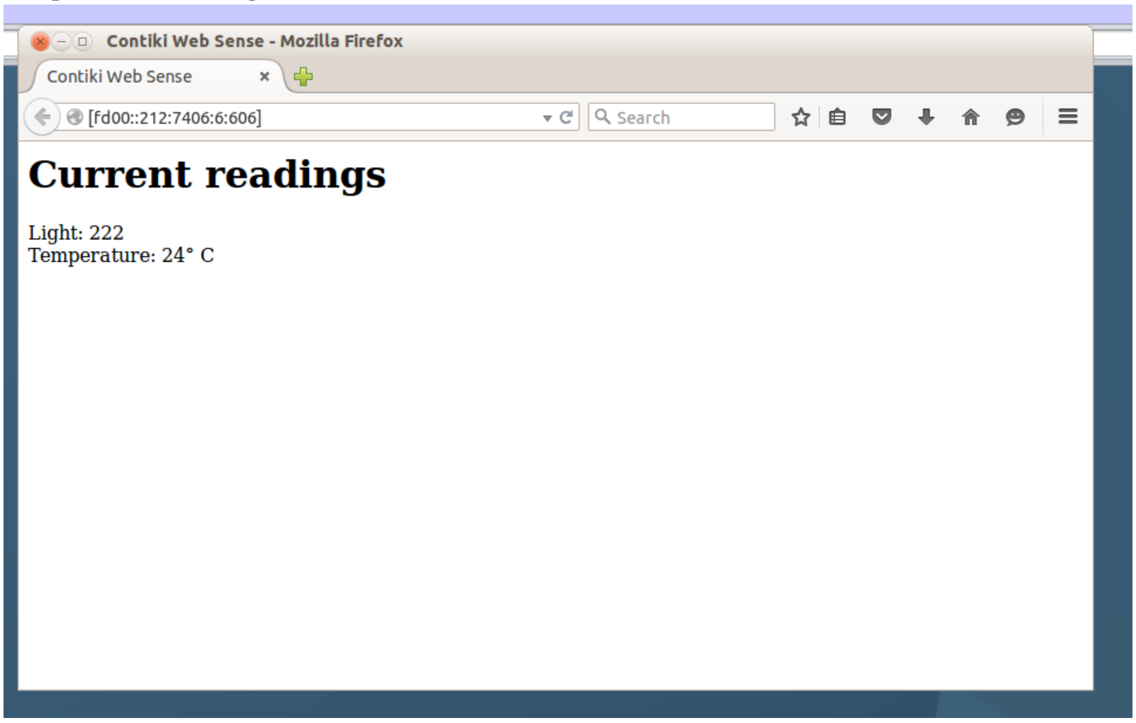
Neighbors

```
fe80::212:7402:2:202
fe80::212:7406:6:606
fe80::212:7403:3:303
```

Routes

```
fd00::212:7402:2:202/128 (via fe80::212:7402:2:202) 16711346s
fd00::212:7406:6:606/128 (via fe80::212:7406:6:606) 16711343s
fd00::212:7403:3:303/128 (via fe80::212:7403:3:303) 16711341s
fd00::212:7405:5:505/128 (via fe80::212:7402:2:202) 16710616s
fd00::212:7404:4:404/128 (via fe80::212:7403:3:303) 16710415s
```

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



Current readings

Light: 222
Temperature: 24° C

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.

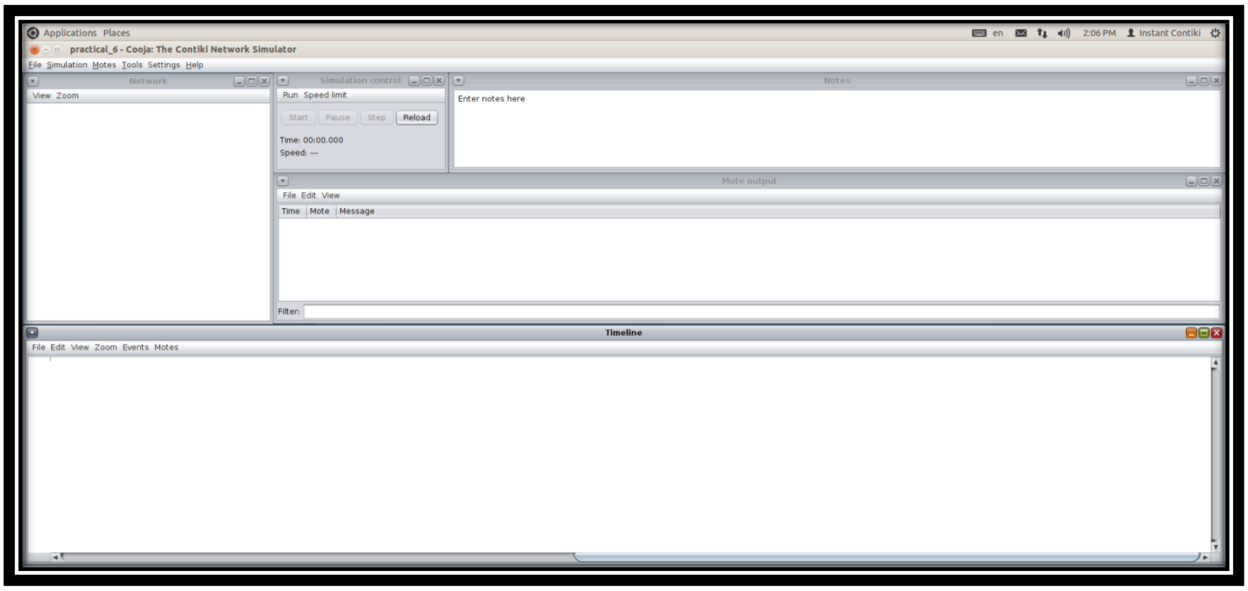
UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed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

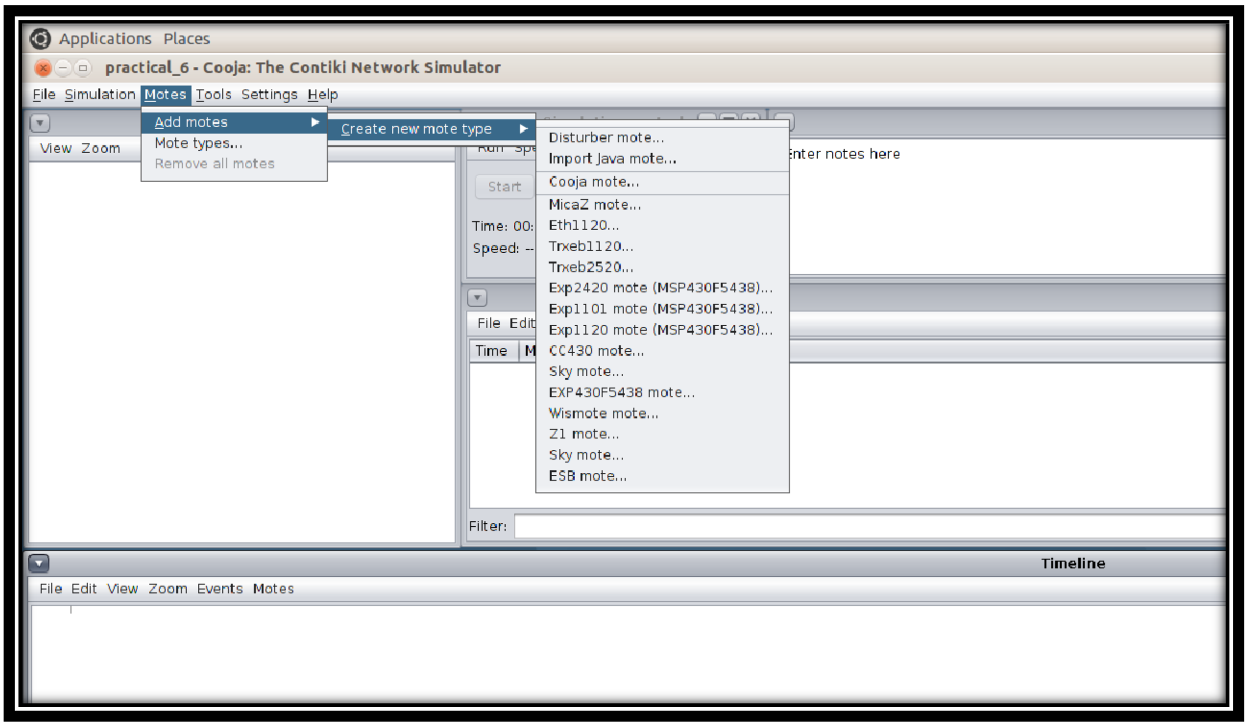
PRACTICAL:

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.

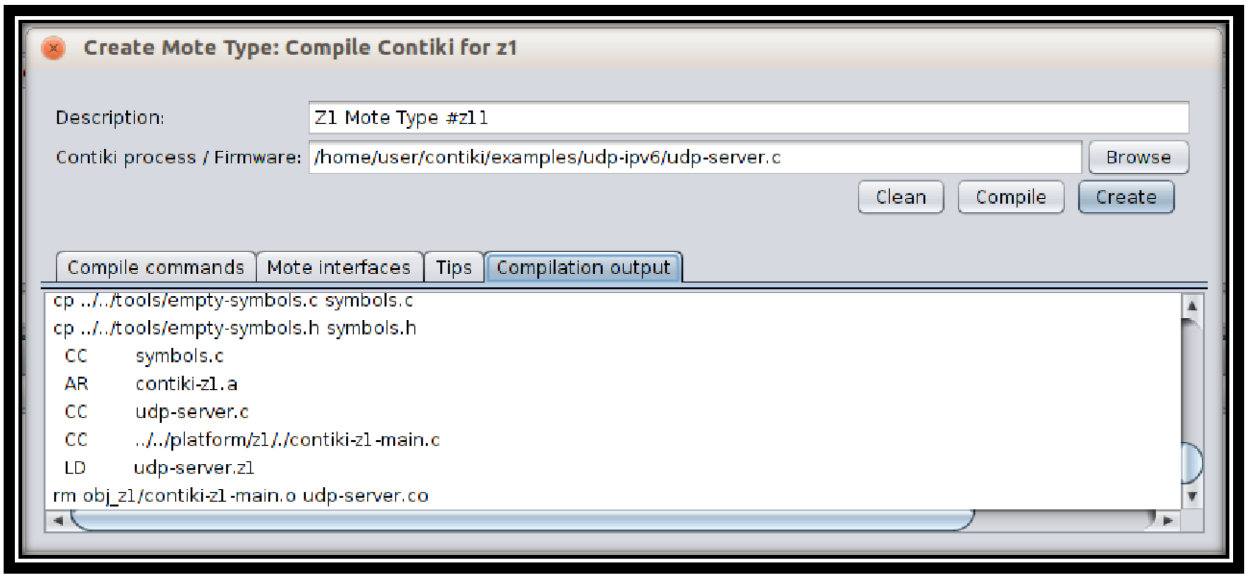


Then, we will first add the router mote, so we will go to Motes > Create new mote type >Z1 mote.



We will browse through example folder to find udp-ipv6 >udp-server.c.

We will compile it and press create.

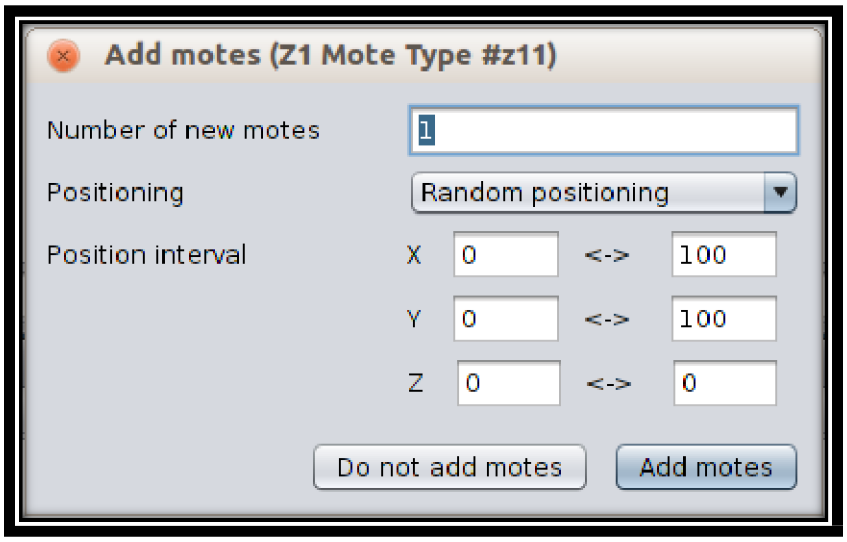


Description: Z1 Mote Type #z11

Contiki process / Firmware: /home/user/contiki/examples/udp-ipv6/udp-server.c

```
cp ../../tools/empty-symbols.c symbols.c
cp ../../tools/empty-symbols.h symbols.h
  CC        symbols.c
  AR        contiki-z1.a
  CC        udp-server.c
  CC        ../../platform/z1/./contiki-z1-main.c
  LD        udp-server.z1
rm obj_z1/contiki-z1-main.o udp-server.co
```

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



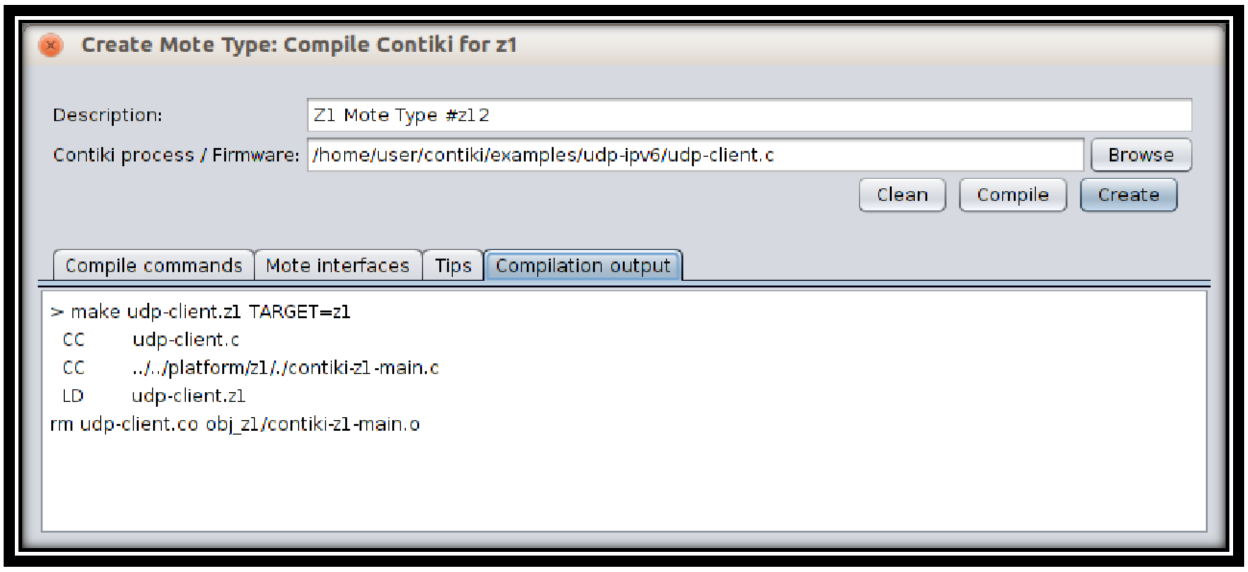
Number of new motes: 1

Positioning: Random positioning

Position interval: X 0 <-> 100; Y 0 <-> 100; Z 0 <-> 0

To create receiving motes, we will follow the same process to add Z1 motes.

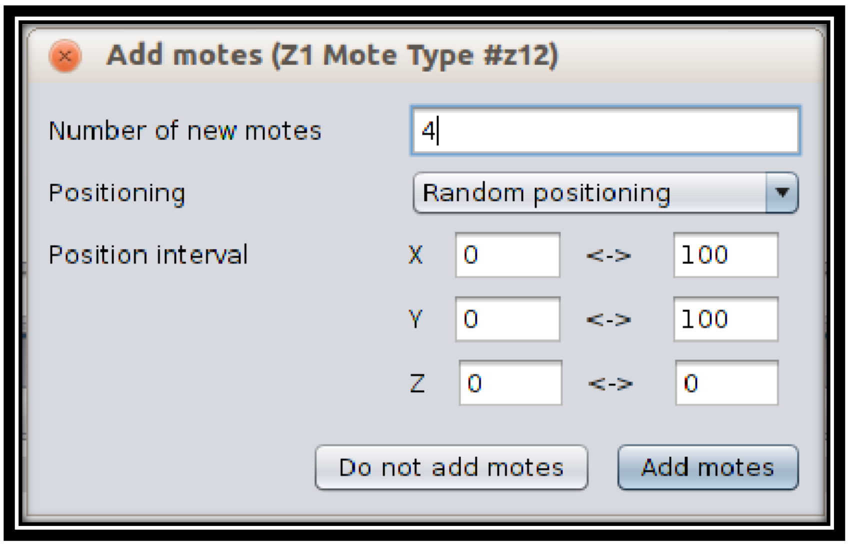
This time, we will browse to udp-ipv6 >udp-client.c.



Description: Z1 Mote Type #z12
Contiki process / Firmware: /home/user/contiki/examples/udp-ipv6/udp-client.c

```
> make udp-client.z1 TARGET=z1
  CC       udp-client.c
  CC       ../../platform/z1/./contiki-z1-main.c
  LD       udp-client.z1
rm udp-client.co obj_z1/contiki-z1-main.o
```

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

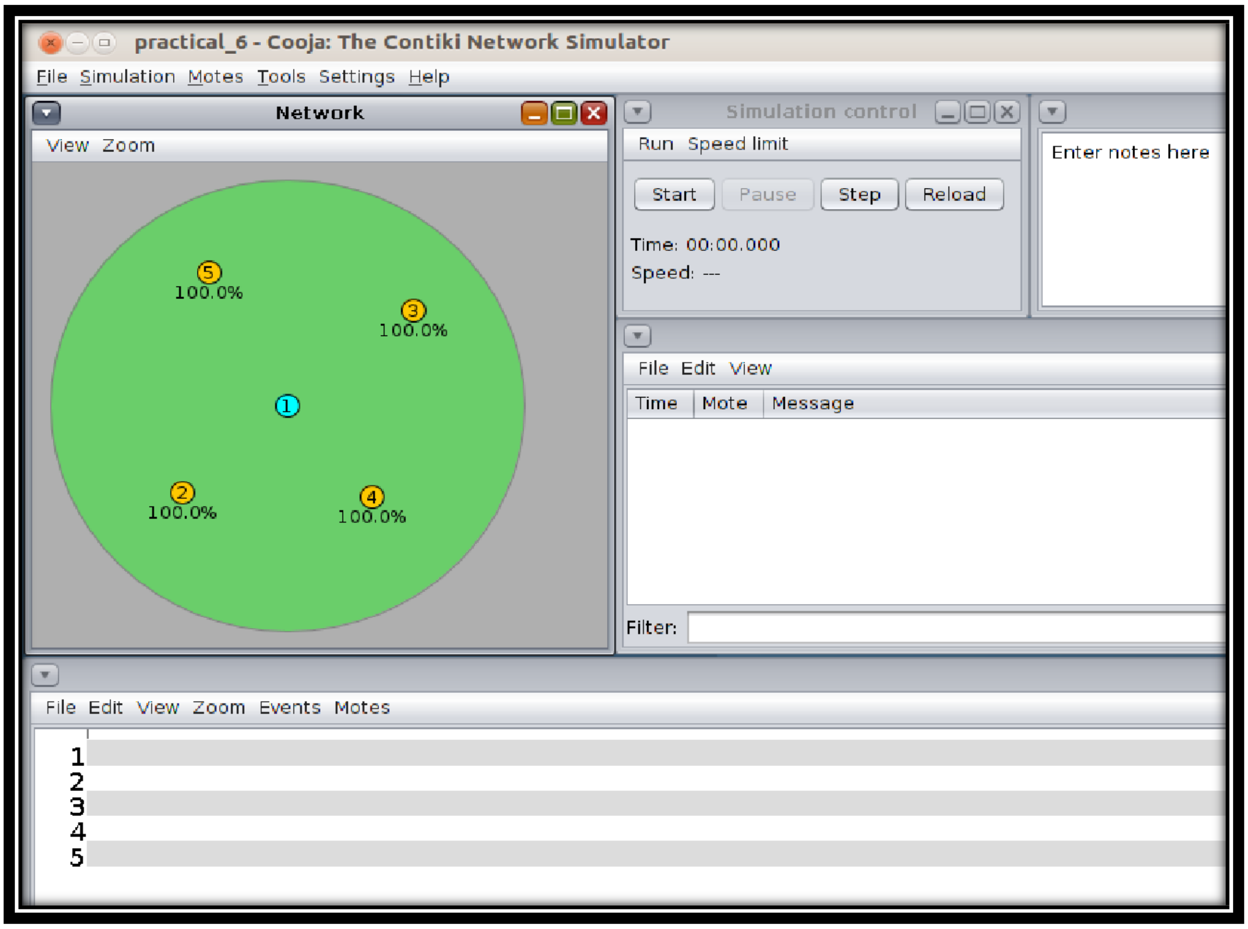


Number of new motes: 4
Positioning: Random positioning
Position interval: X 0 <-> 100, Y 0 <-> 100, Z 0 <-> 0

When we will add motes, we will be able to see a screen like this.

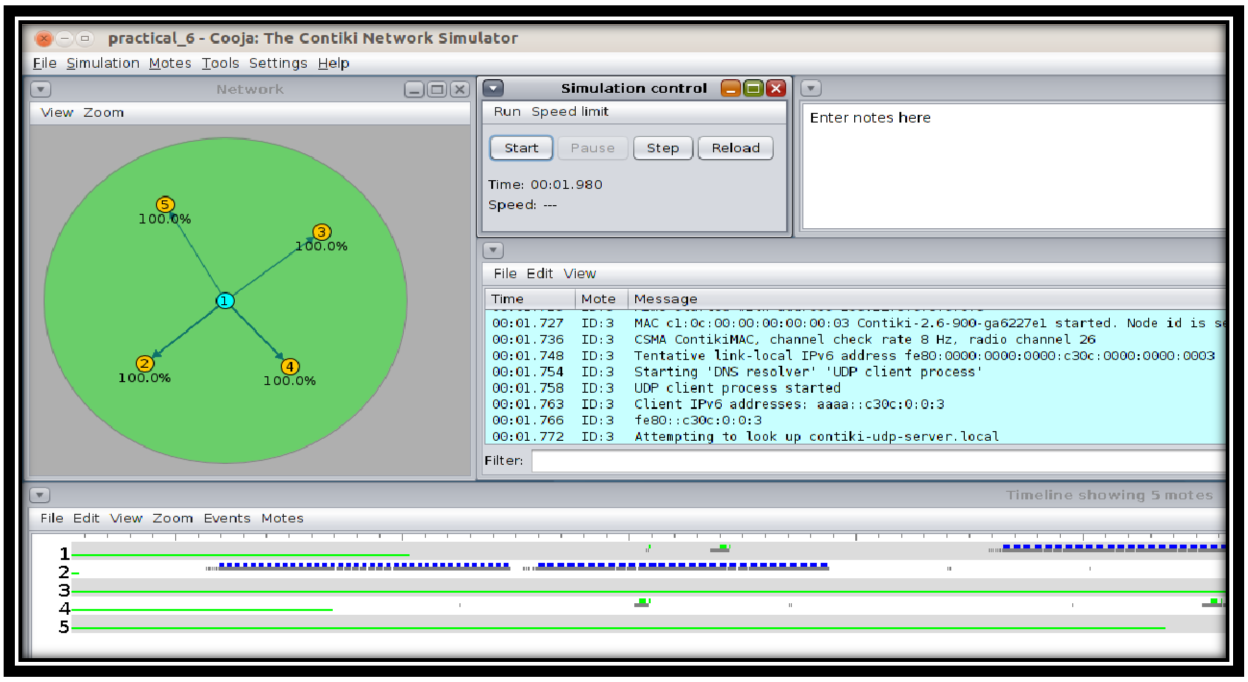
By clicking on the mote, you will be able to see its range.

We can change some view options as our convenience.



```
Time: 00:00.000
Speed: ---
```

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



```
Time: 00:01.980
Speed: ---

Time       Mote   Message
00:01.727  ID:3   MAC c1:0c:00:00:00:00:00:03 Contiki-2.6-900-ga6227e1 started. Node id is se
00:01.736  ID:3   CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26
00:01.748  ID:3   Tentative link-local IPv6 address fe80:0000:0000:0000:c30c:0000:0000:0003
00:01.754  ID:3   Starting 'DNS resolver' 'UDP client process'
00:01.758  ID:3   UDP client process started
00:01.763  ID:3   Client IPv6 addresses: aaaa::c30c:0:0:3
00:01.766  ID:3   fe80::c30c:0:0:3
00:01.772  ID:3   Attempting to look up contiki-udp-server.local
```

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

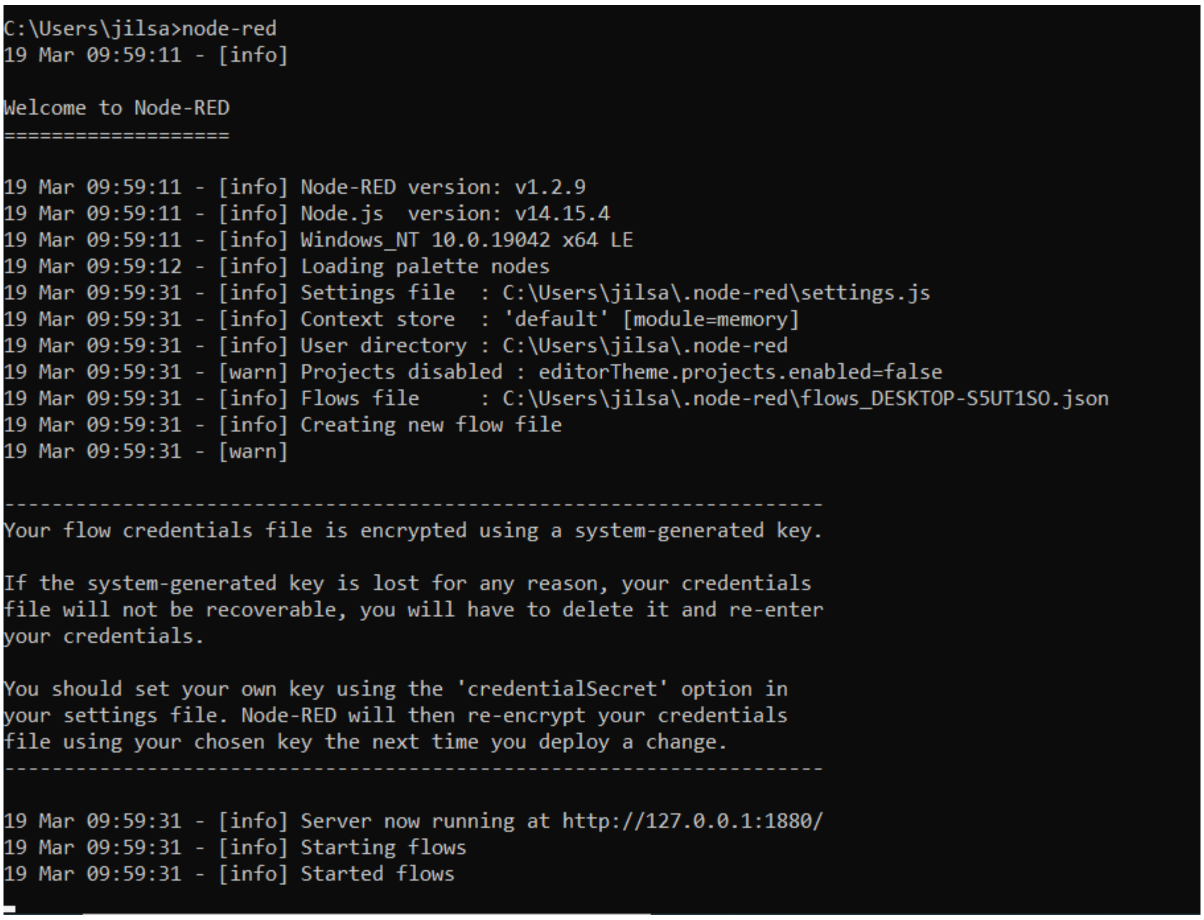
  - MQTT can scale to connect with millions of IoT devices.
- Reliable Message Delivery
  - Reliability of message delivery is important for many IoT use cases. This is why MQTT has 3 defined quality of service levels: 0 - at most once, 1- at least once, 2 - exactly once
- Support for Unreliable Networks
  - 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.

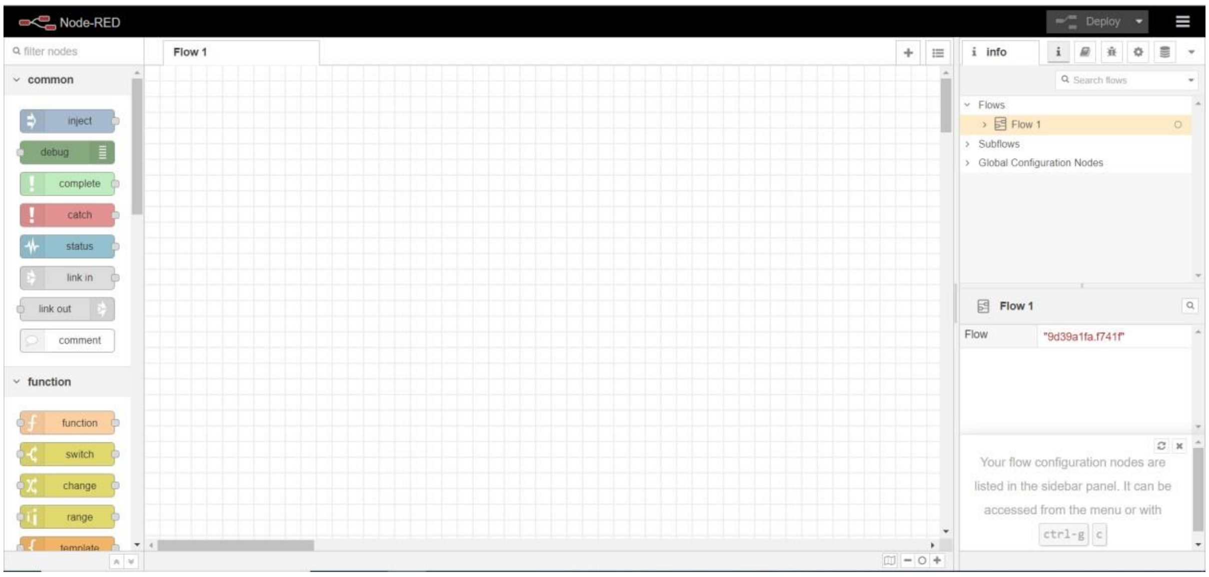
- npm install -g --unsafe-perm node-red

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



```
C:\Users\jilsa>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
```

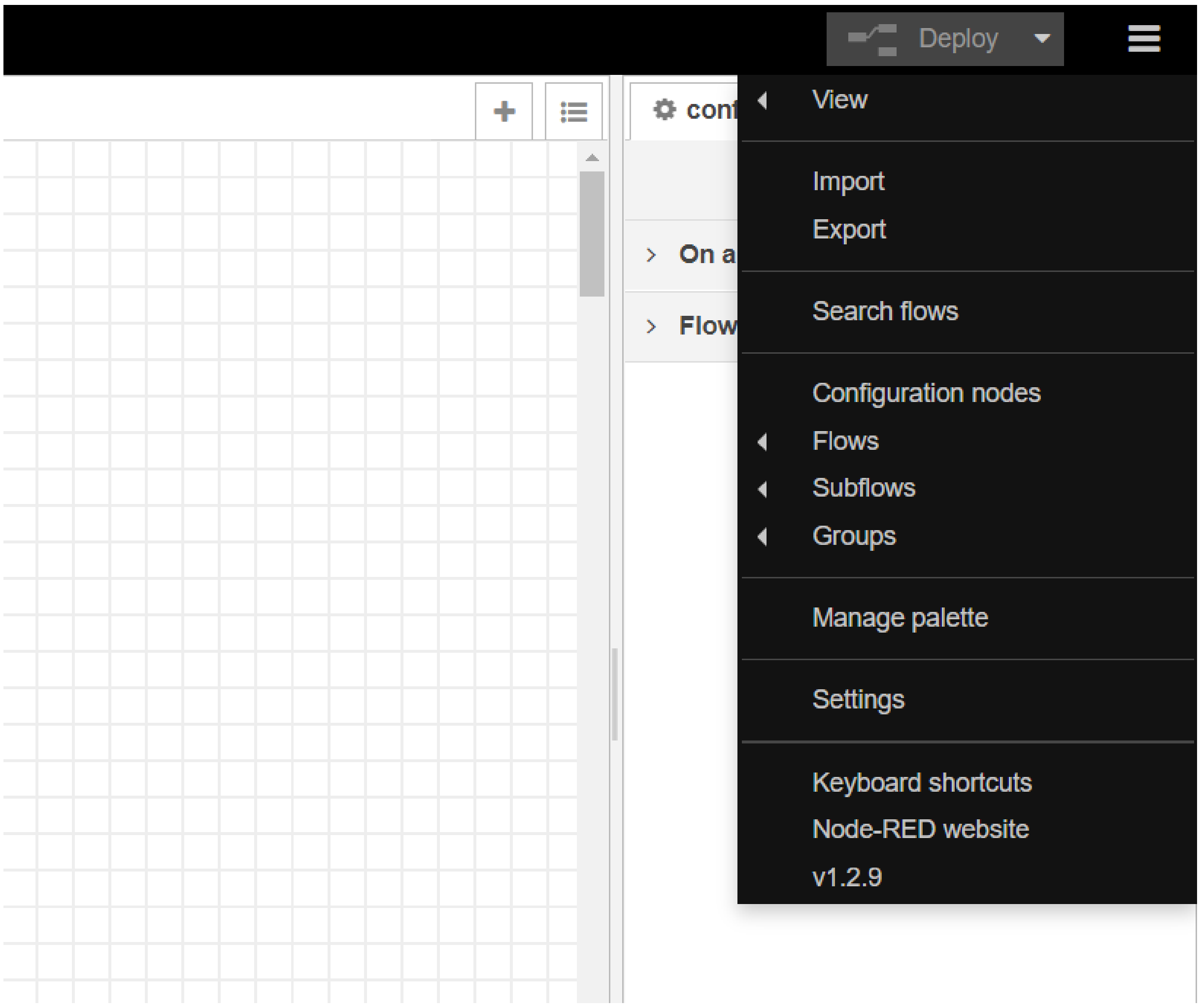
It will give us an IP address to use web-based node-red.



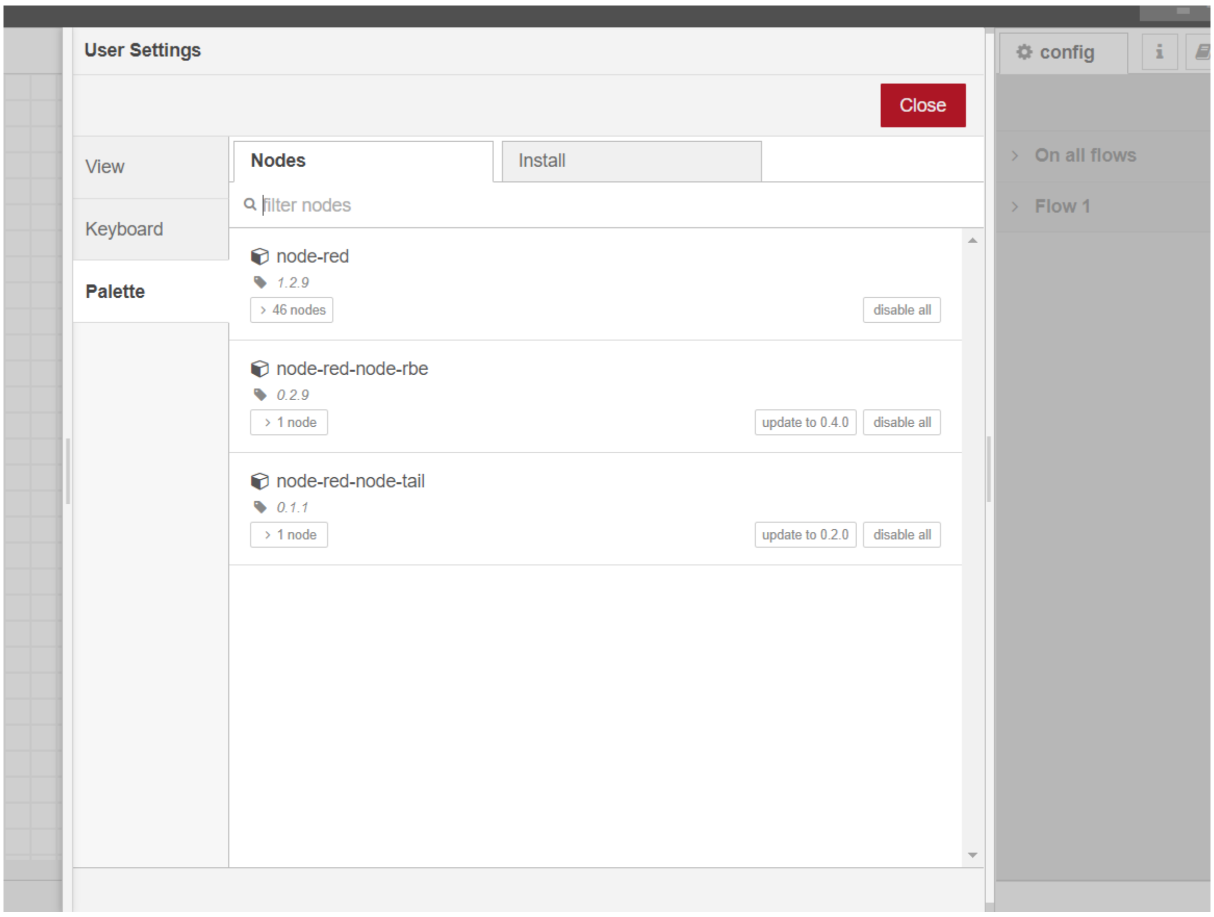
First, we will install broker for MQTT protocol.

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.

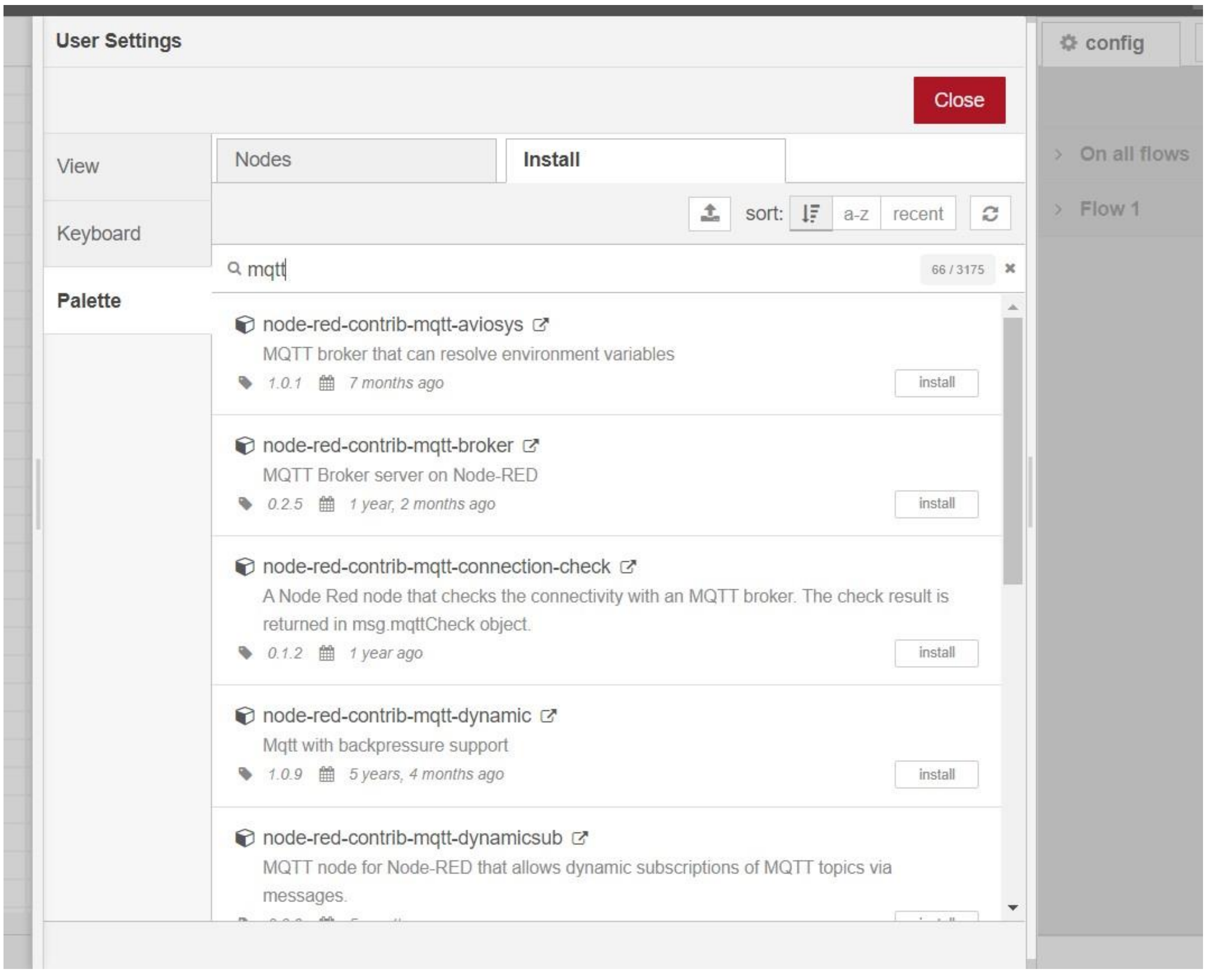


node-red
1.2.9
› 46 nodes disable all

node-red-node-rbe
0.2.9
› 1 node update to 0.4.0 disable all

node-red-node-tail
0.1.1
› 1 node update to 0.2.0 disable all

We go to Install tab and search MQTT.



node-red-contrib-mqtt-aviosys
MQTT broker that can resolve environment variables
1.0.1 7 months ago install

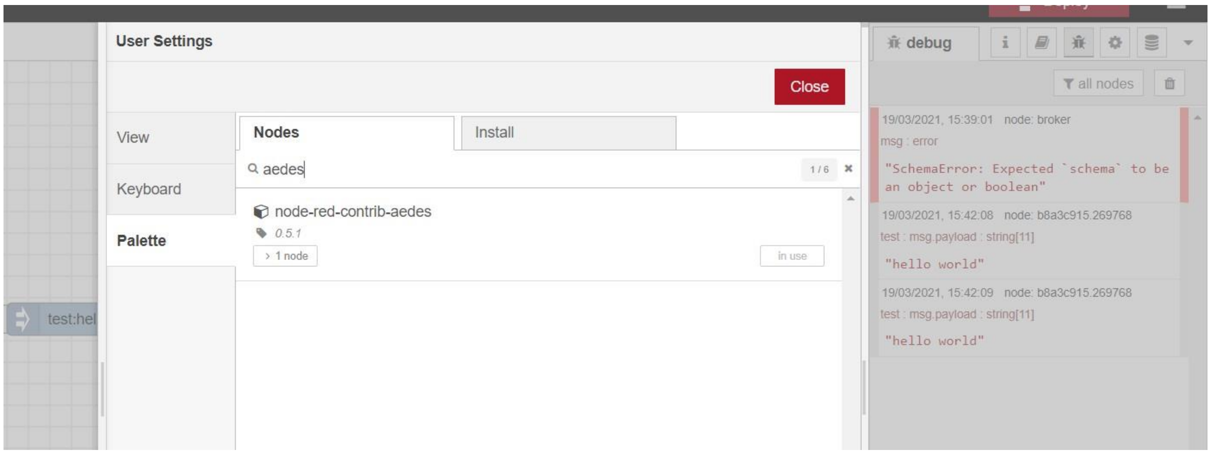
node-red-contrib-mqtt-broker
MQTT Broker server on Node-RED
0.2.5 1 year, 2 months ago install

node-red-contrib-mqtt-connection-check
A Node Red node that checks the connectivity with an MQTT broker. The check result is returned in msg.mqttCheck object.
0.1.2 1 year ago install

node-red-contrib-mqtt-dynamic
Mqtt with backpressure support
1.0.9 5 years, 4 months ago install

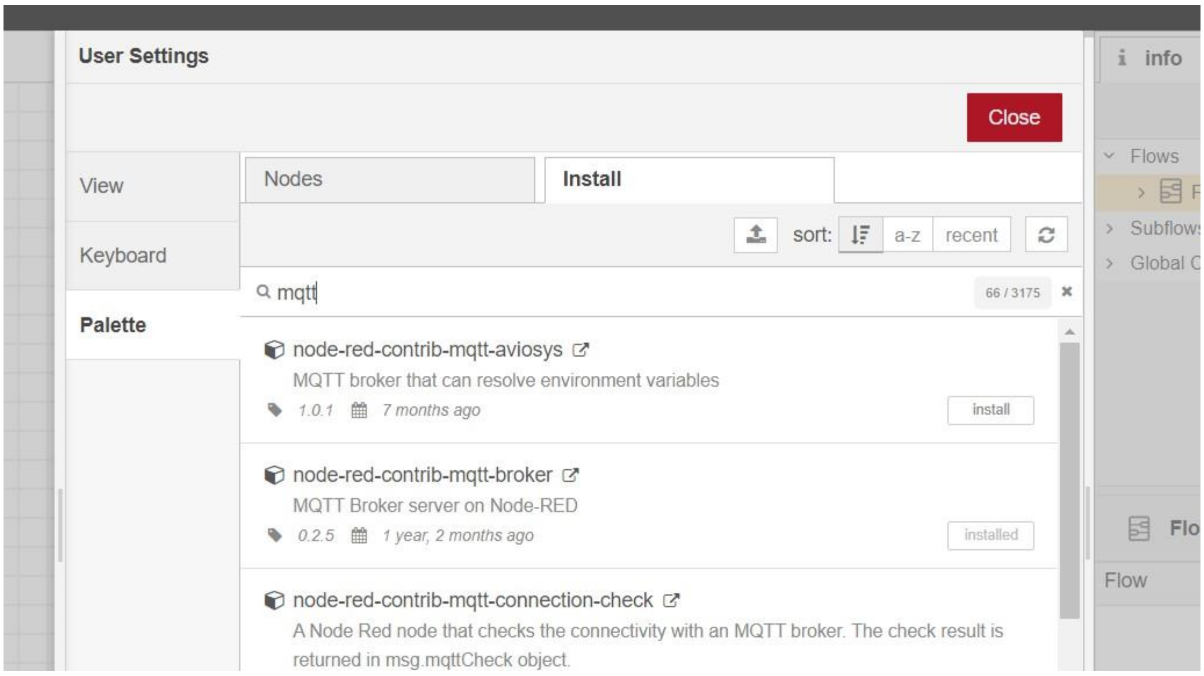
node-red-contrib-mqtt-dynamicsub
MQTT node for Node-RED that allows dynamic subscriptions of MQTT topics via messages.

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



node-red-contrib-aedes
0.5.1
1 node in use

```
19/03/2021, 15:39:01   node: broker
msg : error
"SchemaError: Expected `schema` to be an object or boolean"
19/03/2021, 15:42:08   node: b8a3c915.269768
test : msg.payload : string[11]
"hello world"
19/03/2021, 15:42:09   node: b8a3c915.269768
test : msg.payload : string[11]
"hello world"
```


User Settings

Close

View | Keyboard | Palette

Nodes | Install

sort: a-z recent

mqtt 66 / 3175

node-red-contrib-mqtt-aviosys
MQTT broker that can resolve environment variables
1.0.1 7 months ago install

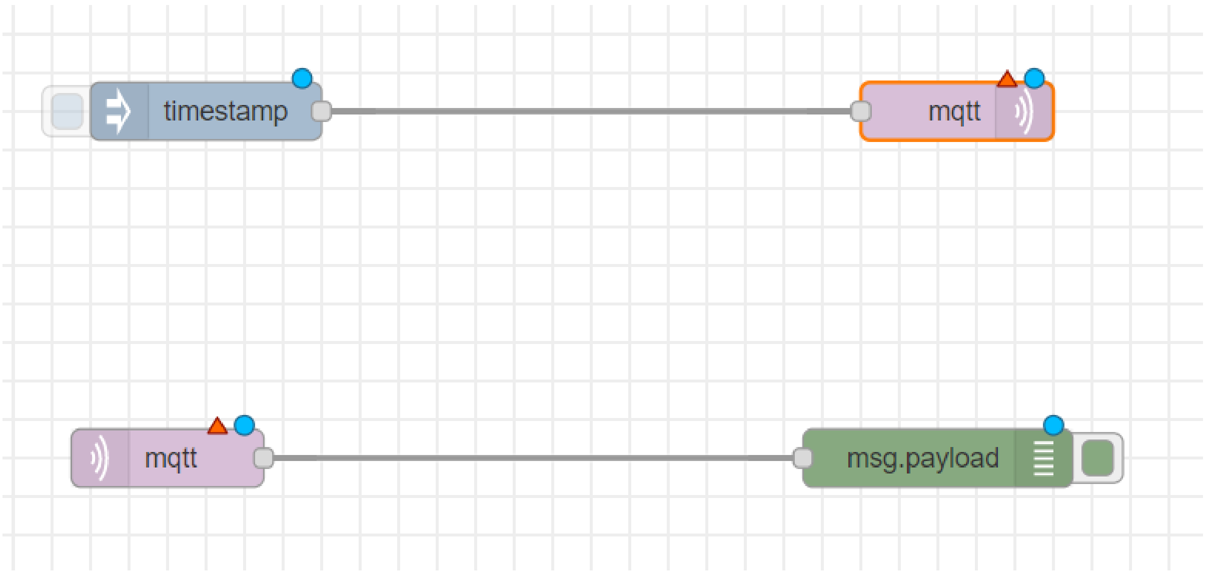
node-red-contrib-mqtt-broker
MQTT Broker server on Node-RED
0.2.5 1 year, 2 months ago installed

node-red-contrib-mqtt-connection-check
A Node Red node that checks the connectivity with an MQTT broker. The check result is returned in msg.mqttCheck object.

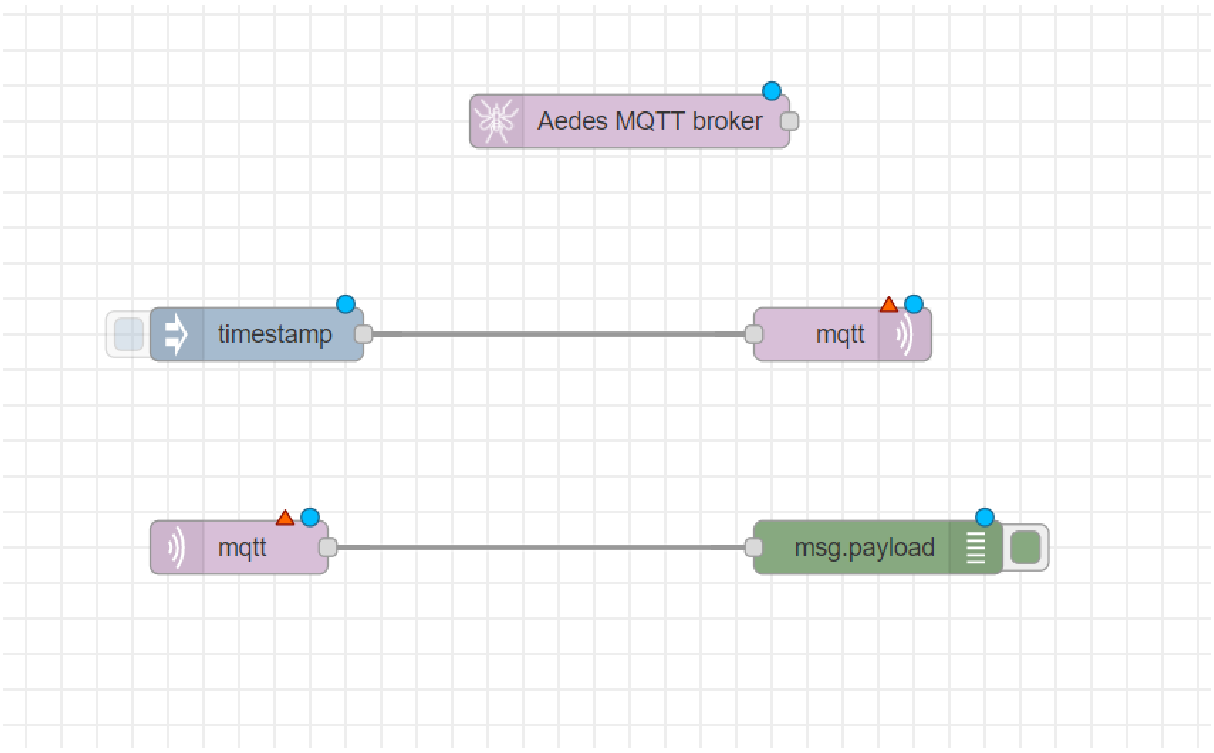
Then we will add 4 nodes.

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

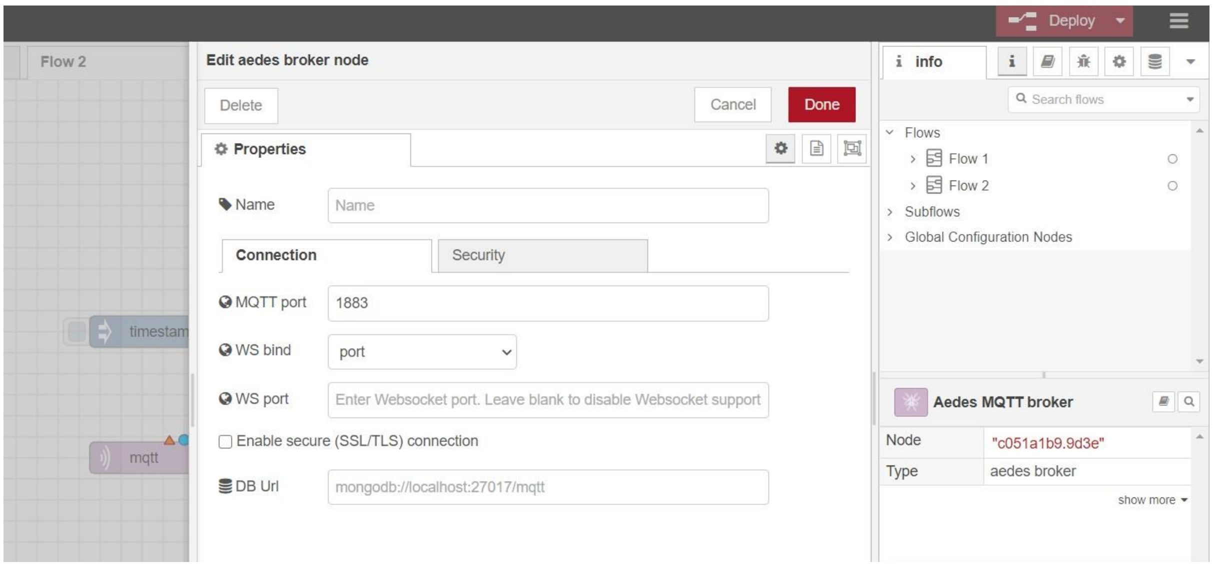
We will connect timestamp node to MQTT out and debug node to MQTT in.



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



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



Edit aedes broker node

Name: Name

Connection | Security

MQTT port: 1883

WS bind: port

WS port: Enter Websocket port. Leave blank to disable Websocket support

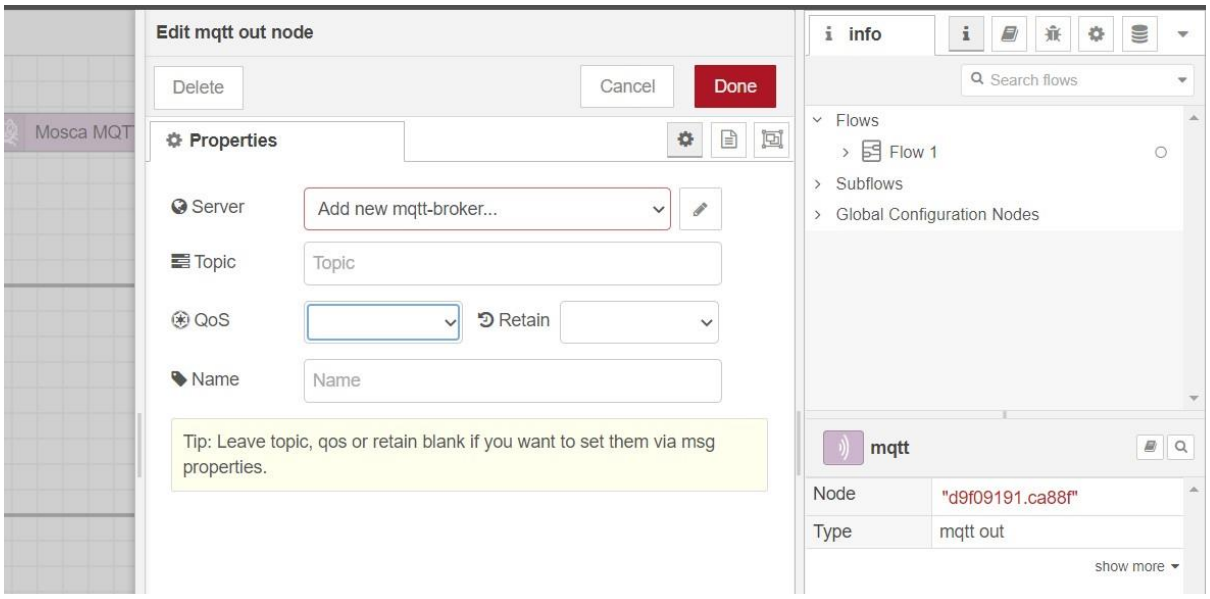
☐ Enable secure (SSL/TLS) connection

DB Url: mongodb://localhost:27017/mqtt

Aedes MQTT broker

Node	"c051a1b9.9d3e"
Type	aedes broker

We will first configure MQTT out node.


Edit mqtt out node

Server: Add new mqtt-broker...
Topic: Topic
QoS:
Retain:
Name: Name

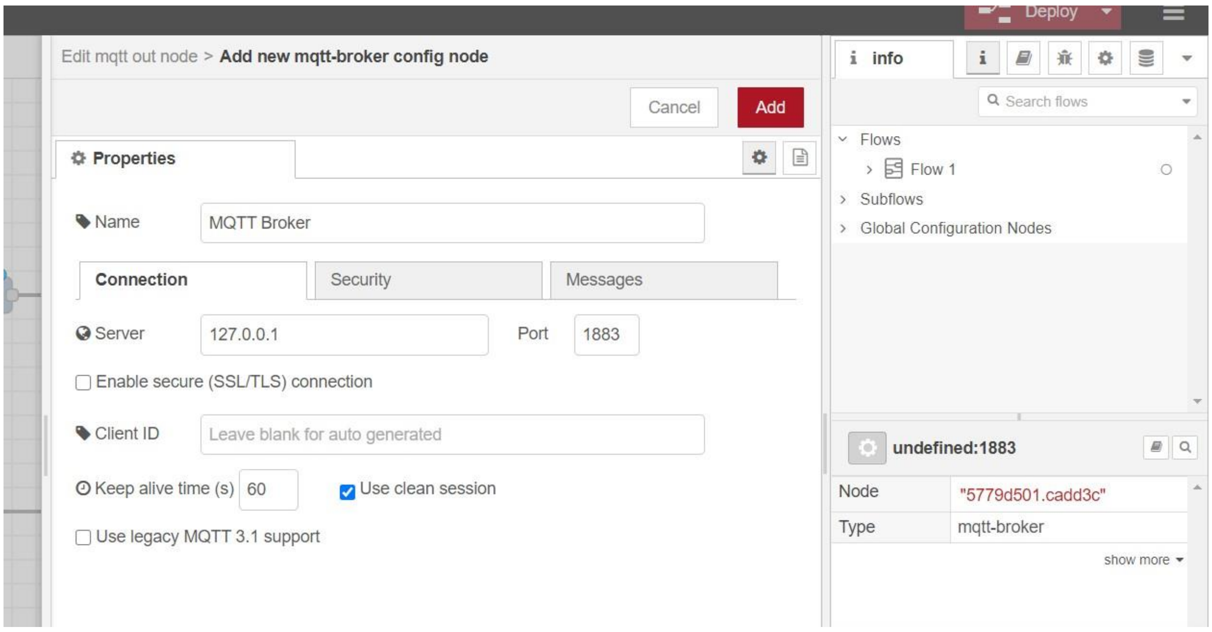
Tip: Leave topic, qos or retain blank if you want to set them via msg properties.

mqtt
Node: "d9f09191.ca88f"
Type: mqtt out

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.


Edit mqtt out node > Add new mqtt-broker config node

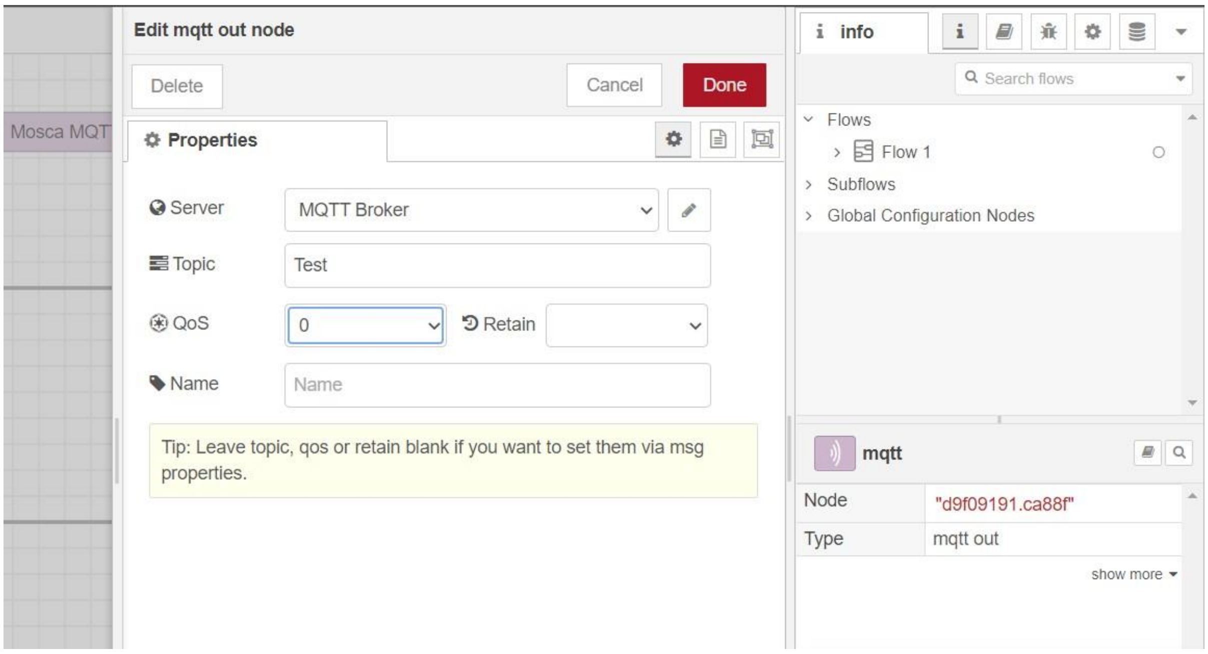
Name: MQTT Broker

Connection | Security | Messages

Server: 127.0.0.1 Port: 1883
☐ Enable secure (SSL/TLS) connection
Client ID: Leave blank for auto generated
Keep alive time (s): 60 ☑ Use clean session
☐ Use legacy MQTT 3.1 support

undefined:1883
Node: "5779d501.cadd3c"
Type: mqtt-broker

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


Edit mqtt out node

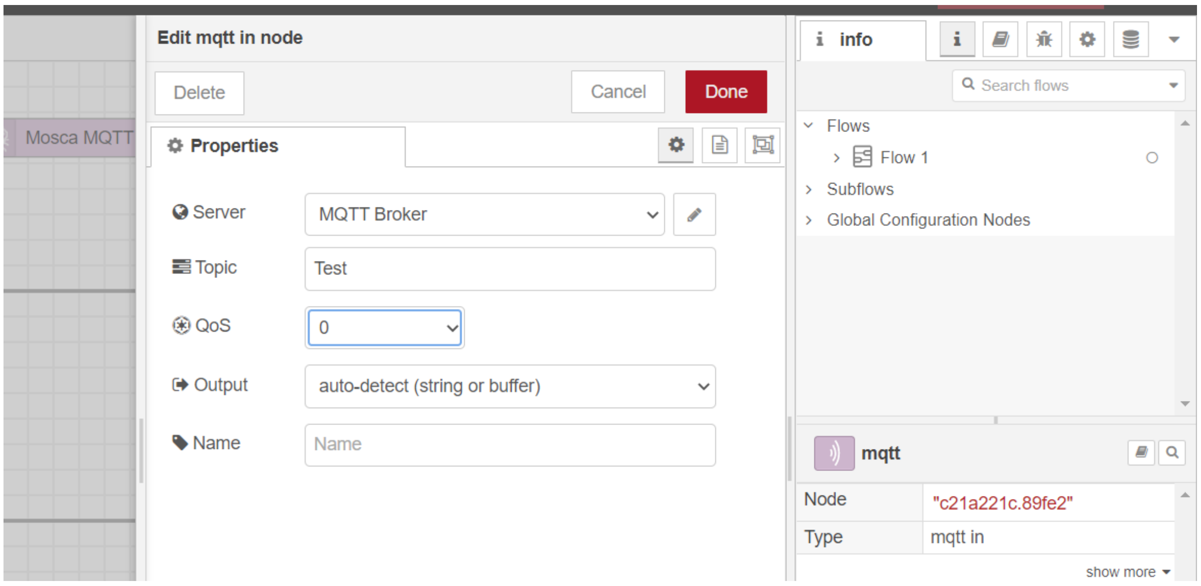
Server: MQTT Broker
Topic: Test
QoS: 0
Retain:
Name: Name

Tip: Leave topic, qos or retain blank if you want to set them via msg properties.

mqtt

Node	"d9f09191.ca88f"
Type	mqtt out

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

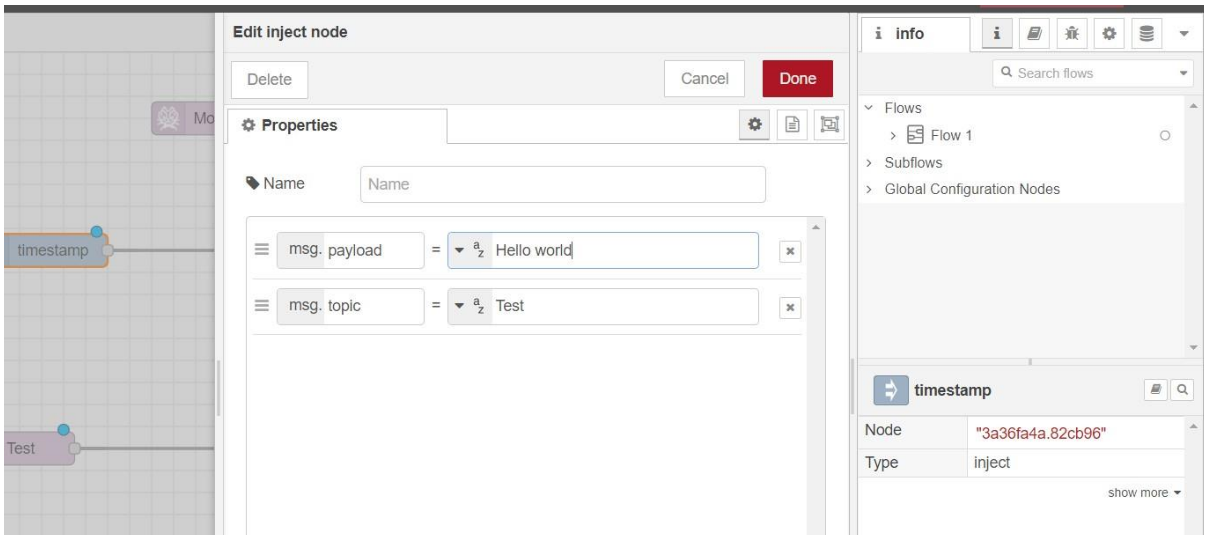

Edit mqtt in node

Server: MQTT Broker
Topic: Test
QoS: 0
Output: auto-detect (string or buffer)
Name: Name

mqtt

Node	"c21a221c.89fe2"
Type	mqtt in

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.



Name: Name

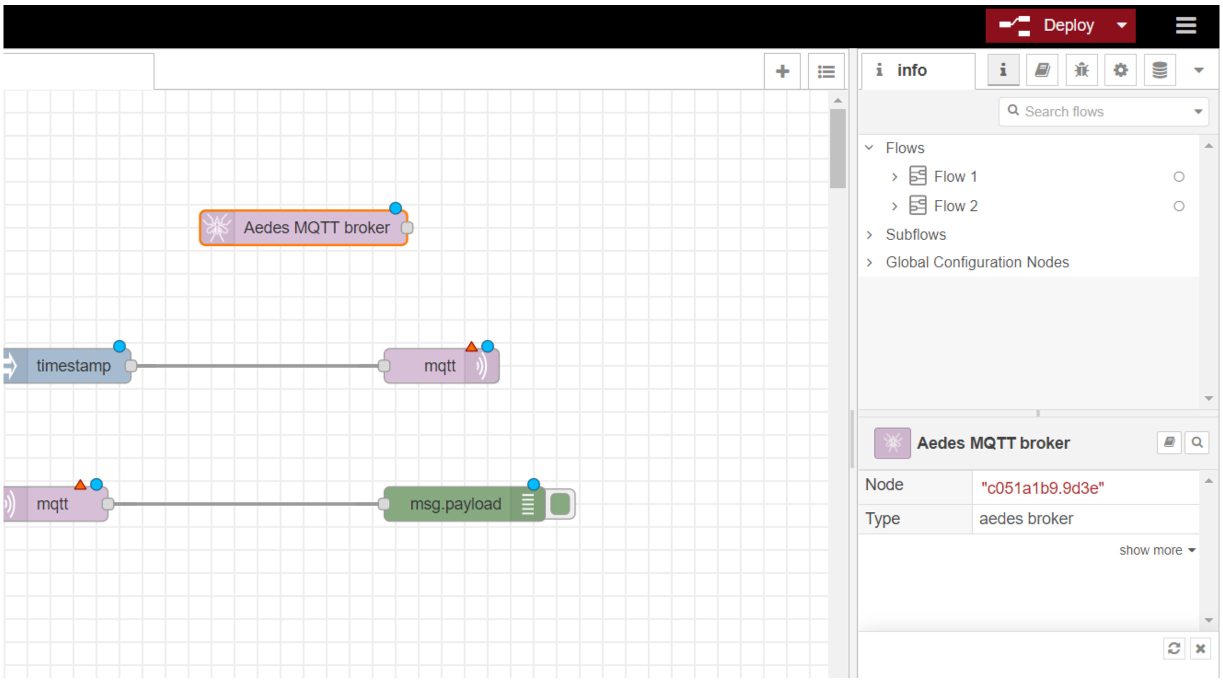
msg. payload = Hello world

msg. topic = Test

timestamp

Node	"3a36fa4a.82cb96"
Type	inject

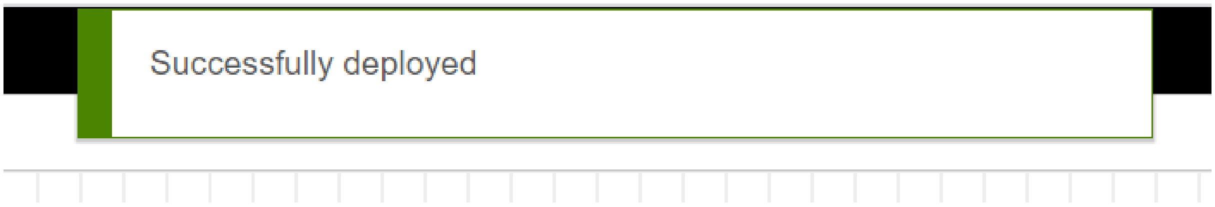
Now we can deploy our model by "Deploy" button on top right corner.



Aedes MQTT broker

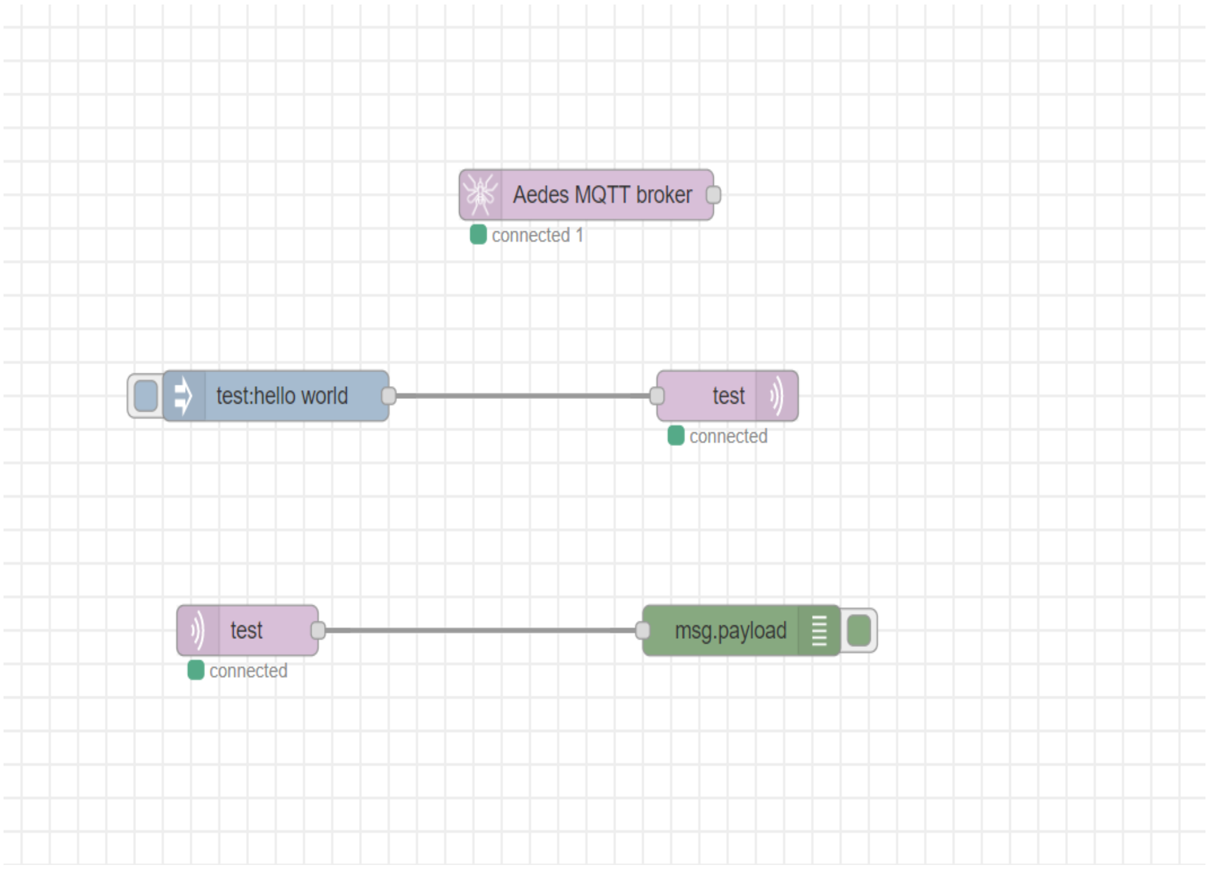
Node	"c051a1b9.9d3e"
Type	aedes broker

We can see "Successfully deployed" message.

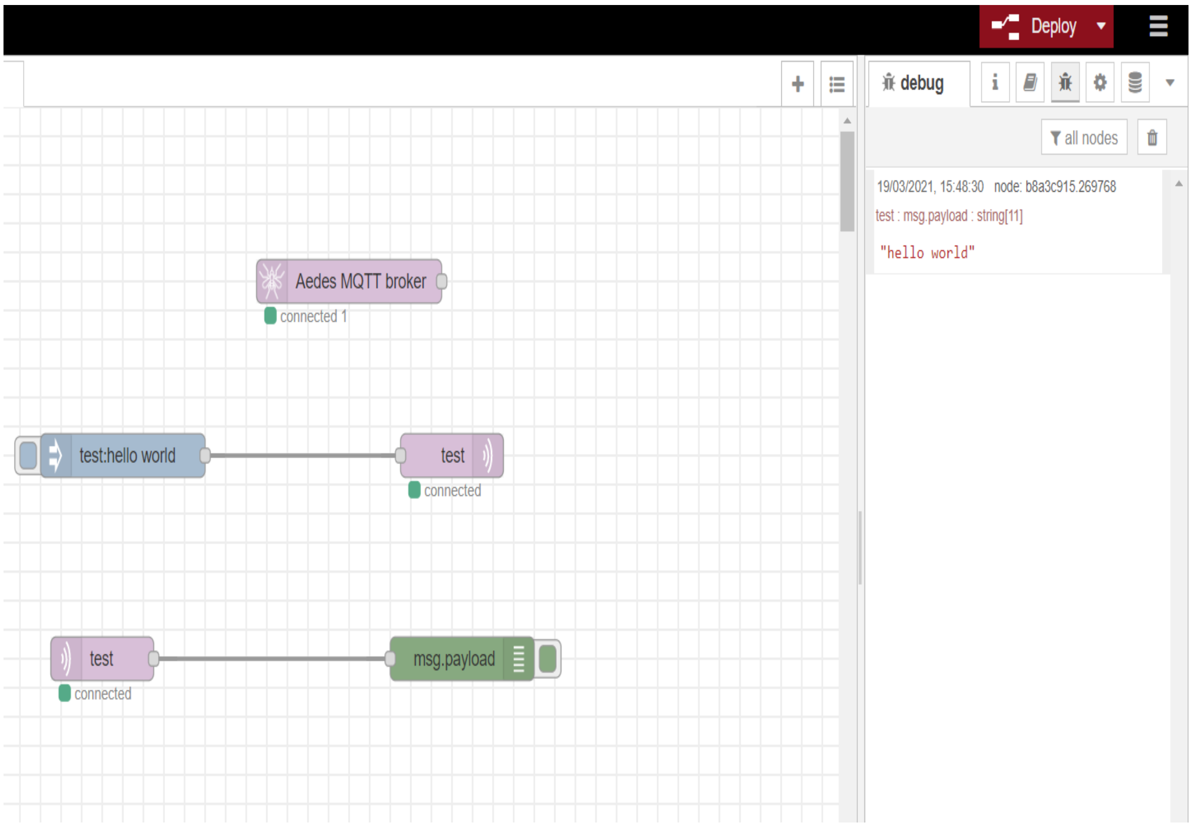


Successfully deployed

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

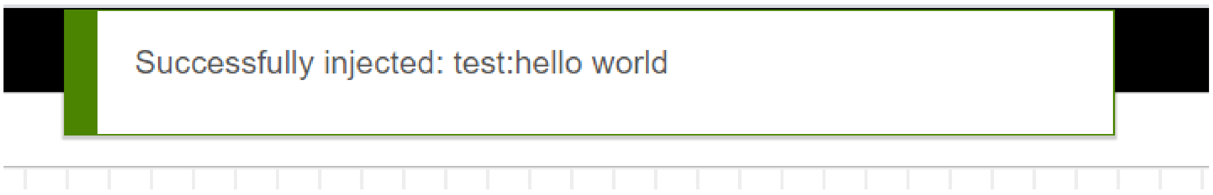


In debug console, which can be opened from right panel, we can see the output received by debug node after clicking inject node to send the text.



```
19/03/2021, 15:48:30   node: b8a3c915.269768
test : msg.payload : string[11]
"hello world"
```

We can see the status on top too.



Successfully injected: test:hello world

CONCLUSION:

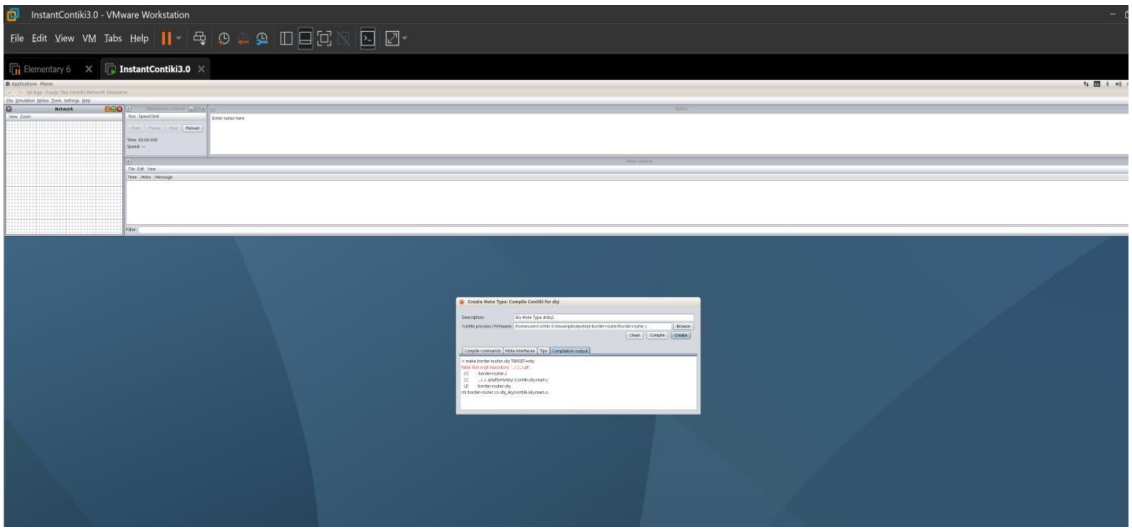
In this practical, we learned about Node red and MQTT. We implemented the MQTT connection using Node red.

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.



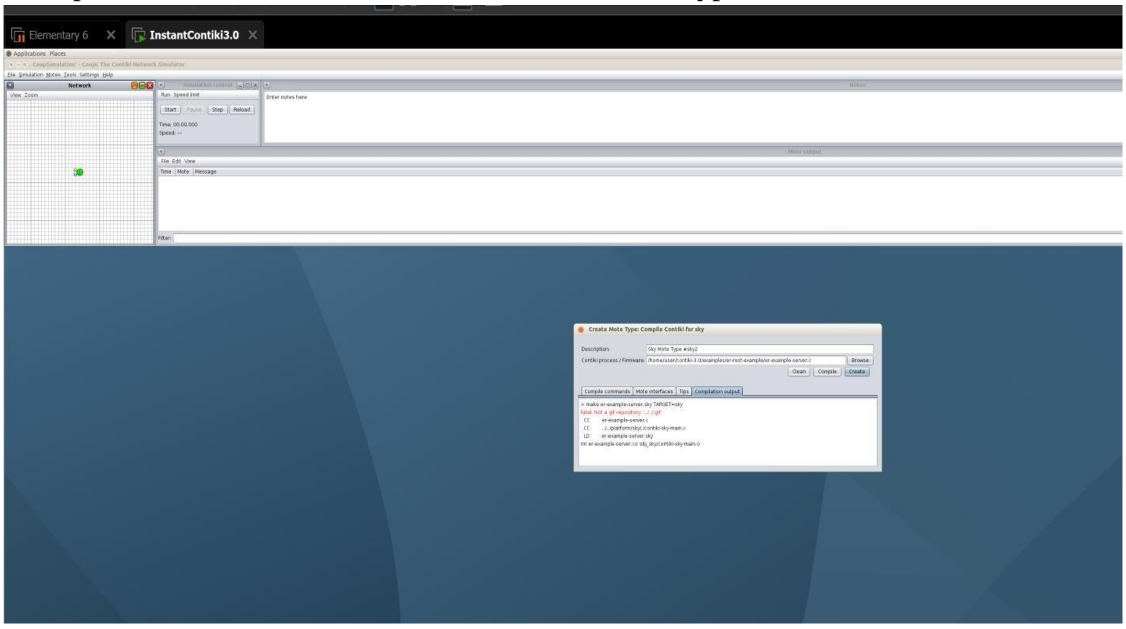
Create Mote Type: Compile Contiki for sky

Description: Sky Mote Type #sky1

Contiki process / Firmware: /home/user/contiki-3.0/examples/ipv6/rpl-border-router/border-router.c

```
> make border-router.sky TARGET=sky
fatal: Not a git repository: '../../../.git'
  CC        border-router.c
  CC        ../../../platform/sky/./contiki-sky-main.c
  LD        border-router.sky
rm border-router.co obj_sky/contiki-sky-main.o
```

2. 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.



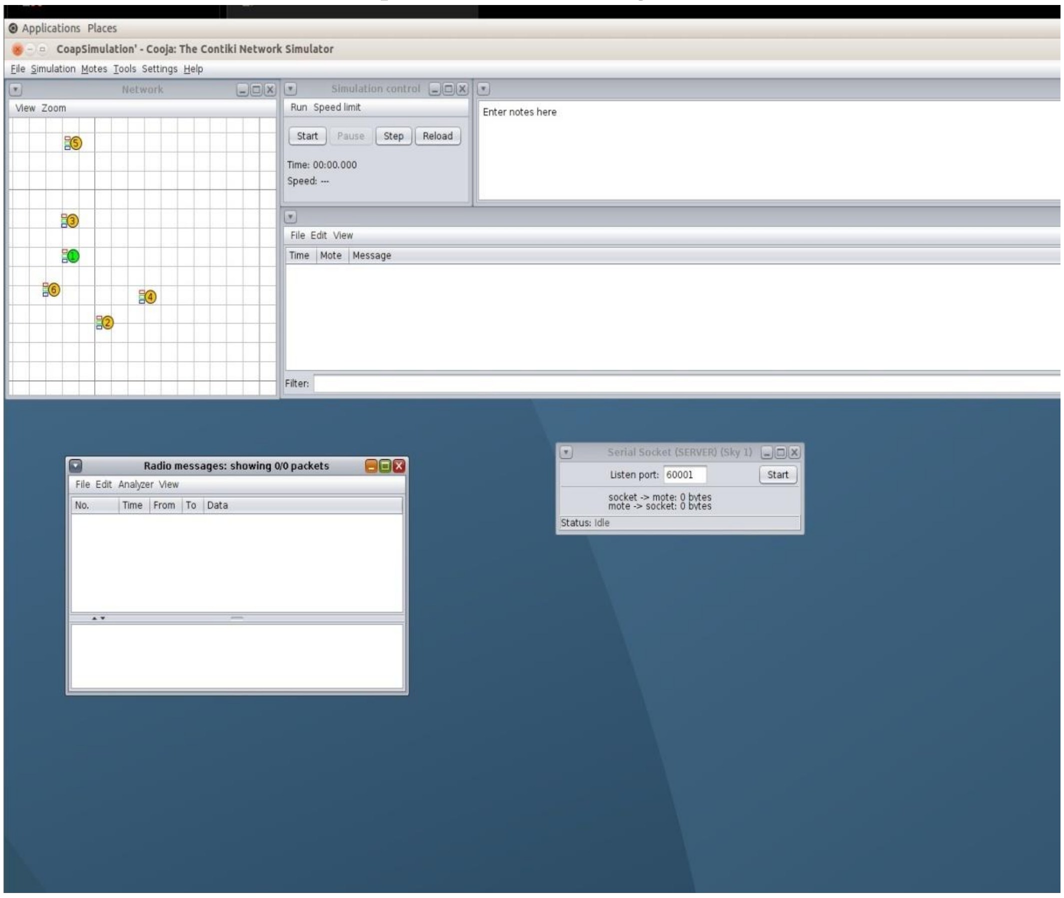
Create Mote Type: Compile Contiki for sky

Description: Sky Mote Type #sky2

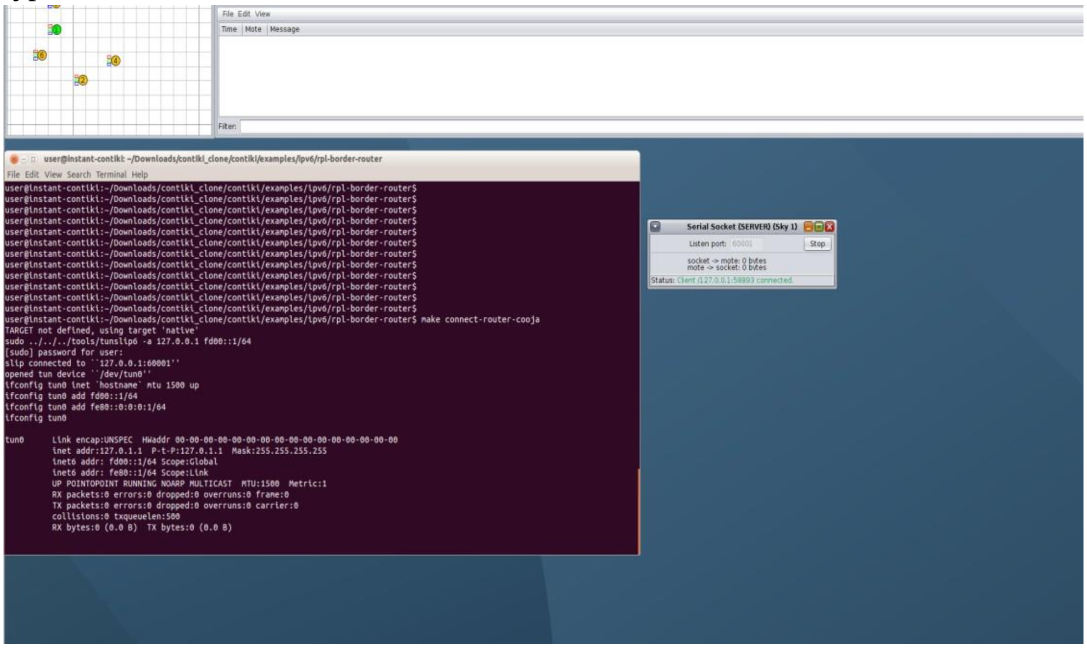
Contiki process / Firmware: /home/user/contiki-3.0/examples/er-rest-example/er-example-server.c

```
> make er-example-server.sky TARGET=sky
fatal: Not a git repository: '../../.git'
  CC        er-example-server.c
  CC        ../../platform/sky/./contiki-sky-main.c
  LD        er-example-server.sky
rm er-example-server.co obj_sky/contiki-sky-main.o
```

3. 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.

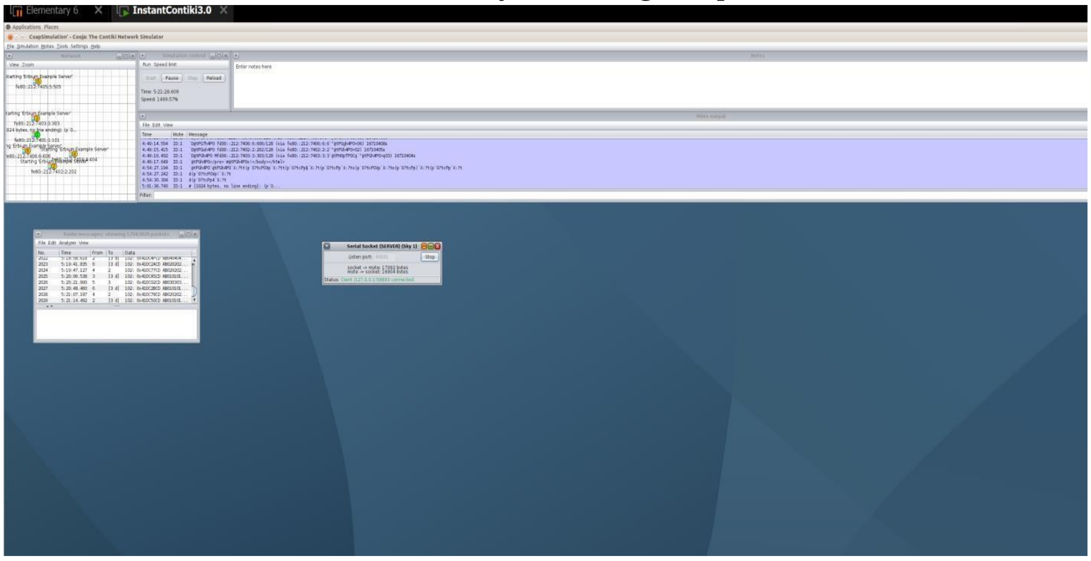


4. After starting the listening port we have to go to the example directory of router and type the command make connect router command to start the connection.


File Edit View
Time Mote Message
Filter:
user@instant-contiki: ~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router
File Edit View Search Terminal Help
Serial Socket (SERVER) (Sky 1)
Listen port: 60001
Stop
socket -> mote: 0 bytes
mote -> socket: 0 bytes
Status: Client /127.0.0.1:58893 connected.

```
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$ make connect-router-cooja
TARGET not defined, using target 'native'
sudo ../../../tools/tunslip6 -a 127.0.0.1 fd00::1/64
[sudo] password for user:
slip connected to ``127.0.0.1:60001''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::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: fd00::1/64 Scope:Global
          inet6 addr: fe80::1/64 Scope:Link
          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)
```

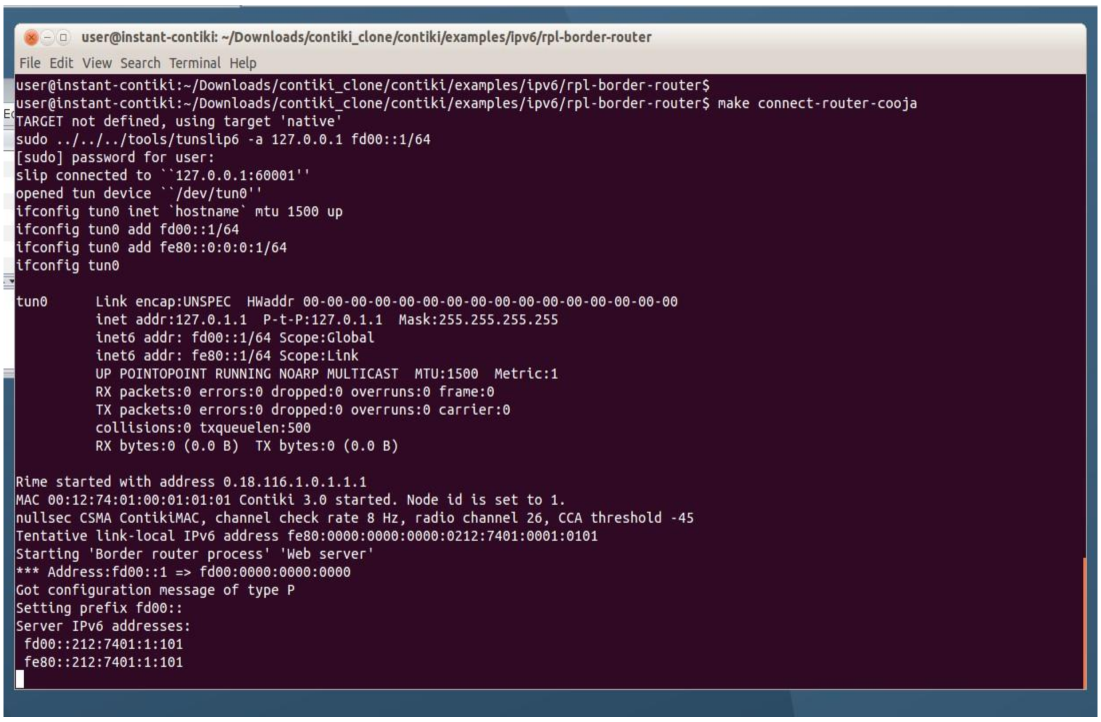
5. Now we will begin the simulation and the traffic will begin instantly and we can also observe all the motes will automatically have assigned ip addresses.



```
Time          Mote   Message
4:49:14.554   ID:1   [illegible] fd00::212:7406:6:606/128 (via fe80::212:7406:6:6 [illegible]) 16710408
4:49:15.415   ID:1   [illegible] fd00::212:7402:2:202/128 (via fe80::212:7402:2:2 [illegible]) 16710405
4:49:16.492   ID:1   [illegible] fd00::212:7403:3:303/128 (via fe80::212:7403:3:3 [illegible]) 16710404
4:49:17.649   ID:1   [illegible]</pre>[illegible]</body></html>
4:54:27.104   ID:1   [illegible]
4:54:27.242   ID:1   [illegible]
4:54:30.304   ID:1   [illegible]
5:01:36.740   ID:1   # [1024 bytes, no line ending]: (p`0...
```

```
No.    Time          From   To      Data
2023   5:19:41.835   6      [3 d]   102: 0x41DC24CD AB020202...
2024   5:19:47.127   4      2       102: 0x41DC77CD AB020202...
2025   5:20:06.538   3      [3 d]   102: 0x41DC85CD AB010101...
2026   5:20:21.900   5      3       102: 0x41DC02CD AB030303...
2027   5:20:48.482   6      [3 d]   102: 0x41DC28CD AB010101...
2028   5:21:07.197   4      2       102: 0x41DC79CD AB020202...
2029   5:21:14.492   2      [3 d]   102: 0x41DC50CD AB010101...
```

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

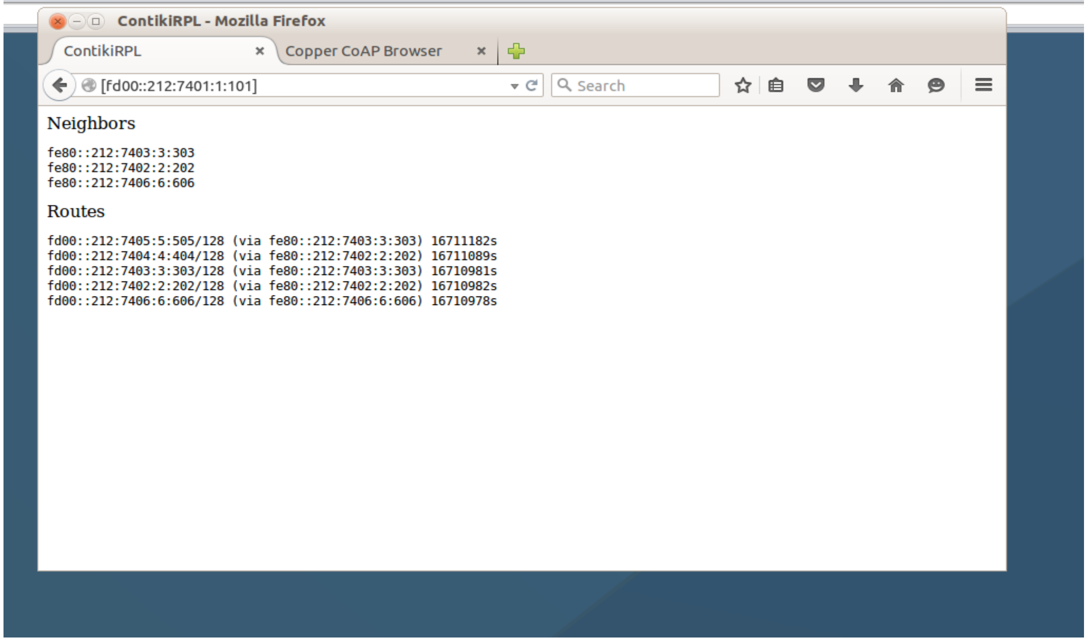


```
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$
user@instant-contiki:~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router$ make connect-router-cooja
TARGET not defined, using target 'native'
sudo ../../../tools/tunslip6 -a 127.0.0.1 fd00::1/64
[sudo] password for user:
slip connected to ``127.0.0.1:60001''
opened tun device ``/dev/tun0''
ifconfig tun0 inet `hostname` mtu 1500 up
ifconfig tun0 add fd00::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: fd00::1/64 Scope:Global
          inet6 addr: fe80::1/64 Scope:Link
          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)

Rime started with address 0.18.116.1.0.1.1.1
MAC 00:12:74:01:00:01:01:01 Contiki 3.0 started. Node id is set to 1.
nullsec CSMA ContikiMAC, channel check rate 8 Hz, radio channel 26, CCA threshold -45
Tentative link-local IPv6 address fe80:0000:0000:0000:0212:7401:0001:0101
Starting 'Border router process' 'Web server'
*** Address:fd00::1 => fd00:0000:0000:0000
Got configuration message of type P
Setting prefix fd00::
Server IPv6 addresses:
 fd00::212:7401:1:101
 fe80::212:7401:1:101
```

user@instant-contiki: ~/Downloads/contiki_clone/contiki/examples/ipv6/rpl-border-router
File Edit View Search Terminal Help

7. 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.



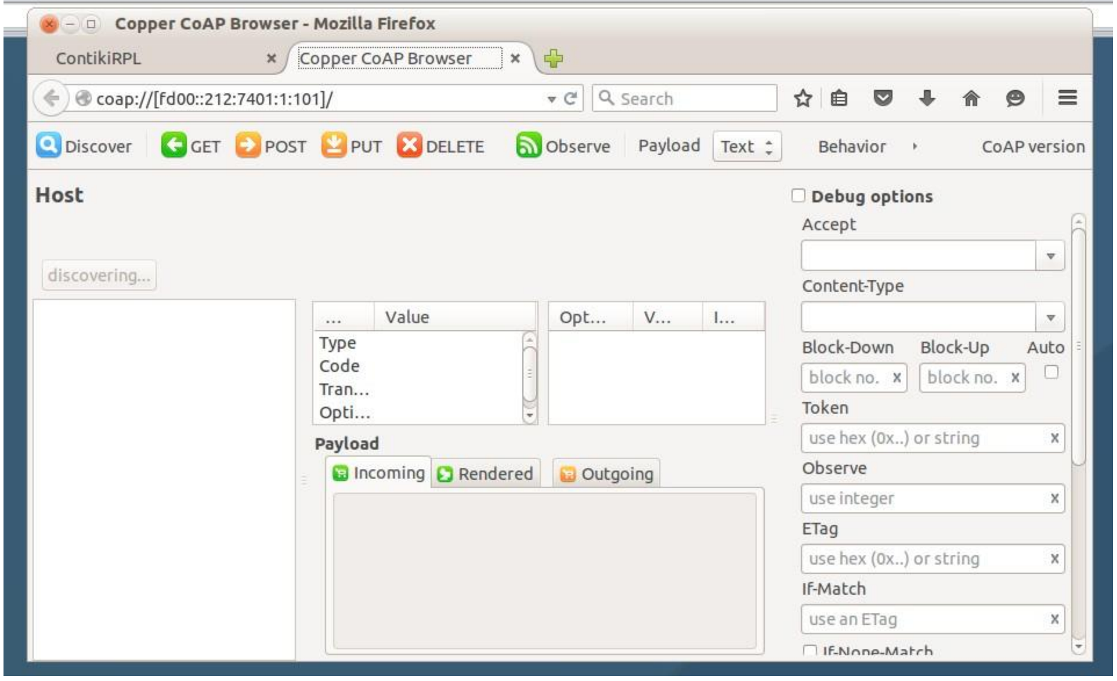
Neighbors

```
fe80::212:7403:3:303
fe80::212:7402:2:202
fe80::212:7406:6:606
```

Routes

```
fd00::212:7405:5:505/128 (via fe80::212:7403:3:303) 16711182s
fd00::212:7404:4:404/128 (via fe80::212:7402:2:202) 16711089s
fd00::212:7403:3:303/128 (via fe80::212:7403:3:303) 16710981s
fd00::212:7402:2:202/128 (via fe80::212:7402:2:202) 16710982s
fd00::212:7406:6:606/128 (via fe80::212:7406:6:606) 16710978s
```

8. 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:

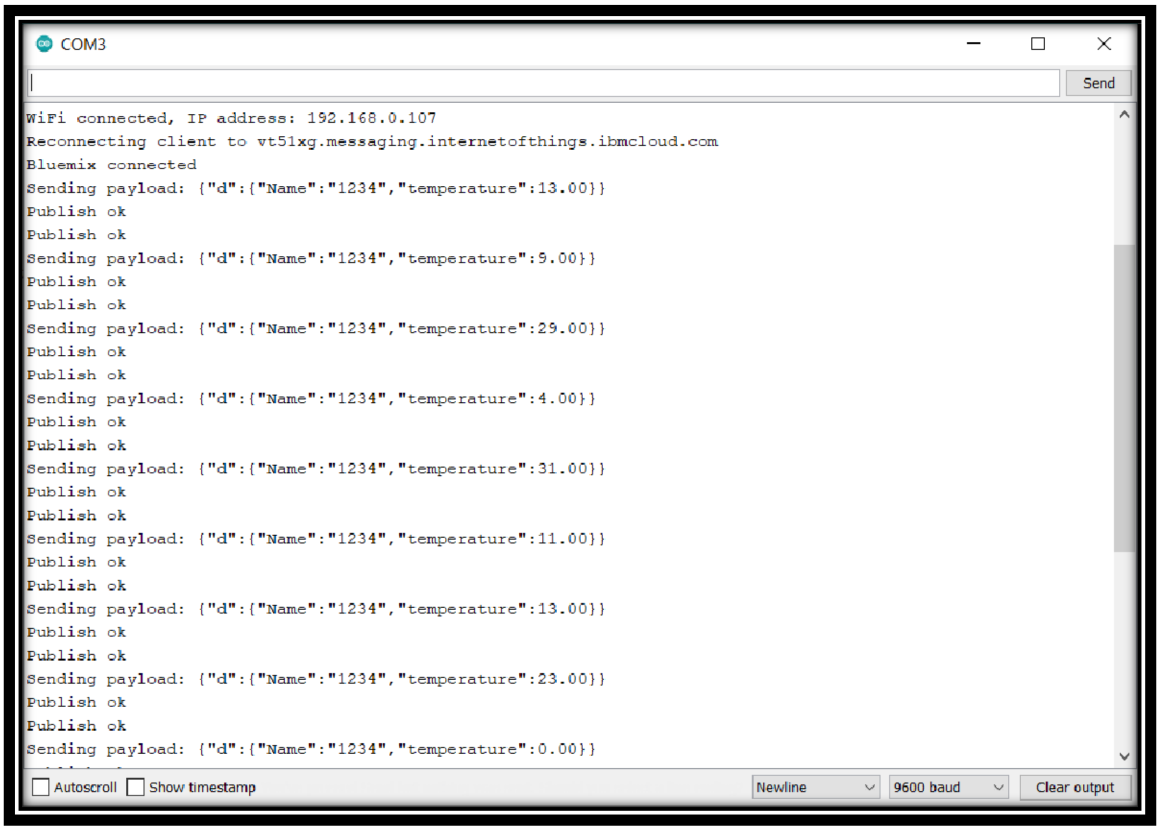
- Bluemix:
    - Bluemix is the IBM open cloud platform that provides mobile and web developers access to IBM software for integration security transaction and other key functions as well as software from business partners.
    - Built on Cloud Foundry open source technology Bluemix provides pre-built 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:

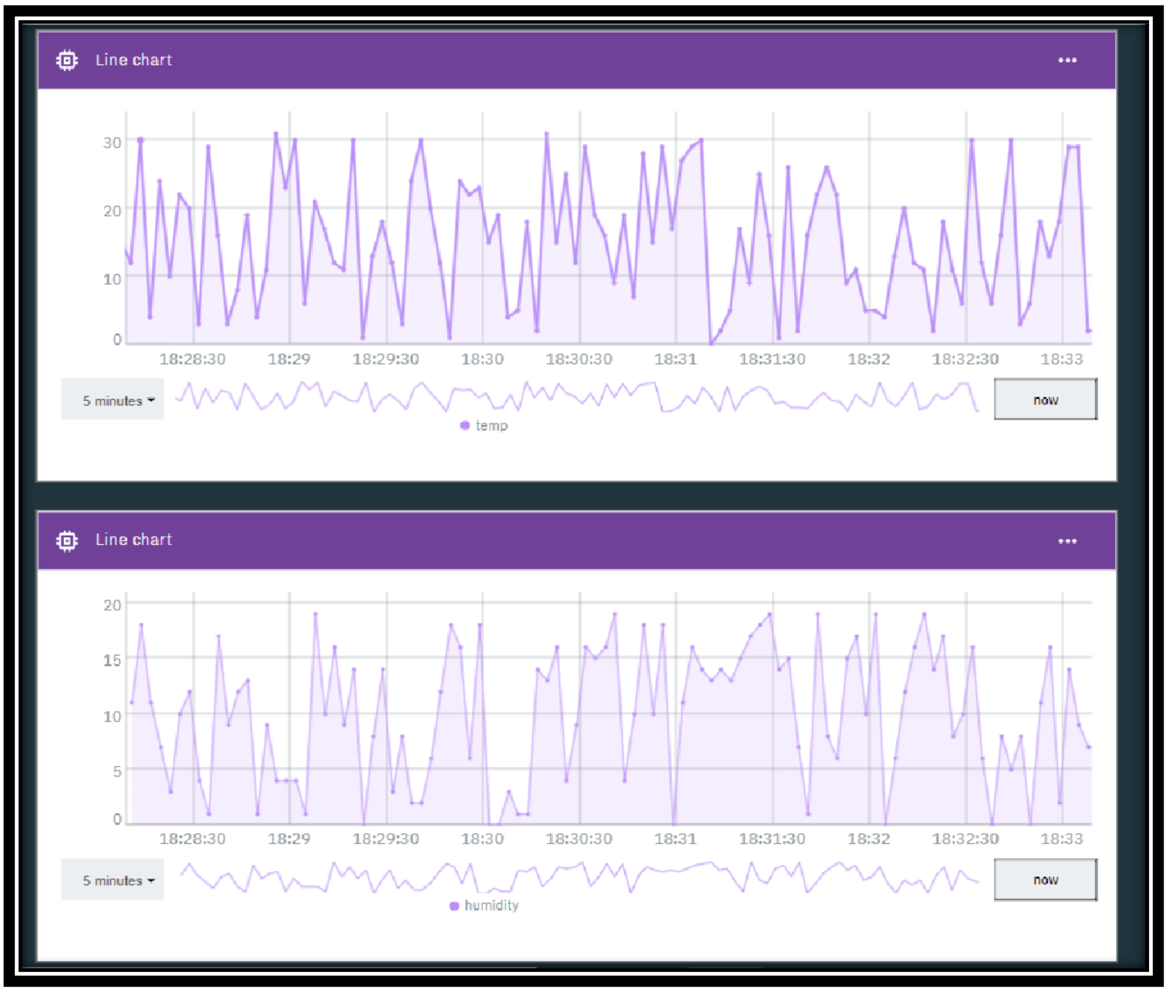
Bluemix:

```
#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");
        }
    }
}
```



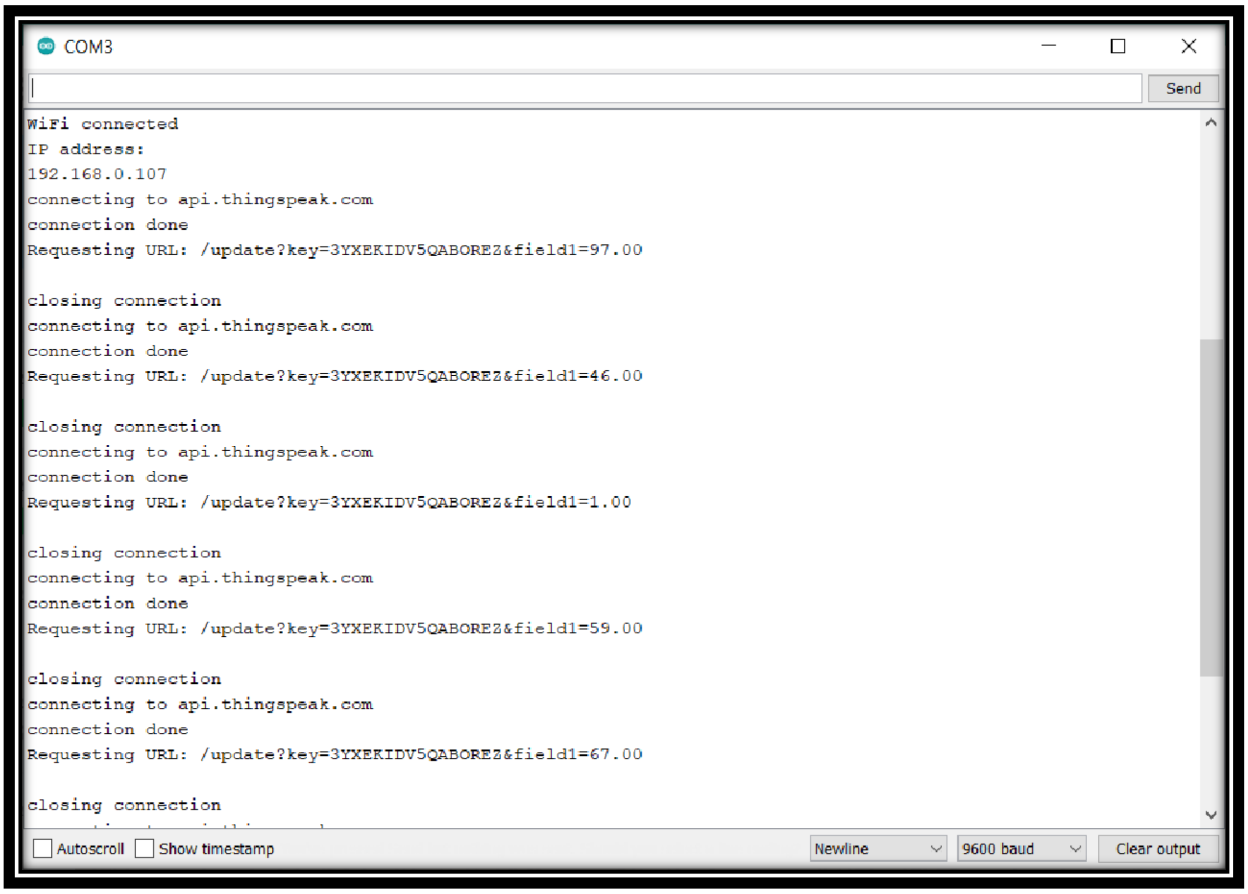
```
WiFi connected, IP address: 192.168.0.107
Reconnecting client to vt51xg.messaging.internetofthings.ibmcloud.com
Bluemix connected
Sending payload: {"d":{"Name":"1234","temperature":13.00}}
Publish ok
Publish ok
Sending payload: {"d":{"Name":"1234","temperature":9.00}}
Publish ok
Publish ok
Sending 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}}
```



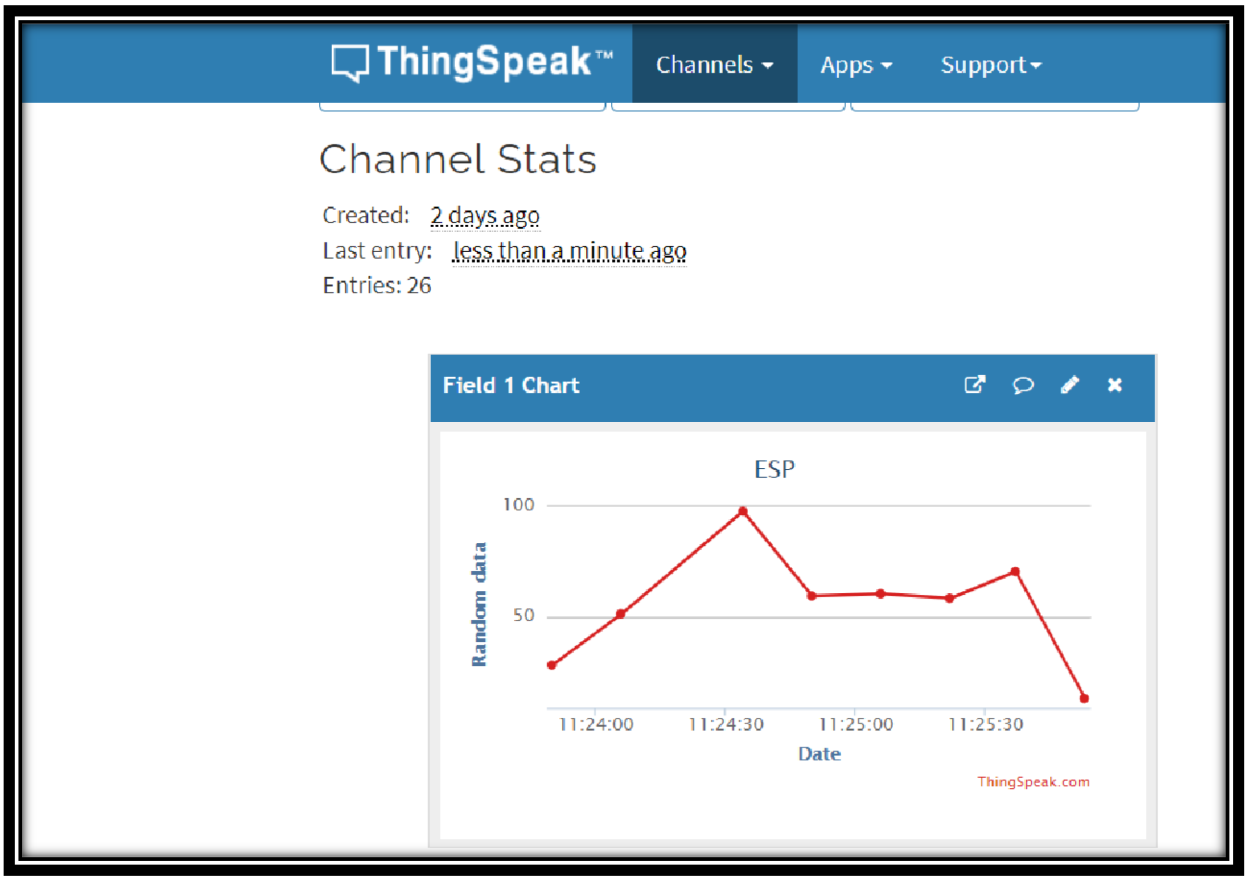
Thingspeak:

```
#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");
}
```



```
WiFi connected
IP address:
192.168.0.107
connecting to api.thingspeak.com
connection done
Requesting URL: /update?key=3YXEKIDV5QABOREZ&field1=97.00

closing connection
connecting to api.thingspeak.com
connection done
Requesting URL: /update?key=3YXEKIDV5QABOREZ&field1=46.00

closing connection
connecting to api.thingspeak.com
connection done
Requesting URL: /update?key=3YXEKIDV5QABOREZ&field1=1.00

closing connection
connecting to api.thingspeak.com
connection done
Requesting URL: /update?key=3YXEKIDV5QABOREZ&field1=59.00

closing connection
connecting to api.thingspeak.com
connection done
Requesting URL: /update?key=3YXEKIDV5QABOREZ&field1=67.00

closing connection
```



Channel Stats

Created: 2 days ago

Last entry: less than a minute ago

Entries: 26

CONCLUSION:

In this practical, we learned about IBM bluemix and Thingspeak platforms. We uploaded datato 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.

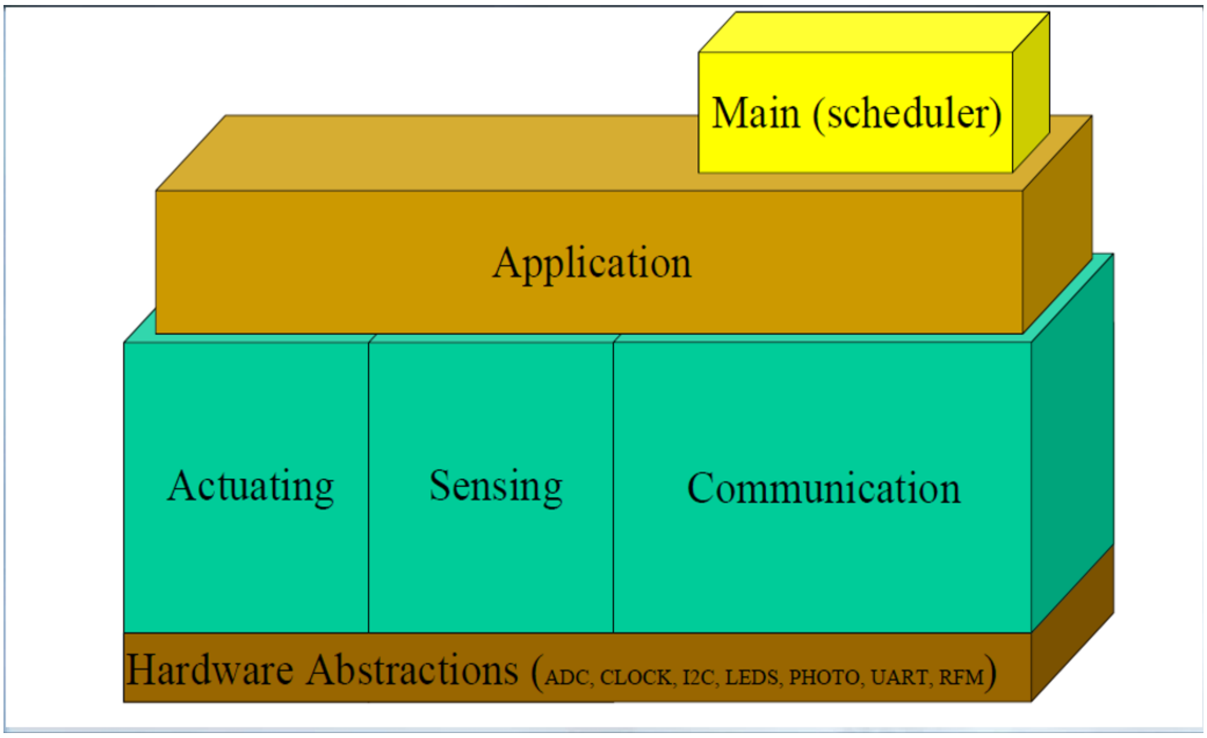
The need for a design like TinyOS is mandatory in applications such as smart transit and smart factories. Because of thousands of sensors, it is important to have a very small memory footprint to reduce power requirements.

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.

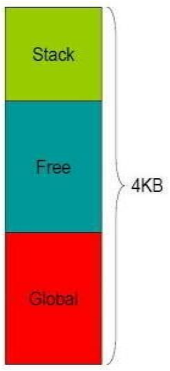
- **Environmental monitoring:**
  - Since each TinyOS system can be embedded in a small sensor, they are useful in monitoring air pollution, forest fires, and natural disaster prevention.

- **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.


Main (scheduler)
Application
Actuating
Sensing
Communication
Hardware Abstractions (ADC, CLOCK, I2C, LEDS, PHOTO, UART, RFM)

<u>TinyOS models:</u>

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

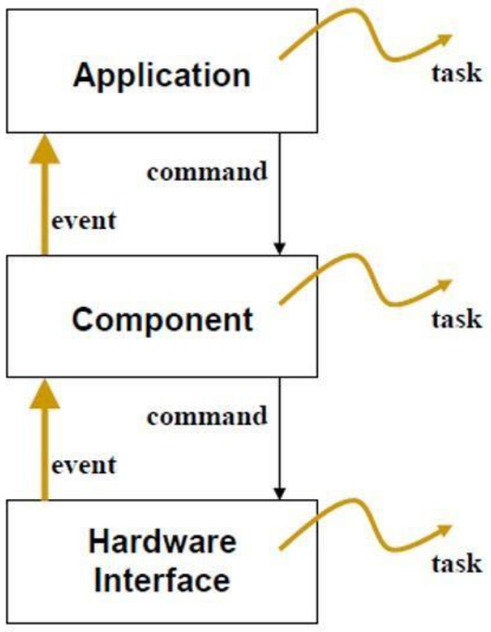


2. Thread model:
   - **Power-Aware Two-levels Scheduling**

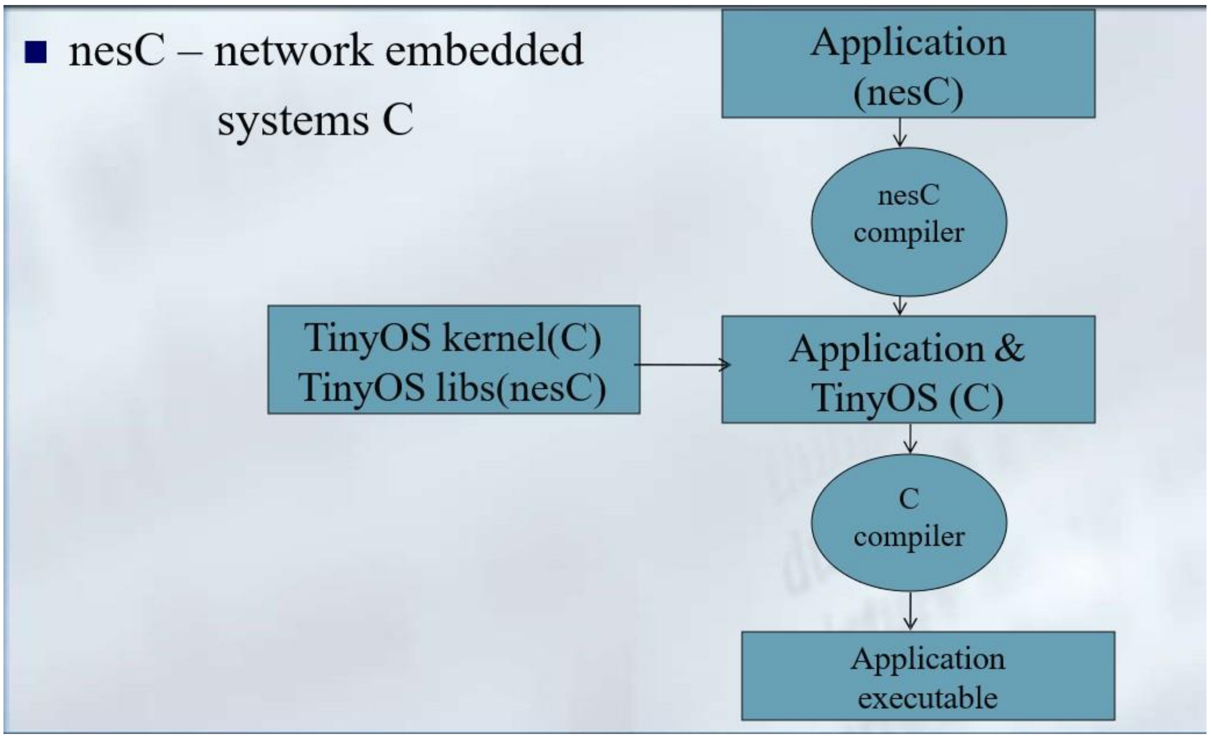
        - Long running tasks and interrupt events
        - Sleep unless tasks in queue, wakeup on event
    - **Tasks**
        - Time-flexible, background jobs
        - Atomic with respect to other tasks
        - Can be preempted by events
    - **Events**
        - Time-critical, shorter duration
        - 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.

<u>TinyOS basic constructs:</u>



nesC language:



- nesC – network embedded systems C

- 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.