A Mini Project Report on

IOT NODE NETWORK SOLUTION USING CONTIKI TOOL

Submitted for partial fulfilment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted By

Mr. UPPALA SAIRAM	**19911A04G9**
Miss. VANGA SRIKRISHNAVENI	**19911A04H0**
Mr. KODAPARTHI HARISH	**20915A0417**

Under the esteemed guidance of,

Dr. M . RAJENDRA PRASAD

Department of Electronics and Communication Engineering

Vidya Jyothi Institute of Technology

(An Autonomous Institution)

(Accredited by NAAC & NBA, Approved by AICTE New Delhi & Permanently Affiliated to JNTUH)

Aziz Nagar Gate, C.B. Post, Hyderabad-500 075

2022-2023

Vidya Jyothi Institute of Technology

(An Autonomous Institution)

(Accredited by NAAC & NBA, Approved by AICTE New Delhi & Permanently Affiliated to JNTUH)

Aziz Nagar Gate, C.B. Post, Hyderabad-500 075

Department of Electronics and Communication Engineering

CERTIFICATE

This is to certify that the project report entitled **IOT NODE NETWORK SOLUTION USING CONTIKI TOOL** is being submitted by **Mr. UPPALA SAIRAM (19911A04G9**), **Miss. VANGA SRIKRISHNAVENI (19911A04H0) & Mr. KODAPARTHI HARISH (20915A0417)** of B.Tech IV-I semester of Electronics & Communication Engineering is a record Bonafide work carried out by them. The results embodied in this report have not been submitted to any other University for the award of any degree.

INTERNAL GUIDE	**HEAD OF THE DEPARTMENT**
Dr. M. Rajendra Prasad	**Dr. S. Thulasi Prasad**
Department Of ECE	**Professor & HoD**
	Department Of ECE

EXTERNAL EXAMINER

DECLARATION

This is to certify that the work reported in the present project entitled **IOT NODE NETWORK SOLUTION USING CONTIKI TOOL ,** is a record of work done by us in the Department of Electronics and Communication Engineering, Vidya Jyothi Institute of Technology, Jawaharlal Nehru Technological University, Hyderabad. The reports are based on the project work done entirely by us and not copied from any other source.

Project Associates:

Mr. UPPALA SAIRAM	**19911A04G9**
Miss. VANGA SRIKRISHNAVENI	**19911A04H0**
Mr. KODAPARTHI HARISH	**20915A0417**

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our project internal guide,

Dr. M. Rajendra Prasad who has guided and supported us through every stage in the project.

We are really grateful to our **HoD Dr. S. Thulasi Prasad** for his time and much needed valuable guidance throughout our study.

We express our hearted gratitude to our principal **Dr. A. Padmaja** for giving us spontaneous encouragement for completing the project.

We thank **Dr. E. Saibaba Reddy**, Director of Vidya Jyothi Institute of Technology for encouraging us in the completion of our project.

It is our privilege to express our gratitude and indebtedness to our Secretary & Correspondent of Vidya Jyothi Institute of Technology **Shri. P. Rajeshwar Reddy** for his moral support.

We express our heartfelt thanks to the Staff of Electronics and Communication Department, Vidya Jyothi Institute of Technology for helping us in carrying out our project successfully.

Project Associates:

Mr. UPPALA SAIRAM	**19911A04G9**
Miss. VANGA SRIKRISHNAVENI	**19911A04H0**
Mr. KODAPARTHI HARISH	**20915A0417**

INDEX

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This Project describes the security solution for IoT network by using the Contiki and Network Simulator tools for faithful simulation study of networks and the behavior of nodes among them. Contiki OS consists of Cooja simulator which is built for the purpose of Simulation of the networks. We also use Network-Simulator 2 tool which is used for tracing the network files and provide the graphical user interface for the Networks we compile.

Here we implement the Three scenarios on Node placement in the network, by assigning UDP sender and Sink nodes.

For the 1st scenario we take 50% of the Sender nodes in the radio range of sink node and collect the data like the following parameters Average temperature, received packets per node etc.…

For the 2nd scenario we place 100% of the Sender nodes in the range of the Sink node and analyze the parameters by taking at the same time interval.

For the 3rd scenario we introduce a malicious node in the network in between UDP sender and Sink nodes. Therefore, analyzing the packet drop and interaction between the nodes.

CHAPTER 1

INTRODUCTION

Internet of Things (IoT) is the networking of physical objects that contain electronics embedded within their architecture in order to communicate and sense interactions amongst each other or with respect to the external environment. In the upcoming years, IoT-based technology will offer advanced levels of services and practically change the way people lead their daily lives. Advancements in medicine, power, gene therapies, agriculture, smart cities, and smart homes are just a very few of the categorical examples where IoT is strongly established.

Main components used in IoT as follows:

•**Low-power embedded systems:** Less battery consumption, high performance are the inverse factors that play a significant role during the design of electronic systems.

•**Sensors:** Sensors are the major part of any IoT applications. It is a physical device that measures and detect certain physical quantity and convert it into signal which can be provide as an input to processing or control unit for analysis purpose.

•**Control Units:** It is a unit of small computer on a single integrated circuit containing microprocessor or processing core, memory and programmable input/output devices/peripherals. It is responsible for major processing work of IoT devices, and all logical operations are carried out here.

•**Cloud computing:** Data collected through IoT devices is massive and this data must be stored on a reliable storage server. This is where cloud computing comes into play. The data is processed and learned, giving more room for us to discover where things like electrical faults/errors are within the system.

•**Availability of big data:** We know that IoT relies heavily on sensors, especially in real-time. As these electronic devices spread throughout every field, their usage is going to trigger a massive flux of big data.

•**Networking connection:** In order to communicate, internet connectivity is a must where each physical object is represented by an IP address. However, there are only a limited number of addresses available according to the IP naming. Due to the growing number of devices, this naming system will not be feasible anymore. Therefore, researchers are looking for another alternative naming system to represent each physical object.

The working of IOT is different for different IoT echo systems (architectures).

The entire working process of IoT starts with the device themselves, such as smartphones, digital watches, electronic appliances, which securely communicate with the IoT platform.

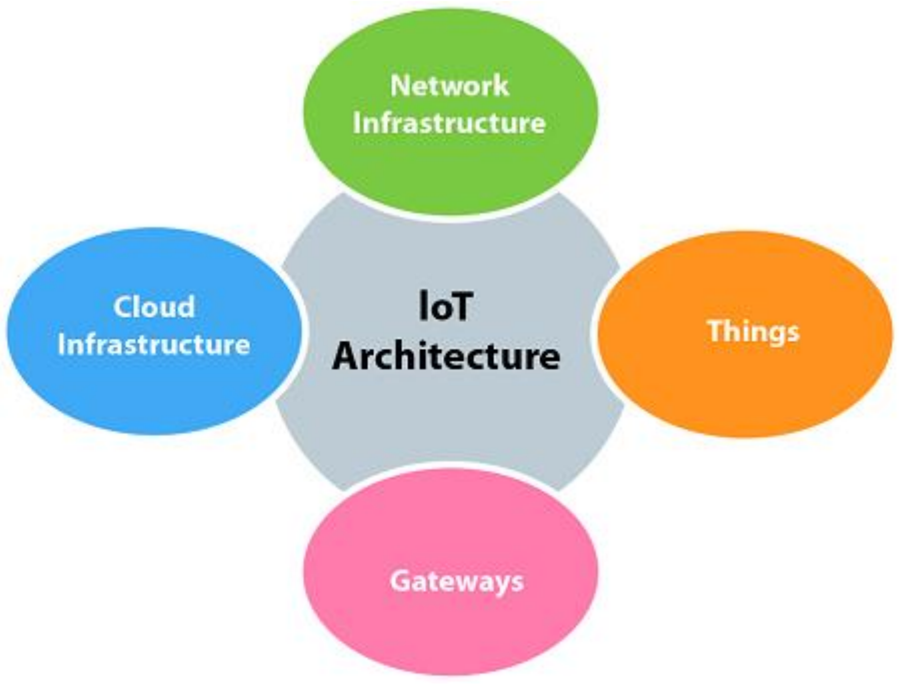


Figure 1.0 IoT Architecture

1.1 Features of IoT

The most important features of IoT on which it works are connectivity, analysing, integrating, active engagement, and many more. Some of them are listed below:

Connectivity: Connectivity refers to establish a proper connection between all the things of IoT-to-IoT platform it may be server or cloud. After connecting the IoT devices, it needs a high-speed messaging between the devices and cloud to enable reliable, secure and bi-directional communication.

Analysing: After connecting all the relevant things, it comes to real-time analysing the data collected and use them to build effective business intelligence. If we have a good insight into data gathered from all these things, then we call our system has a smart system.

Integrating: IoT integrating the various models to improve the user experience as well.

Artificial Intelligence: IoT makes things smart and enhances life through the use of data. For example, if we have a coffee machine whose beans have going to end, then the coffee machine itself order the coffee beans of your choice from the retailer.

Sensing: The sensor devices used in IoT technologies detect and measure any change in the environment and report on their status. IoT technology brings passive networks to active networks. Without sensors, there could not hold an effective or true IoT environment.

Active Engagement: IoT makes the connected technology, product, or services to active engagement between each other.

Endpoint Management: It is important to be the endpoint management of all the IoT system otherwise, it makes the complete failure of the system. For example, if a coffee machine itself order the coffee beans when it goes to end but what happens when it orders the beans from a retailer and we are not present at home for a few days, it leads to the failure of the IoT system. So, there must be a need for endpoint management.

1.2 IoT Design Framework

The IoT decision framework provides a structured approach to create a powerful IoT product strategy. The IoT decision framework is all about the strategic decision making. The IoT Decision Framework helps us to understand the areas where we need to make decisions and ensures consistency across all our strategic business decision, technical and more.

The IoT decision framework is much more important as the product or services communicates over networks goes through five different layers of complexity of technology.

1. Device Hardware
2. Device Software
3. Communications
4. Cloud Platform
5. Cloud Application

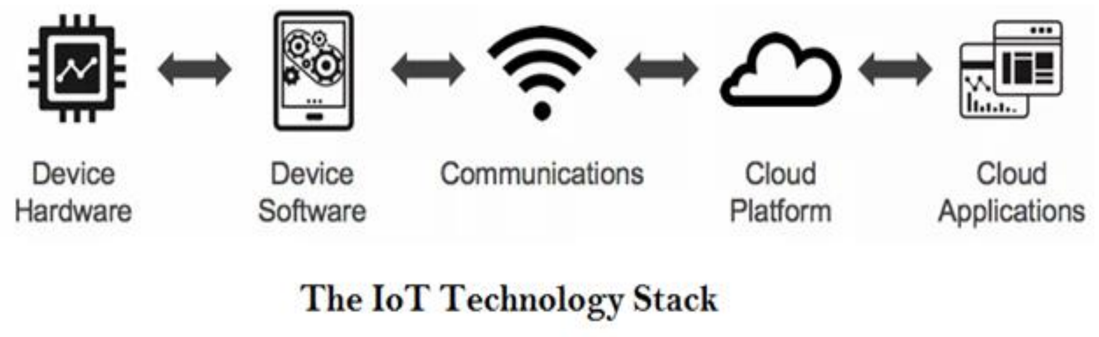


Figure 1.2 IoT Technology Stack

1.3 RPL protocol

Rpl stands for " Routing Protocol for Low power and Lossy Network".

It is a distance-vector protocol that supports a variety of Data link protocols.

RPL builds a Destination Oriented Directed Acrylic Graph (DODAG) which has only one route from each leaf node to the root.

All the traffic in this DODAG is routed through the root.

Initially, each node sends a DODAG information Object (DIO) announcing them self as a root.

This information travels in the network and complete DODAG is gradually built.

When a node wants to join the network, it sends a DODAG information Solicitation (DIS) request and root responds back with a DAO Acknowledgement (DAO-ACK) confirming the join.

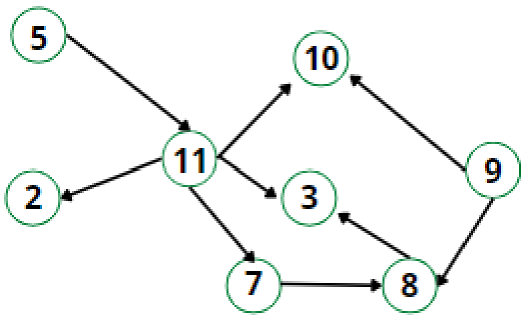


Figure 1.3 Directed Acrylic Graph

1.4 6LOWPAN

The 6LOWPAN protocol refers to IPV6 LOW POWER PERSONAL AREA NETWORK which uses a lightweight IP-based communication to travel over low data rate networks.

It has limite9d processing ability to transfer information wirelessly using an internet protocol. So , it is mainly used for home and building automation.

Operates within 2.4 GHz frequency range.

With 250 Kbps transfer rate.

It has a maximum length of 128-Bit header packets.

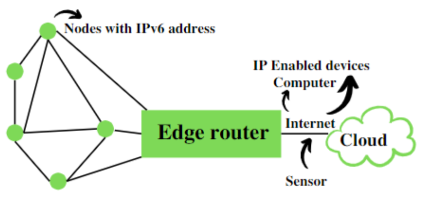


Figure 1.4 6LOWPAN

1.5 COAP (Constrained Application Protocol)

It is a session layer protocol that provides the Restful (HTTP) interface between HTTP client and server.

It is designed to use devices on the same constrained network between devices and general nodes on the internet.

COAP enables low power sensors to use Restful services while meeting their low power constraints. This protocol is specially built for IOT systems primarily based on HTTP protocols.

This network is used within the limited network or in a constrained environment.

The whole architecture of COAP consists of COAP clients, server, REST COAP proxy and REST internet.

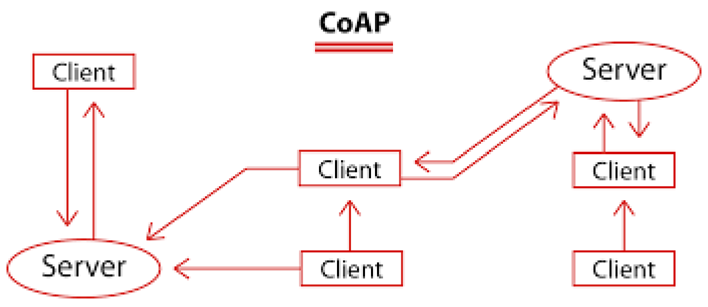


Figure 1.5 COAP protocol

1.6 IPV6

IP Address = " Internet Protocol Address"

It is a unique number provided to every device connected to the network, such as Android phone, laptop, Mac etc…

An IP address is represented in an integer number separated by a dot (.)

Example: 192.167.12.46

IPv6 is the next generation of IP addresses. The main difference between IPV4 and IPV6 is the address size of IP address.

IPV6 is a 128-bit hexadecimal address.

IPV6 provides a large address space, and it contains a simple header as compared to IPV4.

An IPV6 address are 128-Bit hexadecimal addresses,, written in 8-separate sections having each of them have 16 bits.

It also allows to remove the starting zeros (0) of each 16-Bit section. If two or more consecutive sections 16-Bit contains all zeros (0.0), they can be compressed using double colons(::)

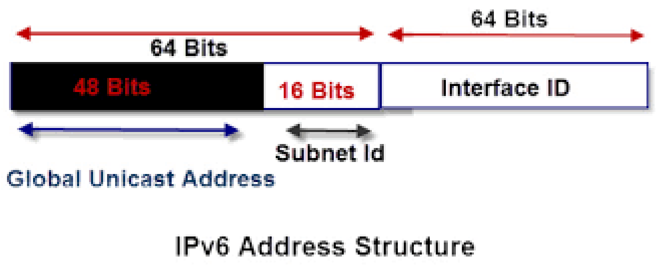


Figure 1.6 IPV6 Address Structure

1.7 Micro IP(uIP):

IP is the world’s smallest full TCP/IP stack. Intended for tiny microcontroller systems where code size and RAM are severely constrained, uIP only requires 4-5 kilobytes of code space and a few hundred bytes of RAM. uIP has been ported to a wide range of systems and has found its way into many commercial products.

Operations of uIP :

- Called in a time loop.
- Call manages other network behaviour.
- uIP calls the hardware driver to send the packet of data.
- It used packet buffer in a half-duplex way for transmission and reception.
- It needs to retransmit, its just call the applications code to send previous data.
- One task per connection.
- Task communicates with a task in a distant computer on other hand.
- Connections are held in a ray.
- On each call , uIP tries to serve a connection.
- Works in 10KB ROM , 2KB RAM and Contiki can easily fit in 100KB.

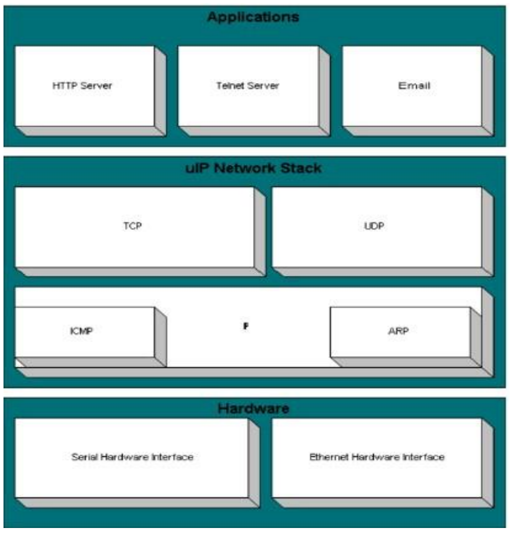


Figure 1.7 uIP Stack

1.8 Project Overview

This project describes about the node security between each node in the networks which are simulated by using the Cooja simulator and Network simulator.

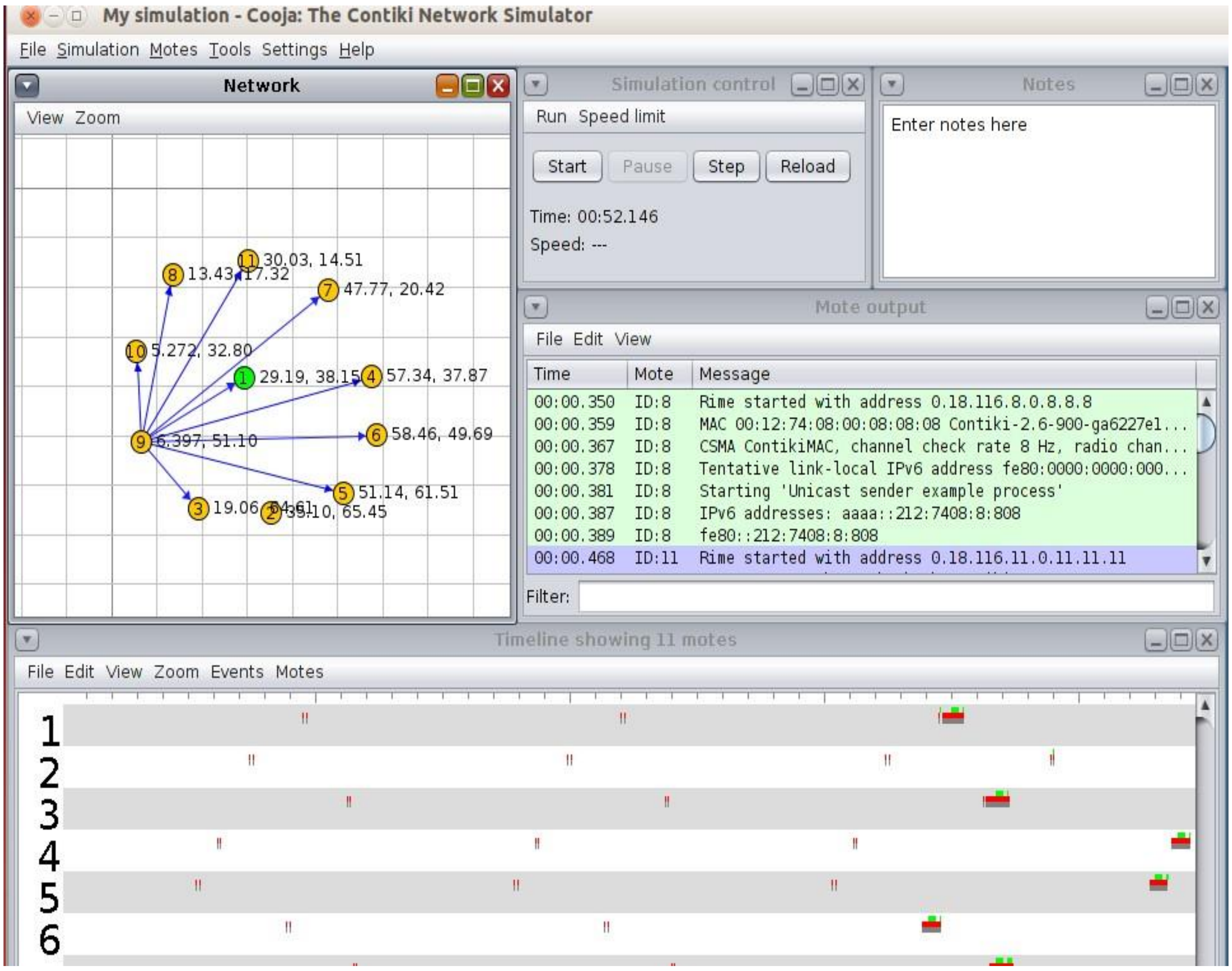
As part of the project, we are going to simulate the networks in 3 scenarios' basically analysing the parameters like packets received, power consumption, radio duty cycle, network hops, and radio duty cycle.

For the 1st scenario we take 50 % sender nodes inside and outside the range of sink node.

For the 2ND scenario we take 100% sender nodes inside the range of the sink node.

For the 3rd scenario we introduce a malicious node into the sink node range and simulate the network.

We collect the Sensor data using collect view application in Cooja simulator by sending commands to the nodes.



Time: 00:52.146
Speed: ---

```
Time        Mote   Message
00:00.350   ID:8   Rime started with address 0.18.116.8.0.8.8.8
00:00.359   ID:8   MAC 00:12:74:08:00:08:08:08 Contiki-2.6-900-ga6227e1...
00:00.367   ID:8   CSMA ContikiMAC, channel check rate 8 Hz, radio chan...
00:00.378   ID:8   Tentative link-local IPv6 address fe80:0000:0000:000...
00:00.381   ID:8   Starting 'Unicast sender example process'
00:00.387   ID:8   IPv6 addresses: aaaa::212:7408:8:808
00:00.389   ID:8   fe80::212:7408:8:808
00:00.468   ID:11  Rime started with address 0.18.116.11.0.11.11.11
```

Figure 1.8 Network Simulation in Cooja Simulator

CHAPTER 2

LITERATURE SURVEY

1. A Performance Evaluation of RPL in Contiki

In this study we investigate Objective Functions and the most influential parameters on Routing Protocol for Low power and Lossy Network (RPL) performance in Contiki (WSN OS) and then evaluate RPL performance in terms of Energy, Latency, Packet Delivery Ratio, Control overhead, and Convergence Time for the network

2. A Survey on routing protocols supported by the Contiki Internet of things

Standardization and technology advancements have helped the realization of the Internet of things (IoT). The availability of low-cost IoT devices has also played a key role in furthering IoT research, development, and deployment. IoT operating systems (OSs) provide integration of software and hardware components. The availability of standard protocols, heterogeneous hardware support, ease of development, and simulation or emulation support are desirable features of IoT OSs. Contiki OS is one of the contenders for future IoT OS's.

3. Reconstructing data in IoT devices running Contiki OS

The Coffee File System is designed for flash memory with the peculiarities this storage technology brings. Modifying data stored on flash memory can only flip individual bits from "1" to "0", and to flip bits back to "1", a whole erase block needs to be reset. An erase block consists of several pages; for the Flash memory Micron M25P80, a page is 256 bytes, and an erase block is 65 kB (256 pages).

4. Safe Contiki OS: Type and Memory Safety for Contiki OS

Embedded systems, especially Wireless Sensor Nodes are highly prone to Type Safety and Memory Safety issues. Contiki, a prominent Operating System in the domain is even more affected by the problem since it makes extensive use of Type casts and Pointers. The work is an attempt to nullify the possibility of Safety violations in Contiki. We use a powerful, still efficient tool called Deputy to achieve this. We also try to automate the process.

5. Enhancing the QoS of IoT Networks with Lightweight Security Protocol using Contiki

The Internet of Things (IoT) is advancing to prevail the application of the Internet, with the vision to connect everything around us. The deployment of IoT is advancing at a very fast pace and relying on modified versions of the TCP/IP protocol suits. This rapid growth of the field is leaving several critical issues unresolved. Among the most critical issues are the quality of service and security of the delivered data. This research is set to tackle these issues through proposing a data delivery scheme that improves the quality of service (QoS) of classified data. The proposed solution relies on differentiating the priority of the delivered data, and to give preferences to secured and user-defined high priority traffic.

CHAPTER 3
SOFTWARE/HARDWARE TOOLS

3.1 VIRTUAL MACHINE

A virtual workstation is another term for a client device connecting to a virtual machine (VM) that hosts desktops and applications. These virtual machines work on top of hypervisor software that lies on a single powerful hardware. They provide access to full desktop environments and store all the data and applications remotely, presenting a cost-saving adoption for businesses, IT departments, and resource-heavy research work.

Virtual workstations rely on virtualization software, which abstracts operating systems, applications, and data from a computing device's underlying hardware. Being virtual allows it to run on high-spec servers utilizing virtual desktop infrastructure (VDI) or Remote Desktop Session Host (RDSH) benefits, such as central data management. This also allows for load balancing, maintenance on hardware, on-demand compute power addition and redundancy. The ability to run several operating systems simultaneously on the same hardware—and even on the cloud—is what makes users more inclined toward a virtualized workstation.

Figure 3.1 VMware Virtual machine

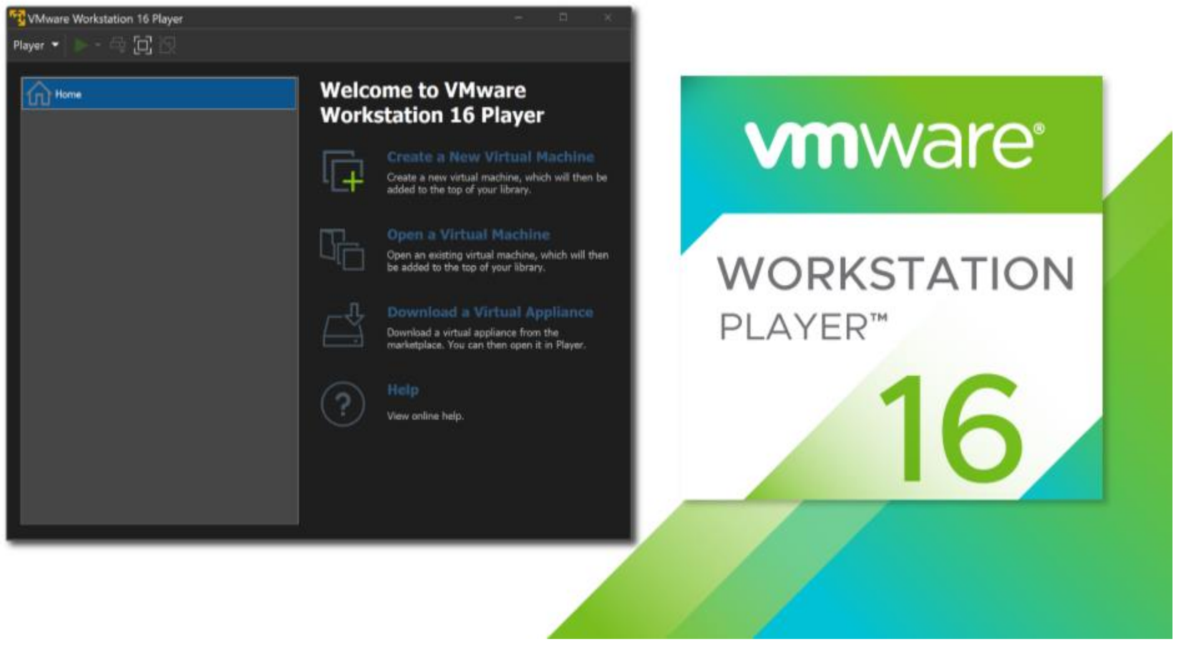
3.2 VMWARE Workstation

VMware Workstation is a great virtualization product that allows you to run multiple virtual machines on Linux and Windows operating systems installed on physical computers. When multiple users need to work with the same virtual machines, they can install VMware Workstation Server on their computers, copy the original VM, and run VM copies on their computers.

Copying a VM to other computers requires additional disk, CPU and memory resources. If the computers are connected to a single LAN (local area network), this method may be not rational. As an alternative, you can provide remote access to a virtual machine (VM) using traditional remote access protocols such as RDP, SSH, VNC etc. This is a credible idea, and a computer running such VMs that are accessible remotely seems to work as a server.

Alternatively, you can log in to the host machine (a machine running a hypervisor such as VMware Workstation), run VMware Workstation and power on VMs manually after power loss or restarting a host machine.

Fortunately, VMware allows you to configure VMware Workstation as a server and share VMs over the network, which is a good option for workgroups.


VMware Workstation 16 Player
Player
Home
Welcome to VMware Workstation 16 Player
Create a New Virtual Machine
Open a Virtual Machine
Download a Virtual Appliance
Help
vmware
WORKSTATION
PLAYER™
16

Create a new virtual machine, which will then be added to the top of your library.

Open an existing virtual machine, which will then be added to the top of your library.

Download a virtual appliance from the marketplace. You can then open it in Player.

View online help.

Figure 3.2 VM ware interface

3.3 Contiki Operating System

Figure 3.3a Contiki OS

Contiki is an operating system for IoT that specifically targets small IoT devices with limited memory, power, bandwidth, and processing power. It uses a minimalist design while still packing the common tools of modern operating systems. It provides functionality for management of programs, processes, resources, memory, and communication. It owes its popularity to being very lightweight (by modern standards), mature, and flexible. Many academics, organization researchers, and professionals consider it a go-to OS.

Contiki only requires a few kilobytes to run, and within a space of under 30KB, it fits its entire operating system – a web browser, web server, calculator, shell, telnet client and daemon, email client, vnc viewer, and ftp. It borrows from operating systems and development strategies from decades ago, which easily exploited equally small space.

Contiki supports standard protocols and recent enabling protocols for IoT :

•uIP (for IPv4) − This TCP/IP implementation supports 8-bit and 16-bit microcontrollers.

•uIPv6 (for IPv6) − This is a fully compliant IPv6 extension to uIP.

•Rime − This alternative stack provides a solution when IPv4 or IPv6 prove prohibitive. It offers a set of primitives for low-power systems.

•6LoWPAN − This stands for IPv6 over low-power wireless personal area networks. It provides compression technology to support the low data rate wireless needed by devices with limited resources.

•RPL − This distance vector IPv6 protocol for LLNs (low-power and lossy networks) allows the best possible path to be found in a complex network of devices with varied capability.

•CoAP − This protocol supports communication for simple devices, typically devices requiring heavy remote supervision.

The **uIP** is an open-source implementation of the TCP/IP network protocol stack intended for use with tiny 8- and 16-bit microcontrollers. It was initially developed by Adam Dunkel’s of the Networked Embedded Systems group at the Swedish Institute of Computer Science, licensed under a BSD style license, and further developed by a wide group of developers.

uIP makes many unusual design choices in order to reduce the resources it requires. uIP's native software interface is designed for small computer systems with no operating system. It can be called in a timed loop, and the call manages all the retries and other network behaviour. The hardware driver is called after uIP is called. uIP builds the packet, and then the driver sends it, and optionally receives a response.

It is normal for IP protocol stack software to keep many copies of different IP packets, for transmission, reception and to keep copies in case they need to be resent. uIP is economical in its use of memory because it uses only one packet buffer. First, it uses the packet buffer in a half-duplex way, using it in turn for transmission and reception. Also, when uIP needs to retransmit a packet, it calls the application code in a way that requests for the previous data to be reproduced.

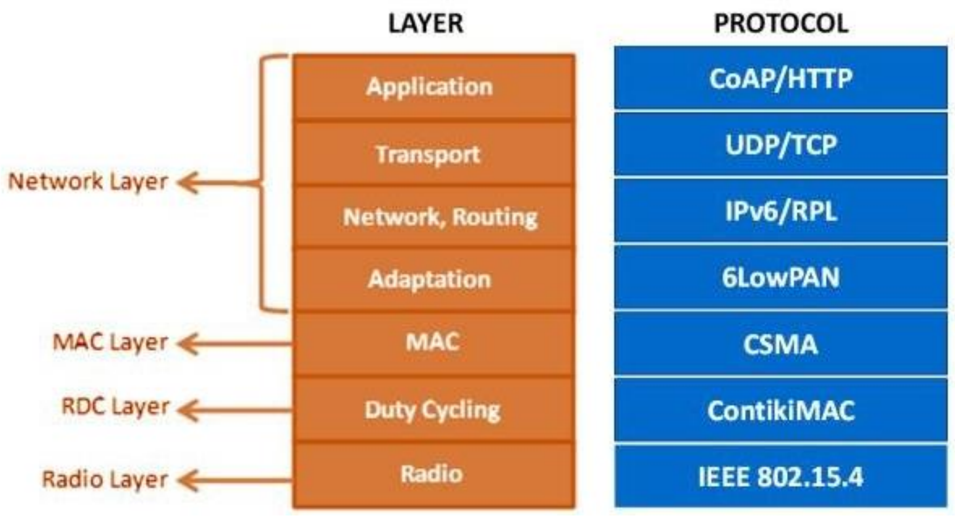


Figure 3.3b Contiki Network Stack

3.4 CONTIKI-COOJA NETWORK SIMULATOR

Cooja network is specified in simulating the sensor nodes as per the features for utilizing in Java native interface. It can implement the programming codes in TinyOS and ContikiOS.

In general, there are plenty of uses in the Cooja network simulator for your better understanding we have listed some advantages

•It is utilized to evaluate the performance of the several localization protocols

•It assists the motes of Contiki simulation in all the networks

•It emulates the motes to reach the hardware level and Cooja is used to deliver the accurate power consumption for the protocols such as DV-Hop, fingerprint, and centroid

The following is about the programming languages in the Cooja network simulator

Programming languages in Cooja Simulator:

•Java

•C

The operating system is a significant part of the research process and here in the Cooja simulator.

OS Support in Cooja Network Simulator

•Contiki-2.7

•Ubuntu-14.04

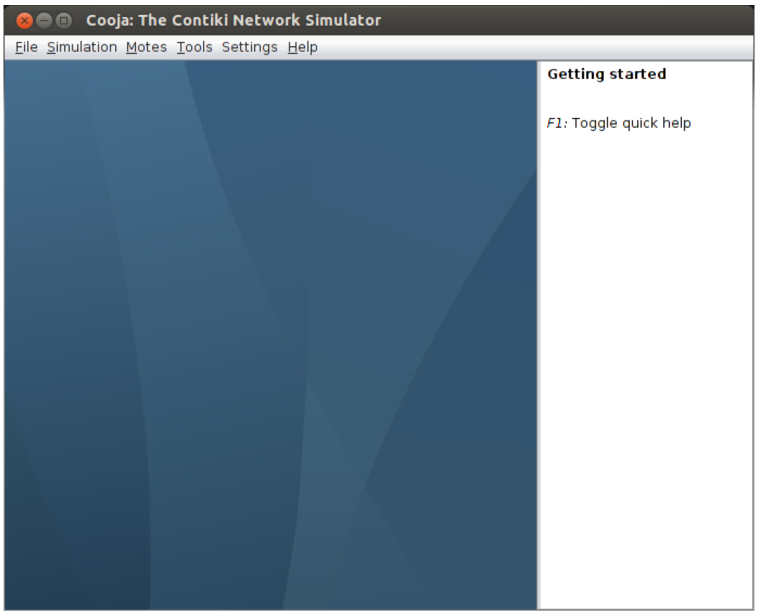

Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Getting started
F1: Toggle quick help

Figure 3.4 Cooja Simulator

3.5 Steps involved in Running Cooja Simulator

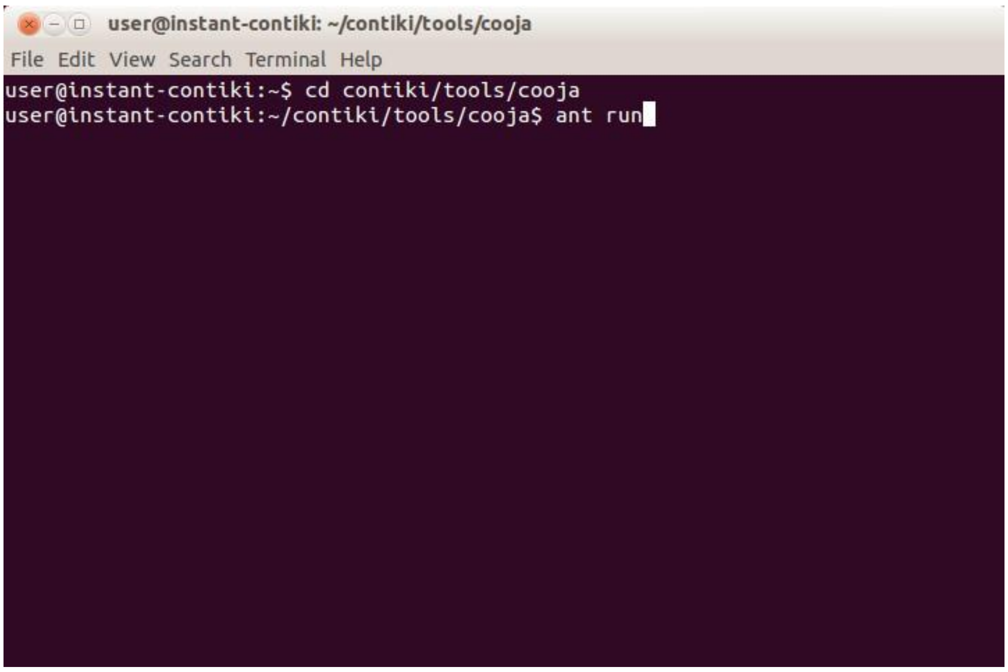
Step1:

The simple way of running Cooja is executing it inside its own directory by opening the terminal and typing the following commands.

Commands for running Cooja simulator:

cd Contiki/tools/cooja

ant run



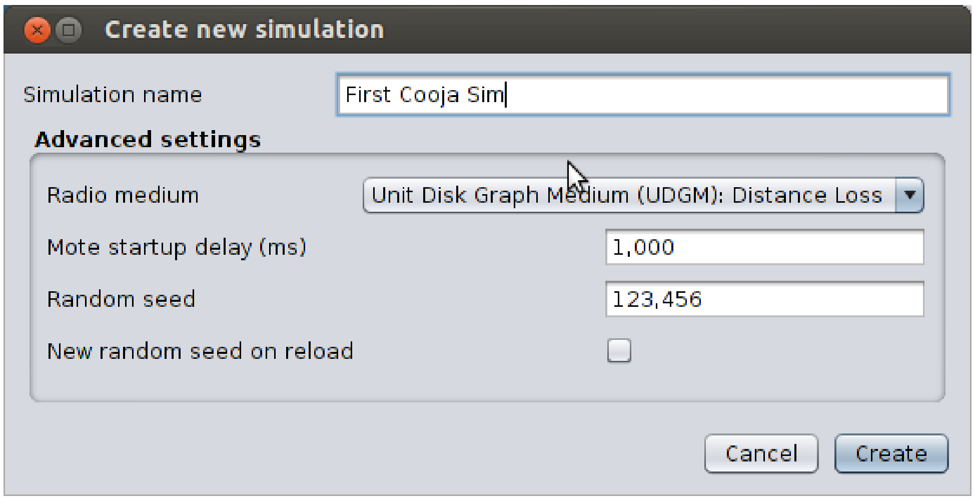
user@instant-contiki: ~/contiki/tools/cooja
File Edit View Search Terminal Help

```
user@instant-contiki:~$ cd contiki/tools/cooja
user@instant-contiki:~/contiki/tools/cooja$ ant run
```

Figure 3.5 Running Cooja Simulator

Step2: Creating a new simulation

In the File menu you can start a new simulation or open an existing one. At this moment we will start a new one. You should select: File > New simulation... The following window should show up.



Simulation name: First Cooja Sim

Radio medium: Unit Disk Graph Medium (UDGM): Distance Loss

Mote startup delay (ms): 1,000

Random seed: 123,456

New random seed on reload: ☐

Figure 3.5a Create new Simulation

In the Simulation name box, you should enter an identifier for the new simulation, and in Advanced settings you can choose parameters of the simulation such as Radio medium, node start-up delay and random seed generation.

After creating a new simulation, Cooja's window is filled with the main simulating tools, as shown in the next image.

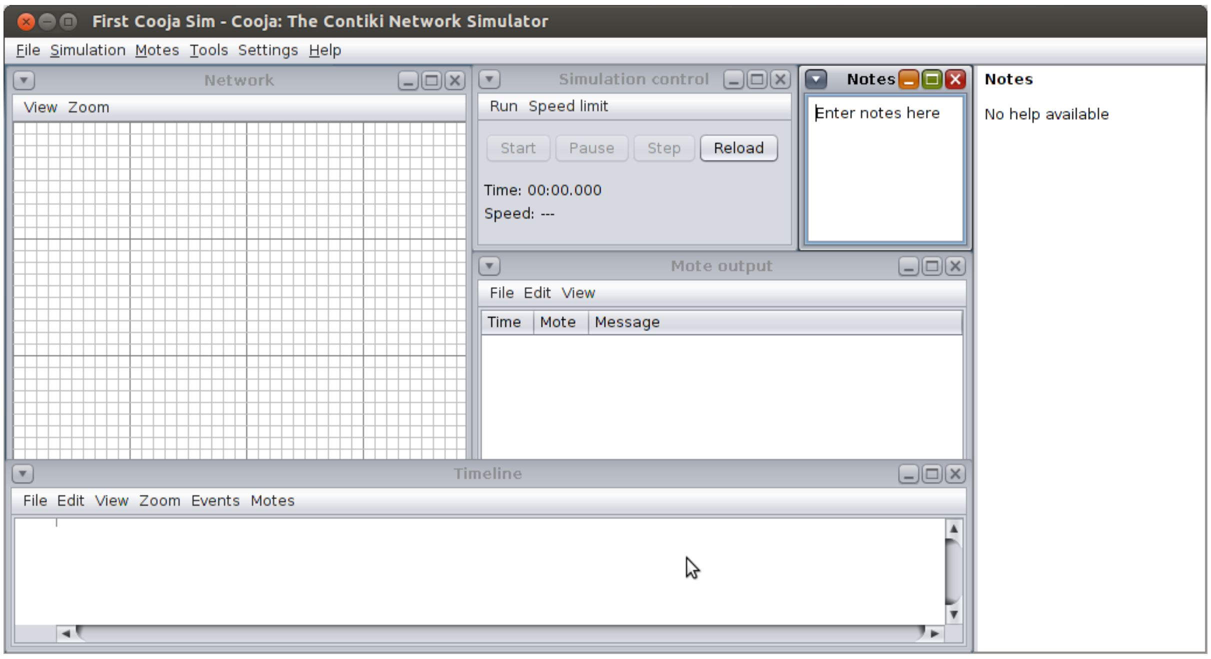


Figure 3.5b Simulation Window

Here we briefly describe the functionalities of each tool:

Network - Shows the location of each node in the network. Can be used to visualize the status of each node, including LEDs, mote IDs, addresses, lof outputs, etc. Initially this window is empty, and we need to populate it with our sensors.

Simulation Control - This panel is used to Start, Pause, Reload or execute Steps of the simulation. It shows the time of execution and the speed of simulation. It means that we can run the events several times faster than it would take in real-time execution.

Notes - This is a simple notepad for taking notes about the simulation.

Mote output - Shows all output of serial interface of the nodes. It is possible to enable one window of Mote output for each node in the simulation.

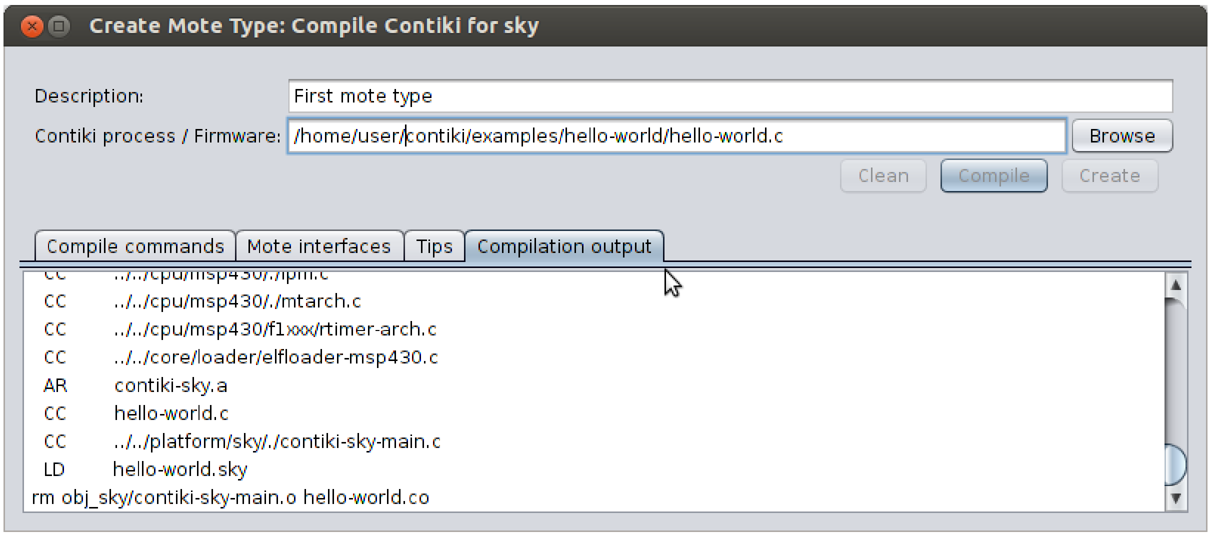
Timeline - Simulation timeline where messages and events such as channel change, LEDs change, log outputs, etc are shown.

In addition to the default tools, it is possible to exhibit other tools such as **Breakpoints**, **Radio messages**, **Script editor**, **Buffer view** and **Mote duty cycle**, which can be enable in the Tools menu.

Step3 :Creating a new mote type

You need to create a new mote type before starting any simulation. You can do this in the menu **Motes** > **Add motes** > **Create new motes type**. Let's select **Sky mote** in order to create a mote of the same type as the used Tmote Sky.

The window that shows up (see below) asks for the **Description** of the new mote type and the **Contiki process / Firmware**. You can name your mote type as *First mote type* and you can select the firmware that will be used during the simulation using the *Browse* button. After selecting the desired firmware, you can test the compilation click the *Compile* button. In this example we will use the Hello World firmware, typically located at /contiki/examples/hello-world/hello-world.c. If the compiling process is successed, you will see a final message: *LD* hello-world.sky in the Compilation output tab.



Description: First mote type

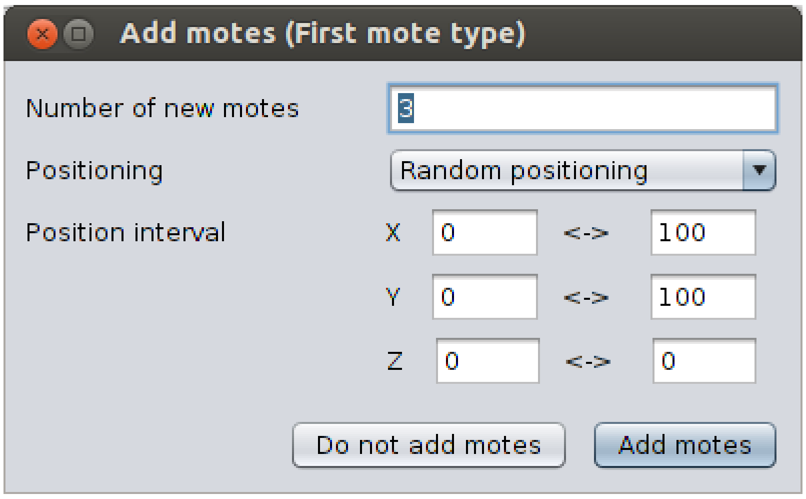
Contiki process / Firmware: /home/user/contiki/examples/hello-world/hello-world.c

```
CC      ../../cpu/msp430/./mtarch.c
CC      ../../cpu/msp430/f1xxx/rtimer-arch.c
CC      ../../core/loader/elfloader-msp430.c
AR      contiki-sky.a
CC      hello-world.c
CC      ../../platform/sky/./contiki-sky-main.c
LD      hello-world.sky
rm obj_sky/contiki-sky-main.o hello-world.co
```

<u>Figure 3.5.c Uploading the Firmware</u>

<u>Step4 :</u>Adding motes and running the simulation

After successfully compiling the firmware in the last step you should click on Create button A new mote type will be created, and you will be able to add a few nodes of that same type in your simulation. You will see the window shown below, where you will be able to set the number of nodes that will be created and specify their positioning. In this example we are going to create 3 new nodes and they will have random positioning.



Number of new motes: 3

Positioning: Random positioning

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

<u>Figure 3.5d Add motes</u>

Step5 : Saving the Simulation file

The simulation configuration and parameters such as number of nodes, type of nodes, firmware used, location of nodes, etc. can be stored in a file for future simulations. You can save your simulation configuration in *File* > *Save simulation as...*. The generated file has extension ".csc".

3.6 NETWORK SIMULATOR

NS-2

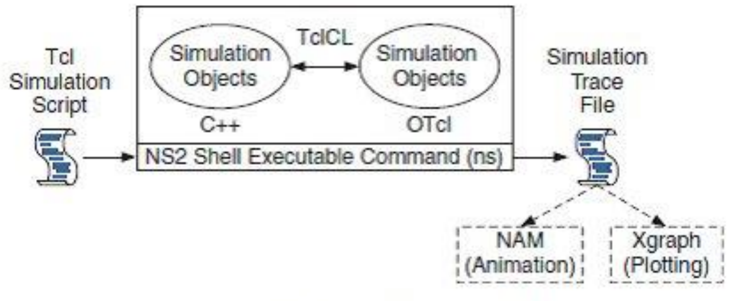
NETWORK SIMULATOR

Figure 3.6 Network Simulator 2

Network Simulator (Version 2), widely known as NS2, is simply an event-driven simulation tool that has proved useful in studying the dynamic nature of communication networks. Simulation of wired as well as wireless network functions and protocols (e.g., routing algorithms, TCP, UDP) can be done using NS2.

In general,NS2 provides users with a way of specifying such network protocols and simulating their corresponding behaviours. Due to its flexibility and modular nature, NS2 has gained constant popularity in the networking research community since its birth in 1989. Ever since, several revolutions and revisions have marked the growing maturity of the tool, thanks to substantial contributions from the players in the field.

NS2 consists of two key languages: CCC and Object-oriented Tool Command Language (OTcl). While the CCC defines the internal mechanism (i.e., a backend) of the simulation, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events (i.e., a frontend). The CCC and the OTcl are linked together using TclCL.



Basic architecture of NS.

Figure 3.6a Architecture of NS

Mapped to a CCC object, variables in the OTcl domains are sometimes referred to as handles. Conceptually, a handle is just a string (e.g., "_o10") in the OTcl domain and does not contain any functionality. Instead, the functionality (e.g., receiving a packet) is defined in the mapped CCC object (e.g., of class Connector).

In the OTcl domain, a handle acts as a frontend which interacts with users and other OTcl objects. It may define its own procedures and variables to facilitate the interaction. Note that the member procedures and variables in the OTcl domain are called instance procedures (instprocs) and instance variables (instvars), respectively.

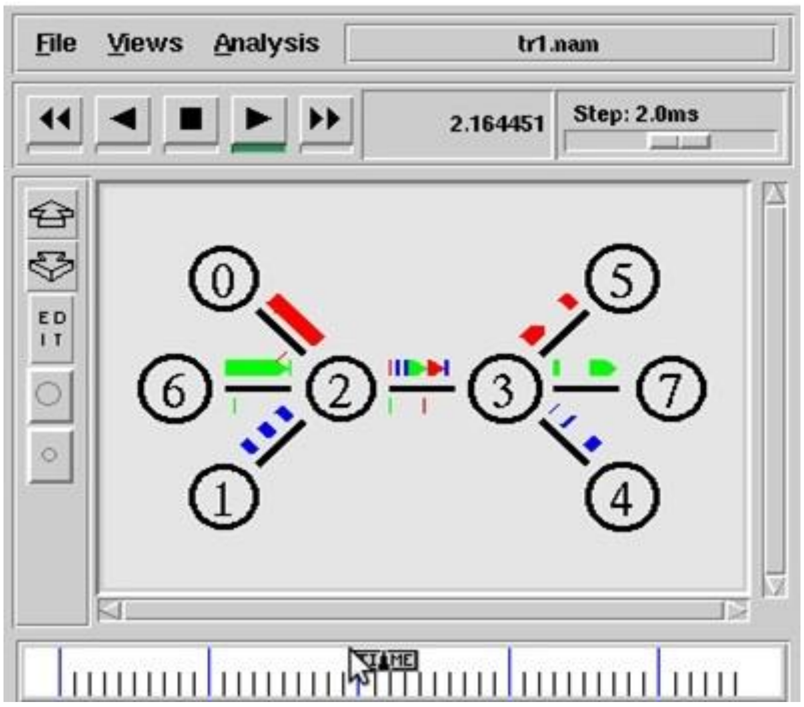


Figure 3.6b Network Animator

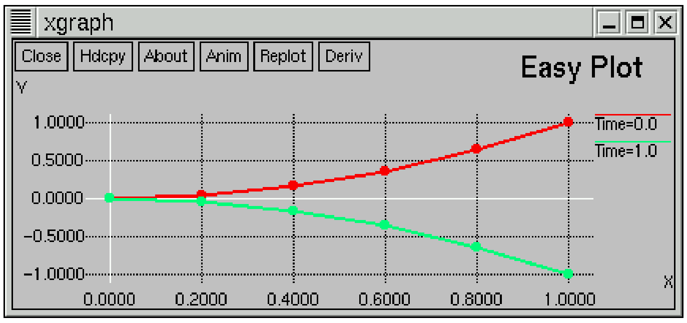


Figure 3.6c Xgraph

Hardware Requirements:

- Laptop with 8GB RAM.
- Storage about 256 GB.

CHAPTER 4
BLOCK DIAGRAM AND IMPLEMENTATION

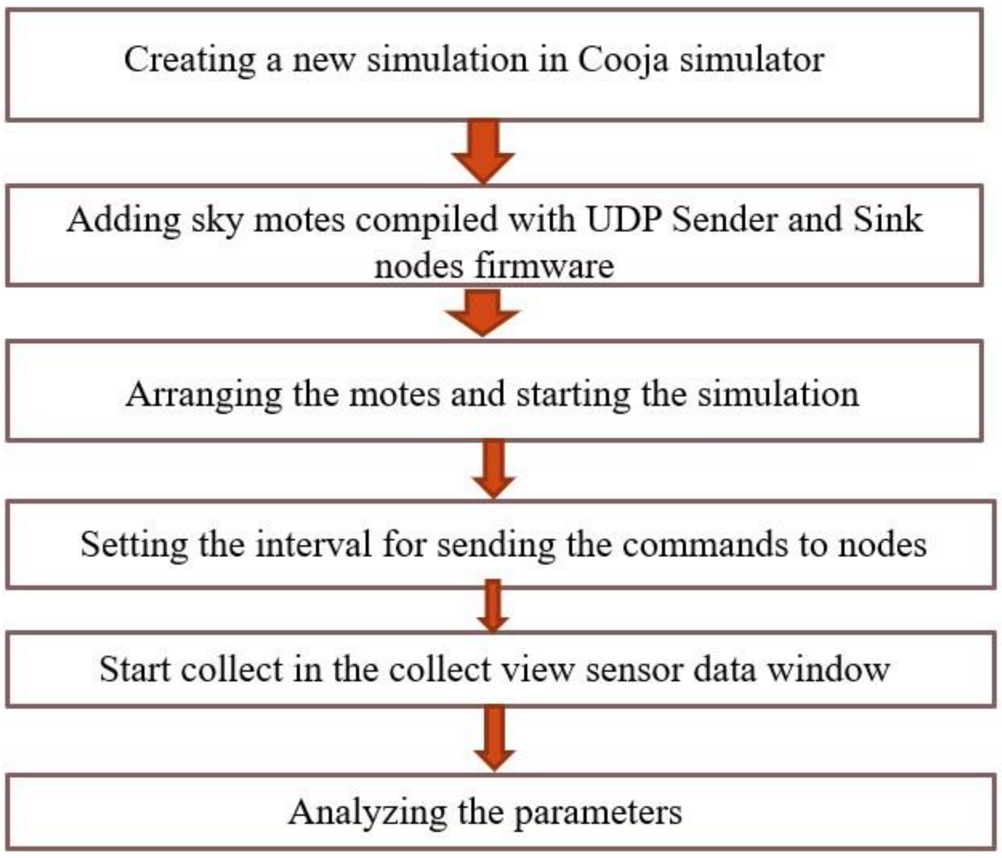

Creating a new simulation in Cooja simulator
Adding sky motes compiled with UDP Sender and Sink nodes firmware
Arranging the motes and starting the simulation
Setting the interval for sending the commands to nodes
Start collect in the collect view sensor data window
Analyzing the parameters

Figure 4.0 Block Diagram

4.0 IMPLEMENTATION

Here in this project, we are going to implement 3 scenarios of simulations using Cooja and Network simulators and analysing the parameters.

4.1 Case 1 Implementation:

- In case 1 we implement 50 % of Sender nodes inside and outside the range of Sink node.
- Firstly, we open the cooja simulator.
- Next, we create UDP sender and Sink nodes.
- Here we upload the firmware of UDP sender and UDP sink node .

➢ We take Sky motes because, Sky mote is a mote platform for extremely low power high data-rate sensor network applications.

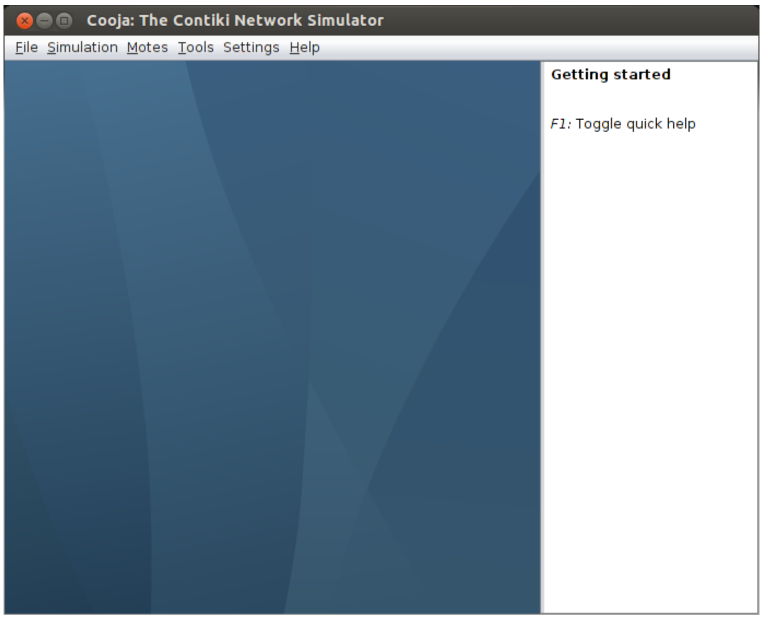

Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Getting started
F1: Toggle quick help

Figure 4.1a Open Cooja Simulator

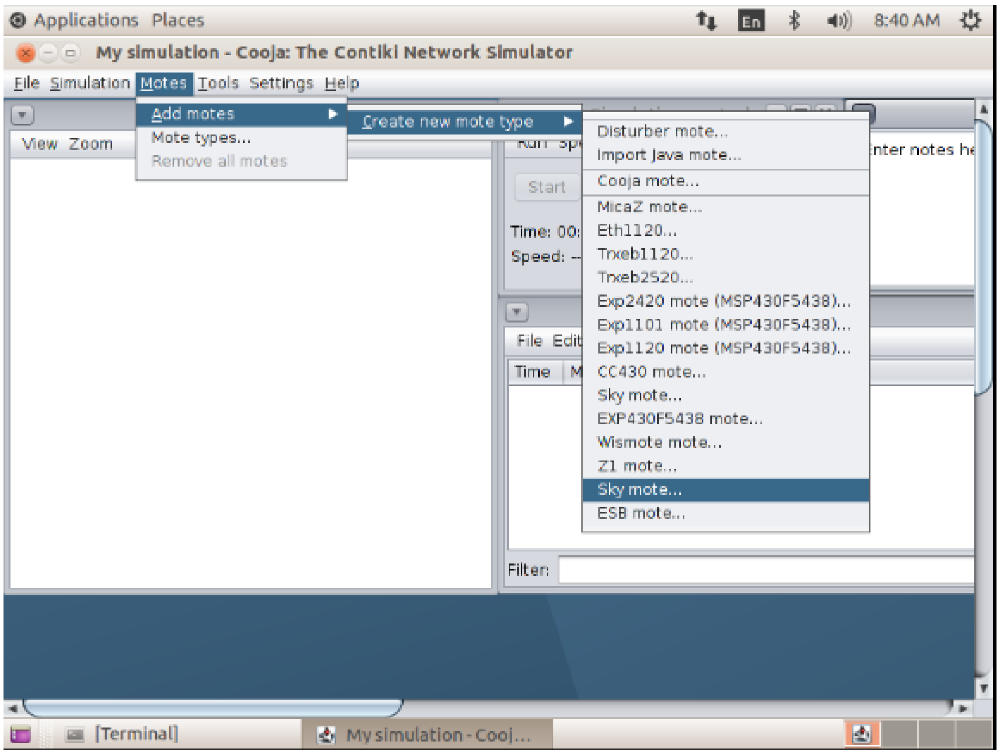
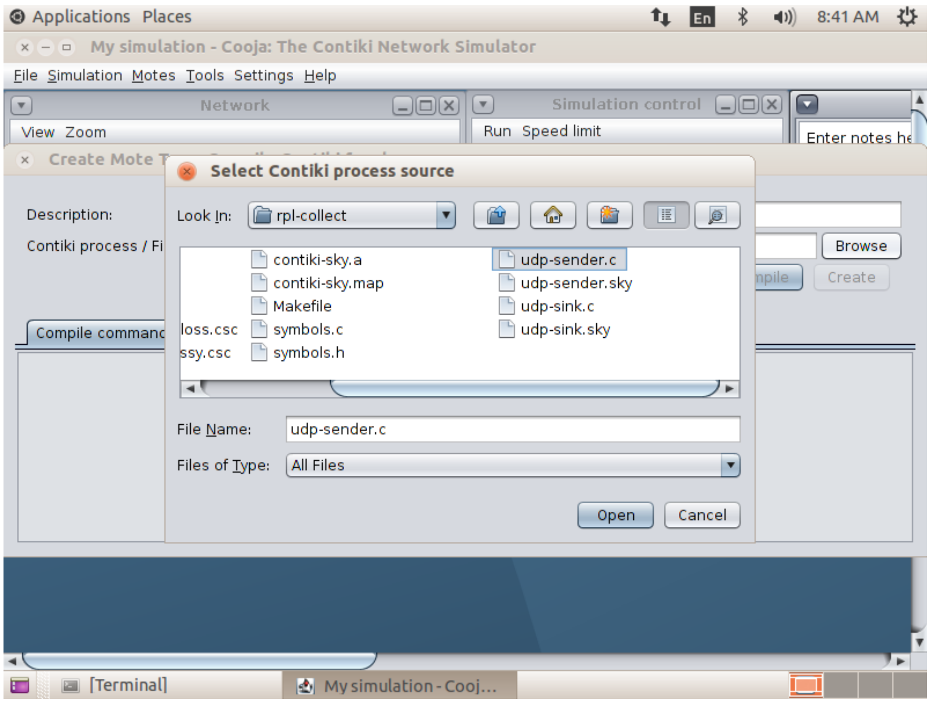


Figure 4.1b Adding the motes



Description:

Contiki process / Fi

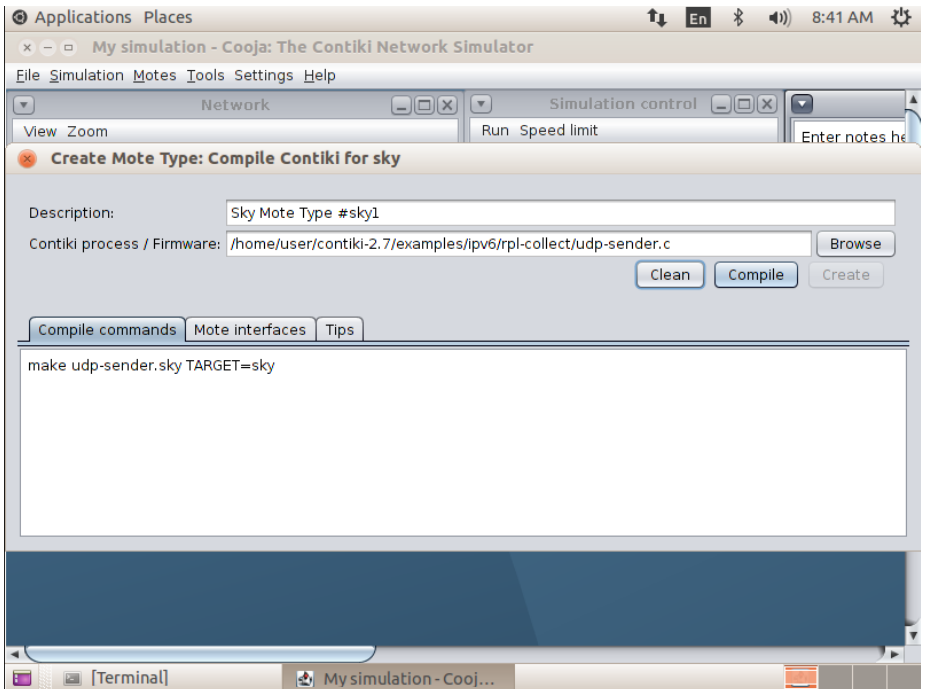
Look In: rpl-collect

```
loss.csc   contiki-sky.a      udp-sender.c
ssy.csc    contiki-sky.map    udp-sender.sky
           Makefile           udp-sink.c
           symbols.c          udp-sink.sky
           symbols.h
```

File Name: udp-sender.c

Files of Type: All Files

Figure 4.1c Uploading udp-Sender Firmware to Sky mote

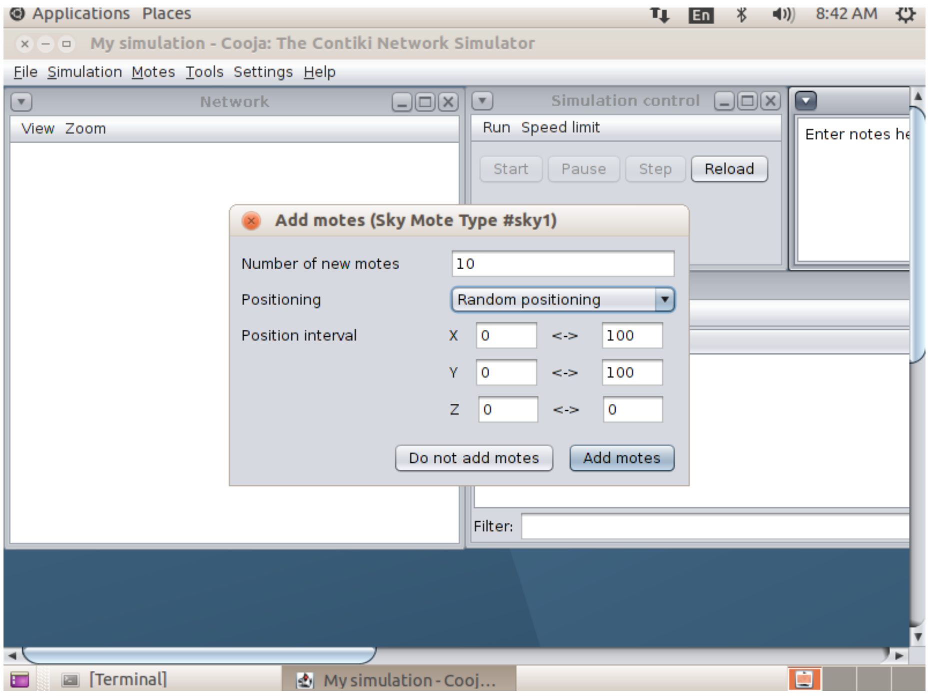


Description: Sky Mote Type #sky1

Contiki process / Firmware: /home/user/contiki-2.7/examples/ipv6/rpl-collect/udp-sender.c

```
make udp-sender.sky TARGET=sky
```

Figure 4.1d Compiling the program into motes

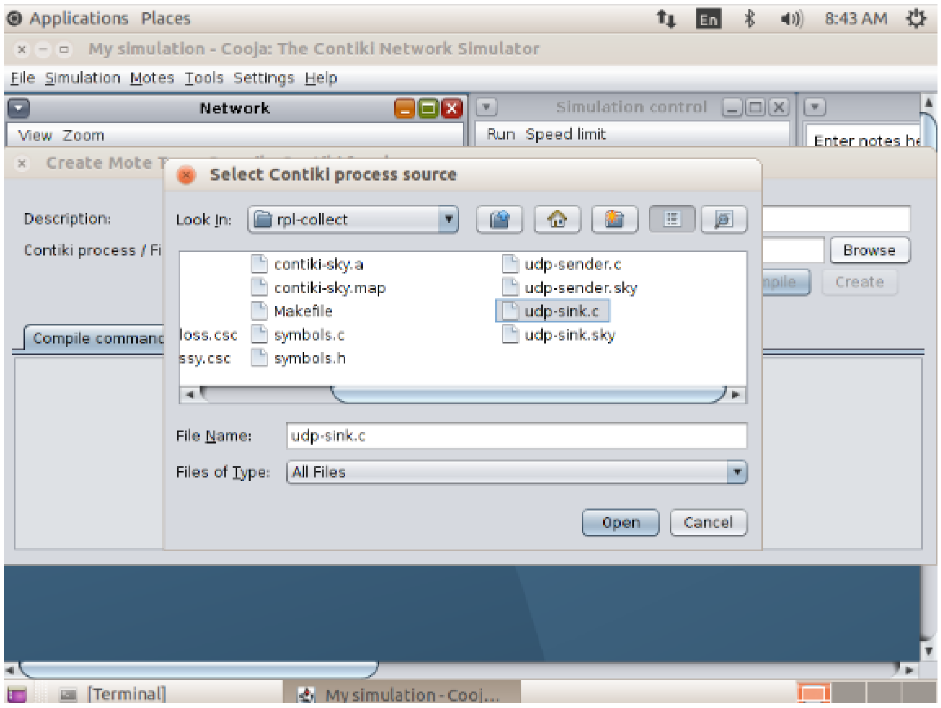


Add motes (Sky Mote Type #sky1)

Number of new motes: 10
Positioning: Random positioning
Position interval: X 0 <-> 100, Y 0 <-> 100, Z 0 <-> 0

Do not add motes | Add motes

Figure 4.1e selecting the number of motes to be added



Select Contiki process source

Look In: rpl-collect

```
loss.csc  contiki-sky.a      udp-sender.c
ssy.csc   contiki-sky.map    udp-sender.sky
          Makefile           udp-sink.c
          symbols.c          udp-sink.sky
          symbols.h
```

File Name: udp-sink.c
Files of Type: All Files

Open | Cancel

Figure 4.1f Uploading the Sink node firmware on to the mote

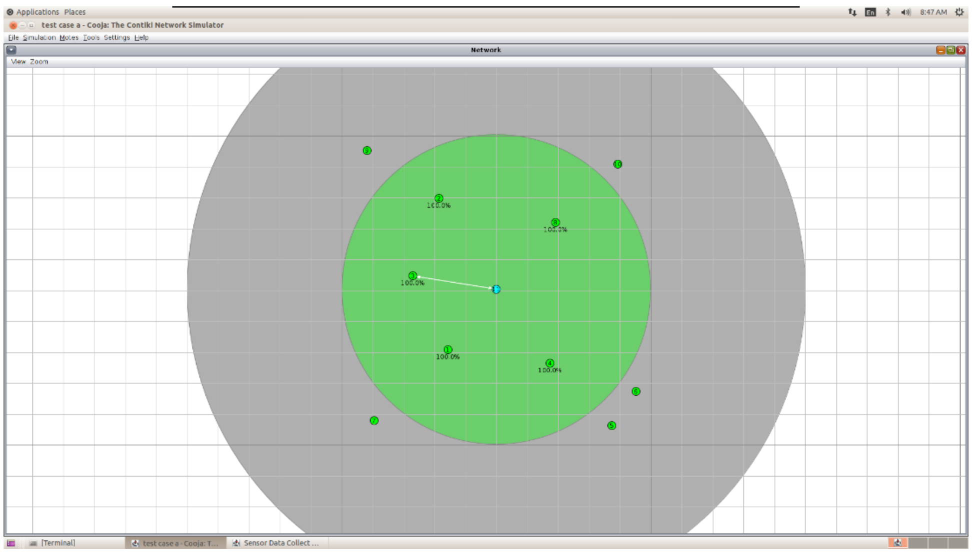


Figure 4.1g Case 1 Network

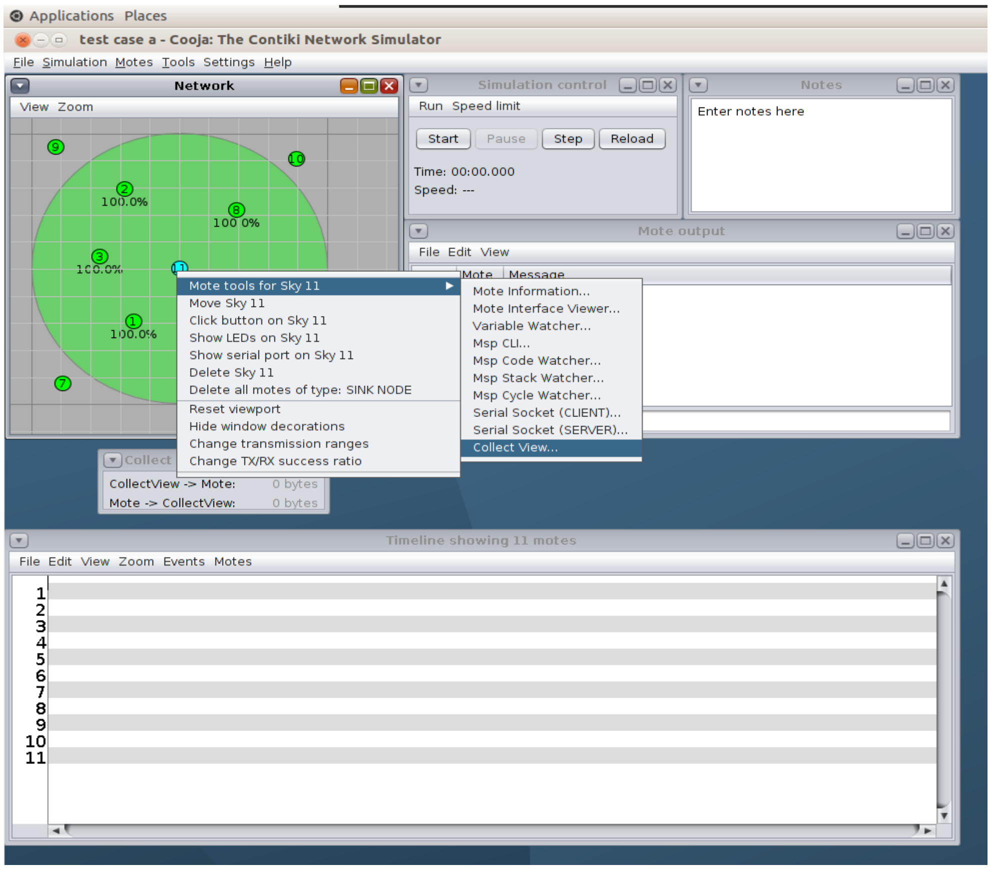
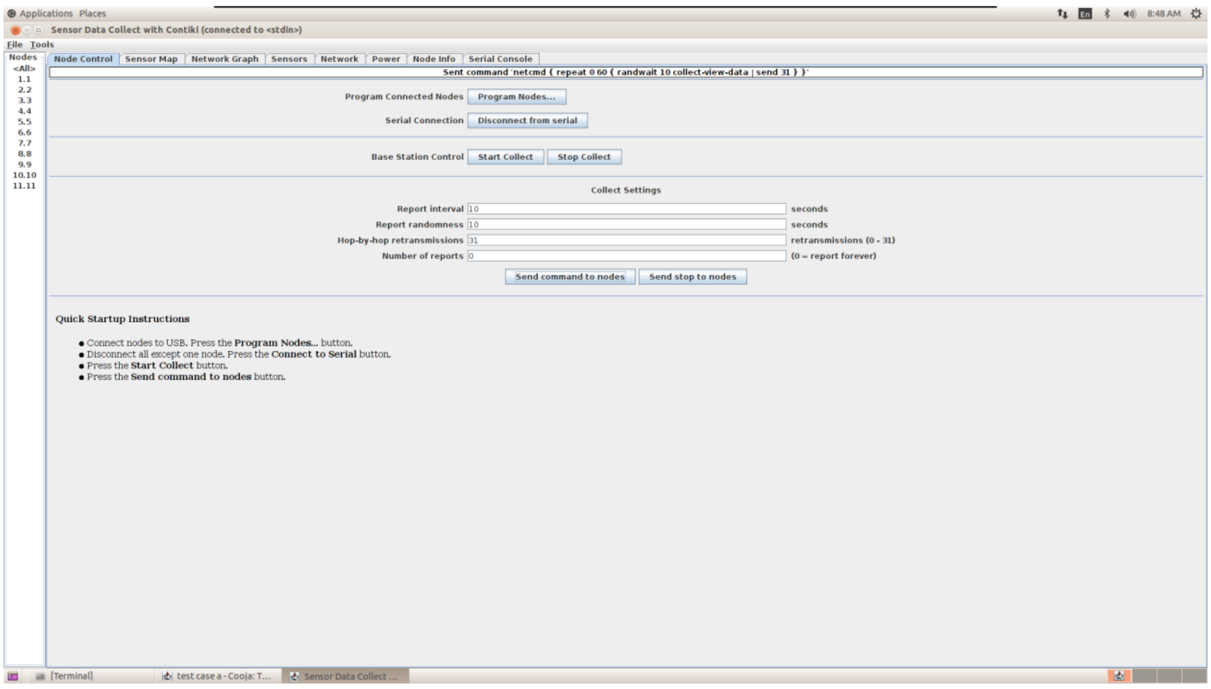


Figure 4.1h Collecting the Data

In order to collect the sensor data , we need to use Collect view application in Cooja Simulator.

To open Collect view Got to the Sink node – select Mote tools- Go to Collect view.



```
Sent command 'netcmd { repeat 0 60 { randwait 10 collect-view-data | send 31 } }'
```

Nodes
<All>
1.1
2.2
3.3
4.4
5.5
6.6
7.7
8.8
9.9
10.10
11.11

Program Connected Nodes: Program Nodes...
Serial Connection: Disconnect from serial
Base Station Control: Start Collect | Stop Collect

Collect Settings

Report interval	10	seconds
Report randomness	10	seconds
Hop-by-hop retransmissions	31	retransmissions (0 - 31)
Number of reports	0	(0 = report forever)

Send command to nodes | Send stop to nodes

Quick Startup Instructions

- Connect nodes to USB. Press the **Program Nodes...** button.
- Disconnect all except one node. Press the **Connect to Serial** button.
- Press the **Start Collect** button.
- Press the **Send command to nodes** button.

Figure 4.1j Collect view

To collect the data, we need to click on Start collect and send commands options.

After the given amount of time, the network starts simulating with the given interval of time between each node.

All the sender nodes provide the information among them and get settled at Sink node at which all the Node's and their data has been stored at.

4.2 Case 2 Implementation:

- ➢ In case 2 we implement 100% sender node inside the range of Sink node .
- ➢ The following the case 2 follows the same steps as case1.
- ➢ But the arrangement is different for case 2.

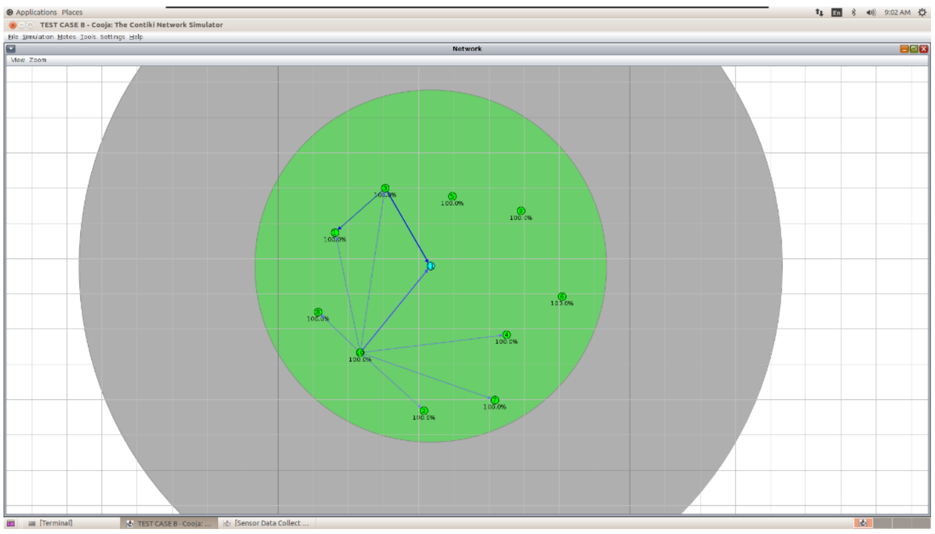


Figure 4.2 case 2 implementation

After simulating the case 2 , we again need to take the sensor data in order to have the results.

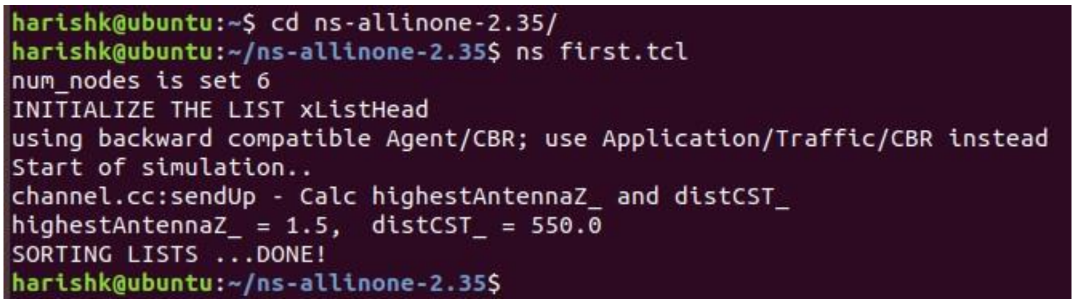
For both the cases we need to take the results after 10 minutes of simulation of the networks.

4.3 Case 3 Implementation:

For case 3 implementation we use Network simulator 2 tool.

As the NS-2 uses both TCL and OTCL and c++ objects in the programming to create the simulation.

The malicious node will drop the packets and stops its connection to respond to the other nodes.



```
harishk@ubuntu:~$ cd ns-allinone-2.35/
harishk@ubuntu:~/ns-allinone-2.35$ ns first.tcl
num_nodes is set 6
INITIALIZE THE LIST xListHead
using backward compatible Agent/CBR; use Application/Traffic/CBR instead
Start of simulation..
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
harishk@ubuntu:~/ns-allinone-2.35$
```

Figure 4.3 Compiling the tcl code

In order to compile the NS2 program , we need to install the ns-allinone-2.35 packages from the network simulator website.

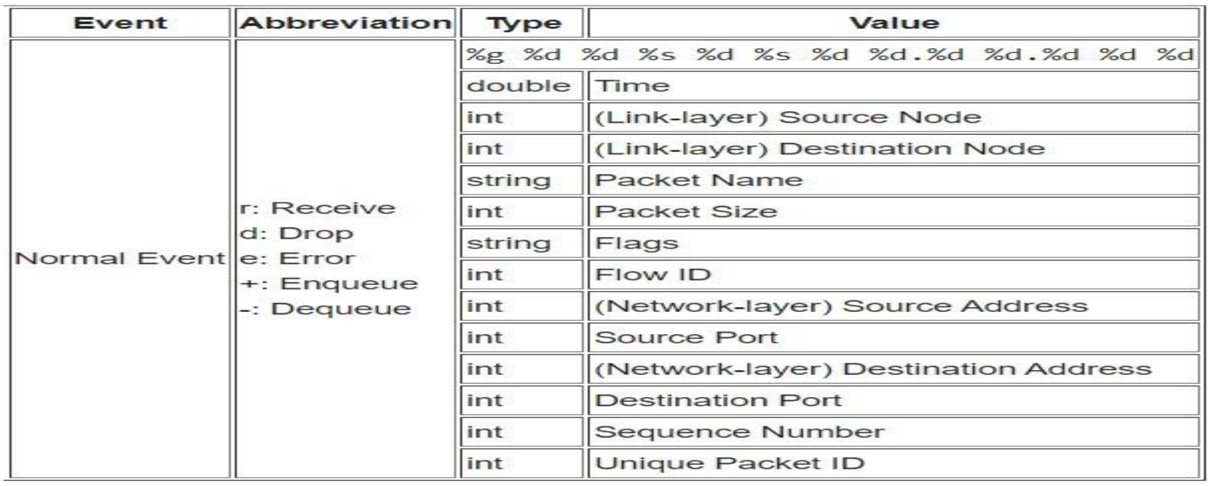
The packages will provide the network animator and Xgraph applications . Both comes in built from the packages.

After we compile and execute the code in the terminal, we get two files they are Nam file and trace file.

Nam file is used by the network animator to display the network output.

Trace files contain packets being sent, received, or only dropped, the so-called packet traces. In fact, the trace files are just a text format in other words, they are human readable, where we can access them using some form of offline software.

These trace files will be analyzed in the below format:



Event	Abbreviation	Type	Value
Normal Event	r: Receive d: Drop e: Error +: Enqueue -: Dequeue	%g %d %d %s %d %s %d %d.%d %d.%d %d %d	
		double	Time
		int	(Link-layer) Source Node
		int	(Link-layer) Destination Node
		string	Packet Name
		int	Packet Size
		string	Flags
		int	Flow ID
		int	(Network-layer) Source Address
		int	Source Port
		int	(Network-layer) Destination Address
		int	Destination Port
		int	Sequence Number
		int	Unique Packet ID

Table -1 NS-2 Trace file format

CHAPTER 5
SOFTWARE IMPLEMENTATION

5.1 Creating an application in Contiki

Step 1

Go to the directory contiki/examples and make new sub directory.

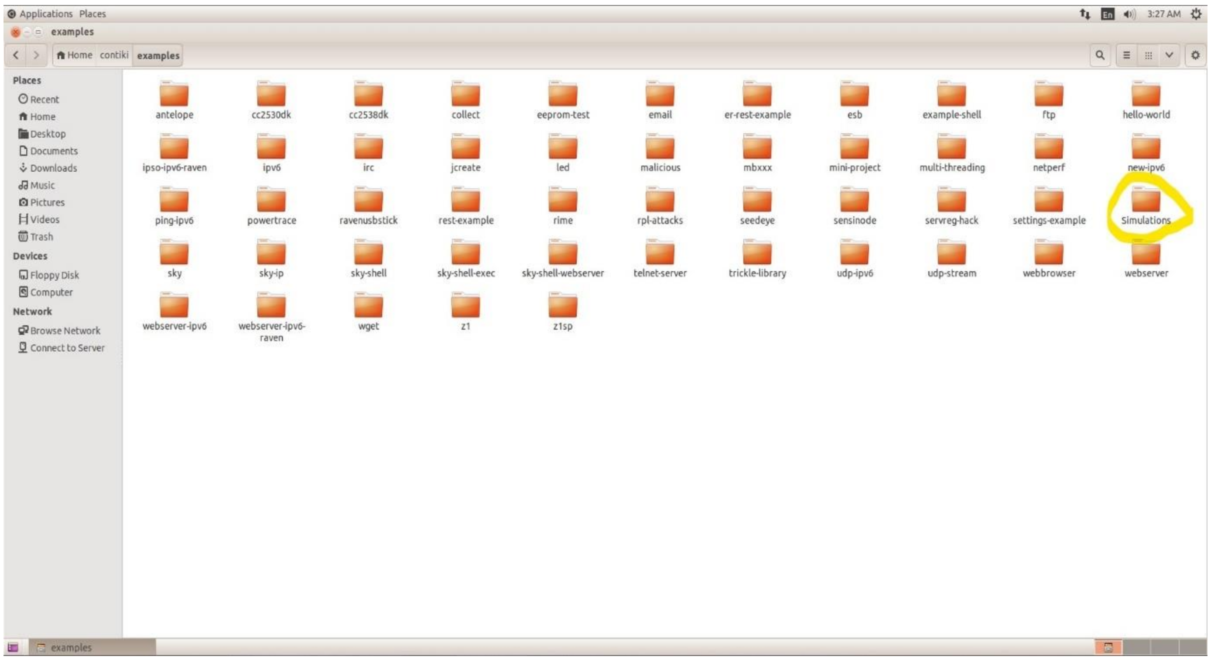


Figure 5.1a creating a new sub directory

Step 2

Now create a new C file named helloworld.c

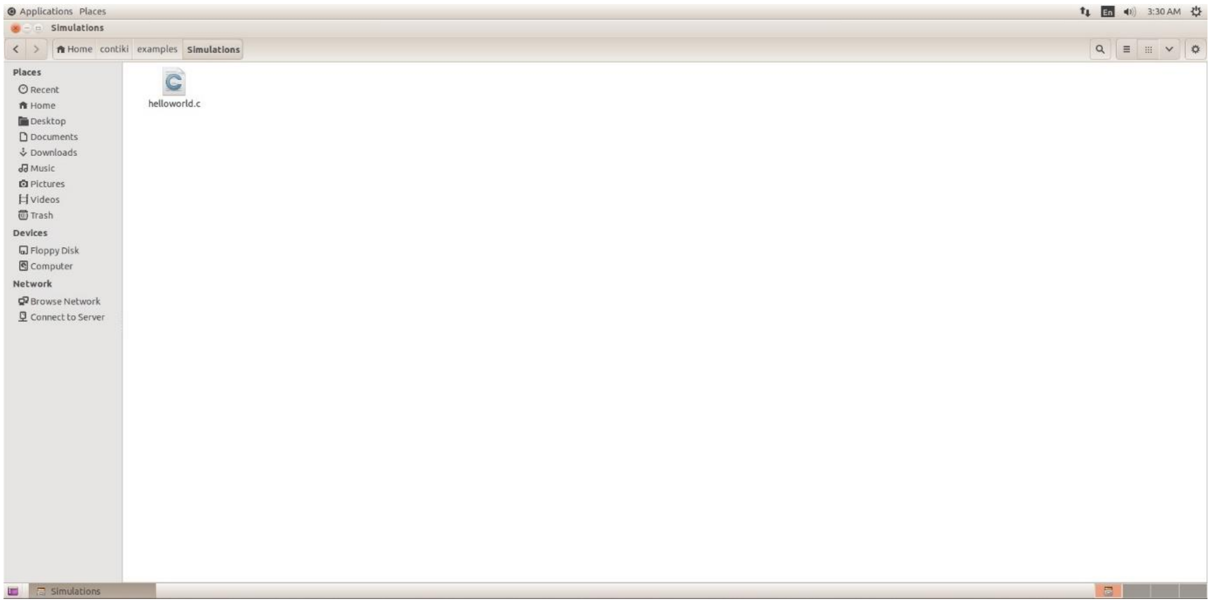


Figure 5.1b create a C file

Step 3

Write helloworld.c

```
/*Header files */
#include "contiki.h"

#include <stdio.h> /* For printf () */
/*---------------------------------------------------------------------------*/
PROCESS (hello_world_process, "Hello world process");
AUTOSTART_PROCESSES(&hello_world_process);
/*---------------------------------------------------------------------------*/
PROCESS_THREAD (hello_world_process, ev, data)
{
/* Begin Process */
  PROCESS_BEGIN();
/* set of C statements */
  printf("Hello, world\n");

/* Process End */
  PROCESS_END();
}
```

Every process in Contiki should start with the PROCESS macro. It takes two arguments

- name: The variable name of the process structure.
- strname: The string representation of the process name.

```
PROCESS(name, strname)
```

Then comes another macro AUTOSTART_PROCESS(struct process &). AUTOSTART_PROCESSES automatically starts the process(es) given in the argument(s) when the module boots.

- &name : Reference to the process name.

```
AUTOSTART_PROCESS(struct process &)
```

Then we call the PROCESS_THREAD function. This function is used to define the protothread of a process. The process is called whenever an event occurs in the system. Each process in the module requires an event handler under the PROCESS_THREAD macro.

- name: The variable name of the process structure.
- process_event_t: The variable of type character. If this variable is same as PROCESS_EVENT_EXIT then PROCESS_EXITHANDLER is invoked.

PROCESS _THREAD(name, process_event_t, process_data_t)

Then comes the PROCESS_BEGIN macro. This macro defines the beginning of a process and must always appear in a PROCESS_THREAD() definition.

PROCESS_BEGIN()

Then we write the set of C statements as per the requirement of the application.

At the end we use another macro called PROCESS_END. This macro defines the end of a process. It must appear in a PROCESS_THREAD() definition and must always be included. The process exits when the PROCESS_END() macro is reached.

PROCESS_END()

Step 4

Makefile - Create makefile in the same folder (contiki/exmples/helloworld)

```
CONTIKI_PROJECT = simulations
All: $(CONTIKI_PROJECT)
#UIP_CONF_IPV6=1
CONTIKI = ../..
Include $(CONTIKI)/Makefile.include
```

Step 5

Create a file Makefile.target and then type TARGET = sky

Step 6

Compilation - Use terminal to go to "contiki/examples/simulations" directory. Once you are in this directory type "make". This will compile your code and generates all the supporting files like .csc file, symbols.c, symbols.h etc.

5.2 Generic structure of Application in Contiki

```
<Header Files>
PROCESS(name, strname);
AUTOSTART_PROCESS(struct process &);
PROCESS_THREAD(name, process_event_t, process_data_t)
{
---------Initialization of required variables------------
PROCESS_BEGIN();

--Set of C statements----

PROCESS_END();
}
```

5.3 UDP SENDER Code for creating Sender nodes

```
#include "contiki.h"

#include "net/uip.h"

#include "net/uip-ds6.h"

#include "net/uip-udp-packet.h"

#include "net/rpl/rpl.h"

#include "dev/serial-line.h"

#if CONTIKI_TARGET_Z1

#include "dev/uart0.h"

#else

#include "dev/uart1.h"

#endif

#include "collect-common.h"

#include "collect-view.h"

#include <stdio.h>

#include <string.h>

#define UDP_CLIENT_PORT 8775

#define UDP_SERVER_PORT 5688

#define DEBUG DEBUG_PRINT

#include "net/uip-debug.h"

static struct uip_udp_conn *client_conn;
static uip_ipaddr_t server_ipaddr;
/*---------------------------------------------------------------------------*/
PROCESS(udp_client_process, "UDP client process");
AUTOSTART_PROCESSES(&udp_client_process, &collect_common_process);
/*---------------------------------------------------------------------------*/
void
collect_common_set_sink(void)
{
  /* A udp client can never become sink */
}
/*---------------------------------------------------------------------------*/
void
collect_common_net_print(void)
{
  rpl_dag_t *dag;
  uip_ds6_route_t *r;
  /* Let's suppose we have only one instance */
  dag = rpl_get_any_dag();
  if(dag->preferred_parent != NULL) {
    PRINTF("Preferred parent: ");
    PRINT6ADDR(rpl_get_parent_ipaddr(dag->preferred_parent));
    PRINTF("\n");
  }
  for(r = uip_ds6_route_head();
      r != NULL;
      r = uip_ds6_route_next(r)) {
    PRINT6ADDR(&r->ipaddr);
  }
  PRINTF("---\n");
}
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  if(uip_newdata()) {
    /* Ignore incoming data */
  }
}
/*---------------------------------------------------------------------------*/
void
collect_common_send(void)
{
  static uint8_t seqno;
  struct {
    uint8_t seqno;
    uint8_t for_alignment;
    struct collect_view_data_msg msg;
  } msg;
  /* struct collect_neighbor *n; */
  uint16_t parent_etx;
```

```
uint16_t rtmetric;
uint16_t num_neighbors;
uint16_t beacon_interval;
rpl_parent_t *preferred_parent;
rimeaddr_t parent;
rpl_dag_t *dag;
if(client_conn == NULL) {
  /* Not setup yet */
  return;
}
memset(&msg, 0, sizeof(msg));
seqno++;
if(seqno == 0) {
  /* Wrap to 128 to identify restarts */
  seqno = 128;
}
msg.seqno = seqno;
rimeaddr_copy(&parent, &rimeaddr_null);
parent_etx = 0;
/* Let's suppose we have only one instance */
dag = rpl_get_any_dag();
if(dag != NULL) {
  preferred_parent = dag->preferred_parent;
  if(preferred_parent != NULL) {
    uip_ds6_nbr_t *nbr;
```

```
      nbr = uip_ds6_nbr_lookup(rpl_get_parent_ipaddr(preferred_parent));
      if(nbr != NULL) {
        /* Use parts of the IPv6 address as the parent address, in reversed byte order. */
        parent.u8[RIMEADDR_SIZE - 1] = nbr->ipaddr.u8[sizeof(uip_ipaddr_t) - 2];
        parent.u8[RIMEADDR_SIZE - 2] = nbr->ipaddr.u8[sizeof(uip_ipaddr_t) - 1];
        parent_etx = rpl_get_parent_rank((rimeaddr_t *) uip_ds6_nbr_get_ll(nbr)) / 2;
      }
    }
    rtmetric = dag->rank;
    beacon_interval = (uint16_t) ((2L << dag->instance->dio_intcurrent) / 1000);
    num_neighbors = RPL_PARENT_COUNT(dag);
  } else {
    rtmetric = 0;
    beacon_interval = 0;
    num_neighbors = 0;
  }

  /* num_neighbors = collect_neighbor_list_num(&tc.neighbor_list); */
  collect_view_construct_message(&msg.msg, &parent,
                                 parent_etx, rtmetric,
                                 num_neighbors, beacon_interval);

  uip_udp_packet_sendto(client_conn, &msg, sizeof(msg),
                        &server_ipaddr, UIP_HTONS(UDP_SERVER_PORT));
}
```

```
/*---------------------------------------------------------------------------*/
void
collect_common_net_init(void)
{
#if CONTIKI_TARGET_Z1
  uart0_set_input(serial_line_input_byte);
#else
  uart1_set_input(serial_line_input_byte);
#endif
  serial_line_init();
}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;

  PRINTF("Client IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(uip_ds6_if.addr_list[i].isused &&
       (state == ADDR_TENTATIVE || state == ADDR_PREFERRED)) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if (state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}
/*---------------------------------------------------------------------------*/
static void
set_global_address(void)
{
  uip_ipaddr_t ipaddr;

  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
  uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr);
  uip_ds6_addr_add(&ipaddr, 0, ADDR_AUTOCONF);
  /* set server address */
  uip_ip6addr(&server_ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_client_process, ev, data)
{
  PROCESS_BEGIN();
  PROCESS_PAUSE();
  set_global_address();
  PRINTF("UDP client process started\n");
  print_local_addresses();
  /* new connection with remote host */
  client_conn = udp_new(NULL, UIP_HTONS(UDP_SERVER_PORT), NULL);
  udp_bind(client_conn, UIP_HTONS(UDP_CLIENT_PORT));
  PRINTF("Created a connection with the server ");
  PRINT6ADDR(&client_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n",
        UIP_HTONS(client_conn->lport), UIP_HTONS(client_conn->rport));
  while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    }
  }
  PROCESS_END();
}
```

5.4 UDP Sink Code for creating Sink node

```
#include "contiki.h"
#include "contiki-lib.h"
#include "contiki-net.h"
#include "net/uip.h"
#include "net/rpl/rpl.h"
#include "net/rime/rimeaddr.h"
```

```
#include "net/netstack.h"

#include "dev/button-sensor.h"

#include "dev/serial-line.h"

#if CONTIKI_TARGET_Z1

#include "dev/uart0.h"

#else

#include "dev/uart1.h"

#endif

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#include "collect-common.h"

#include "collect-view.h"

#define DEBUG DEBUG_PRINT

#include "net/uip-debug.h"

#define UIP_IP_BUF   ((struct uip_ip_hdr *)&uip_buf[UIP_LLH_LEN])

#define UDP_CLIENT_PORT 8775

#define UDP_SERVER_PORT 5688

static struct uip_udp_conn *server_conn;

PROCESS(udp_server_process, "UDP server process");

AUTOSTART_PROCESSES(&udp_server_process,&collect_common_process);

/*---------------------------------------------------------------------------*/

void

collect_common_set_sink(void)
```

```
{
}
/*---------------------------------------------------------------------------*/
void
collect_common_net_print(void)
{
  printf("I am sink!\n");
}
/*---------------------------------------------------------------------------*/
void
collect_common_send(void)
{
  /* Server never sends */
}
/*---------------------------------------------------------------------------*/
void
collect_common_net_init(void)
{
#if CONTIKI_TARGET_Z1
  uart0_set_input(serial_line_input_byte);
#else
  uart1_set_input(serial_line_input_byte);
#endif
  serial_line_init();
  PRINTF("I am sink!\n");
}
/*---------------------------------------------------------------------------*/
static void
tcpip_handler(void)
{
  uint8_t *appdata;
  rimeaddr_t sender;
  uint8_t seqno;
  uint8_t hops;
  if(uip_newdata()) {
    appdata = (uint8_t *)uip_appdata;
    sender.u8[0] = UIP_IP_BUF->srcipaddr.u8[15];
    sender.u8[1] = UIP_IP_BUF->srcipaddr.u8[14];
    seqno = *appdata;
    hops = uip_ds6_if.cur_hop_limit - UIP_IP_BUF->ttl + 1;
    collect_common_recv(&sender, seqno, hops,
                        appdata + 2, uip_datalen() - 2);
  }
}
/*---------------------------------------------------------------------------*/
static void
print_local_addresses(void)
{
  int i;
  uint8_t state;
  PRINTF("Server IPv6 addresses: ");
  for(i = 0; i < UIP_DS6_ADDR_NB; i++) {
    state = uip_ds6_if.addr_list[i].state;
    if(state == ADDR_TENTATIVE || state == ADDR_PREFERRED) {
      PRINT6ADDR(&uip_ds6_if.addr_list[i].ipaddr);
      PRINTF("\n");
      /* hack to make address "final" */
      if (state == ADDR_TENTATIVE) {
        uip_ds6_if.addr_list[i].state = ADDR_PREFERRED;
      }
    }
  }
}
/*---------------------------------------------------------------------------*/
PROCESS_THREAD(udp_server_process, ev, data)
{
  uip_ipaddr_t ipaddr;
  struct uip_ds6_addr *root_if;
  PROCESS_BEGIN();
  PROCESS_PAUSE();
  SENSORS_ACTIVATE(button_sensor);
  PRINTF("UDP server started\n");
#if UIP_CONF_ROUTER
  uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 1);
  /* uip_ds6_set_addr_iid(&ipaddr, &uip_lladdr); */
  uip_ds6_addr_add(&ipaddr, 0, ADDR_MANUAL);
  root_if = uip_ds6_addr_lookup(&ipaddr);
  if(root_if != NULL) {
    rpl_dag_t *dag;
    dag = rpl_set_root(RPL_DEFAULT_INSTANCE,(uip_ip6addr_t *)&ipaddr);
    uip_ip6addr(&ipaddr, 0xaaaa, 0, 0, 0, 0, 0, 0, 0);
    rpl_set_prefix(dag, &ipaddr, 64);
    PRINTF("created a new RPL dag\n");
  } else {
    PRINTF("failed to create a new RPL DAG\n");
  }
#endif /* UIP_CONF_ROUTER */
  print_local_addresses();
  /* The data sink runs with a 100% duty cycle in order to ensure high
     packet reception rates. */
  NETSTACK_RDC.off(1);
  server_conn = udp_new(NULL, UIP_HTONS(UDP_CLIENT_PORT), NULL);
  udp_bind(server_conn, UIP_HTONS(UDP_SERVER_PORT));
  PRINTF("Created a server connection with remote address ");
  PRINT6ADDR(&server_conn->ripaddr);
  PRINTF(" local/remote port %u/%u\n", UIP_HTONS(server_conn->lport),
         UIP_HTONS(server_conn->rport));
  while(1) {
    PROCESS_YIELD();
    if(ev == tcpip_event) {
      tcpip_handler();
    } else if (ev == sensors_event && data == &button_sensor) {
      PRINTF("Initiaing global repair\n");
      rpl_repair_root(RPL_DEFAULT_INSTANCE);
    }
  }
  PROCESS_END();
}
```

5.5 TCL code for creating a malicious node in CASE 3

```
# Define options
set val(chan)         Channel/WirelessChannel  ;# channel type
set val(prop)         Propagation/TwoRayGround ;# radio-propagation model
set val(ant)          Antenna/OmniAntenna      ;# Antenna type
set val(ll)           LL                       ;# Link layer type
set val(ifq)          Queue/DropTail/PriQueue  ;# Interface queue type
set val(ifqlen)       50                       ;# max packet in ifq
set val(netif)        Phy/WirelessPhy          ;# network interface type
set val(mac)          Mac/802_11               ;# MAC type
set val(nn)           6                        ;# number of mobilenodes
set val(rp)           AODV                     ;# routing protocol
set val(x)            800
set val(y)            800
set ns [new Simulator]
#ns-random 0
```

```
set f [open out.tr w]

$ns trace-all $f

set namtrace [open out.nam w]

$ns namtrace-all-wireless $namtrace $val(x) $val(y)

set topo [new Topography]

$topo load_flatgrid 800 800

create-god $val(nn)

set chan_1 [new $val(chan)]

set chan_2 [new $val(chan)]

set chan_3 [new $val(chan)]

set chan_4 [new $val(chan)]

set chan_5 [new $val(chan)]

set chan_6 [new $val(chan)]

# CONFIGURE AND CREATE NODES

$ns node-config  -adhocRouting $val(rp) \

                -llType $val(ll) \

                -macType $val(mac) \

                -ifqType $val(ifq) \

                -ifqLen $val(ifqlen) \

                -antType $val(ant) \

                -propType $val(prop) \

                -phyType $val(netif) \

                #-channelType $val(chan) \

                -topoInstance $topo \

                -agentTrace ON \
                -routerTrace ON \
                -macTrace ON \
                -movementTrace OFF \
                -channel $chan_1
proc finish {} {
    global ns namtrace
    $ns flush-trace
        close $namtrace
        exec nam -r 5m out.nam &
    exit 0
}
# define color index
$ns color 0 blue
$ns color 1 red
$ns color 2 chocolate
$ns color 3 red
$ns color 4 brown
$ns color 5 tan
$ns color 6 gold
$ns color 7 black
set n(0) [$ns node]
$ns at 0.0 "$n(0) color blue"
$n(0) color "0"
$n(0) shape "circle"
set n(1) [$ns node]
```

```
$ns at 0.0 "$n(1) color red"

$n(1) color "blue"

$n(1) shape "circle"

set n(2) [$ns node]

$n(2) color "tan"

$n(2) shape "circle"

set n(3) [$ns node]

$n(3) color "red"

$n(3) shape "circle"

set n(4) [$ns node]

$n(4) color "tan"

$n(4) shape "circle"

set n(5) [$ns node]

$ns at 0.0 "$n(5) color blue"

$n(5) color "red"

$n(5) shape "circle"

for {set i 0} {$i < $val(nn)} {incr i} {

   $ns initial_node_pos $n($i) 30+i*100

}

$ns at 0.0 "[$n(1) set ragent_] malicious"

$ns at 0.0 "$n(0) setdest 100.0 100.0 3000.0"

$ns at 0.0 "$n(1) setdest 200.0 200.0 3000.0"

$ns at 0.0 "$n(2) setdest 300.0 200.0 3000.0"

$ns at 0.0 "$n(3) setdest 400.0 300.0 3000.0"

$ns at 0.0 "$n(4) setdest 500.0 300.0 3000.0"
```

```
$ns at 0.0 "$n(5) setdest 600.0 400.0 3000.0"

# CONFIGURE AND SET UP A FLOW

set sink0 [new Agent/LossMonitor]

set sink1 [new Agent/LossMonitor]

set sink2 [new Agent/LossMonitor]

set sink3 [new Agent/LossMonitor]

set sink4 [new Agent/LossMonitor]

set sink5 [new Agent/LossMonitor]

$ns attach-agent $n(0) $sink0

$ns attach-agent $n(1) $sink1

$ns attach-agent $n(2) $sink2

$ns attach-agent $n(3) $sink3

$ns attach-agent $n(4) $sink4

$ns attach-agent $n(5) $sink5

#$ns attach-agent $sink2 $sink3

set tcp0 [new Agent/TCP]

$ns attach-agent $n(0) $tcp0

set tcp1 [new Agent/TCP]

$ns attach-agent $n(1) $tcp1

set tcp2 [new Agent/TCP]

$ns attach-agent $n(2) $tcp2

set tcp3 [new Agent/TCP]

$ns attach-agent $n(3) $tcp3

set tcp4 [new Agent/TCP]

$ns attach-agent $n(4) $tcp4
```

```
set tcp5 [new Agent/TCP]

$ns attach-agent $n(5) $tcp5

proc attach-CBR-traffic { node sink size interval } {


   #Get an instance of the simulator

   set ns [Simulator instance]


   #Create a CBR  agent and attach it to the node

   set cbr [new Agent/CBR]

   $ns attach-agent $node $cbr

   $cbr set packetSize_ $size

   $cbr set interval_ $interval


 #Attach CBR source to sink;

   $ns connect $cbr $sink

   return $cbr

  }

set cbr0 [attach-CBR-traffic $n(0) $sink5 1000 .030]

$ns at 0.5 "$cbr0 start"

$ns at 5.5 "finish"

puts "Start of simulation.."

$ns run
```

5.6 Modification need to be done in AODV files for Case 3

We need to modify the code in Both AODV .cc and .h files for case 3 implementation.

In .cc file we need to create a new variable

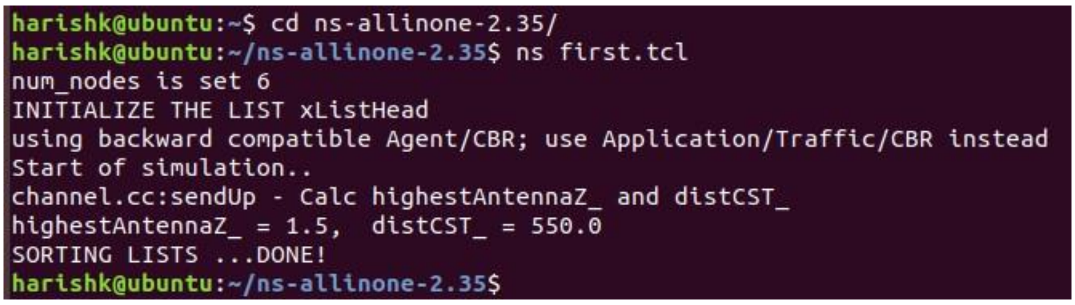
```
bool attacker;
int t_count;
```

And in .h file add the following condition:

```
    If(attacker==true){
Printf(" drop packets");
}
```

After modifying the code again recompile the tcl code and type command as follow:

```
ns first.tcl
```



```
harishk@ubuntu:~$ cd ns-allinone-2.35/
harishk@ubuntu:~/ns-allinone-2.35$ ns first.tcl
num_nodes is set 6
INITIALIZE THE LIST xListHead
using backward compatible Agent/CBR; use Application/Traffic/CBR instead
Start of simulation..
channel.cc:sendUp - Calc highestAntennaZ_ and distCST_
highestAntennaZ_ = 1.5,  distCST_ = 550.0
SORTING LISTS ...DONE!
harishk@ubuntu:~/ns-allinone-2.35$
```

Figure 5.2 Executing the TCL file

CHAPTER 6
RESULTS

6.1 CASE 1 RESULTS:

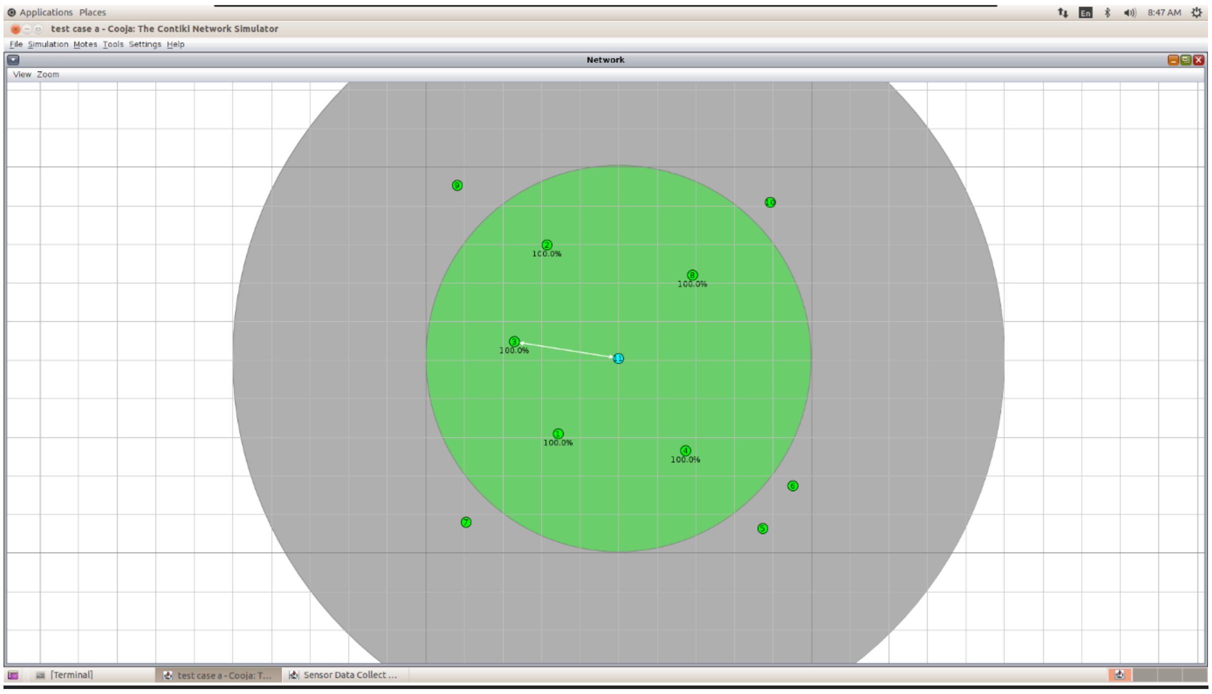

Applications Places
test case a - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
100.0%
[Terminal]
test case a - Cooja: T...
Sensor Data Collect ...

Figure 6.1 CASE 1 Network with 50% nodes inside and outside the range of Sink node

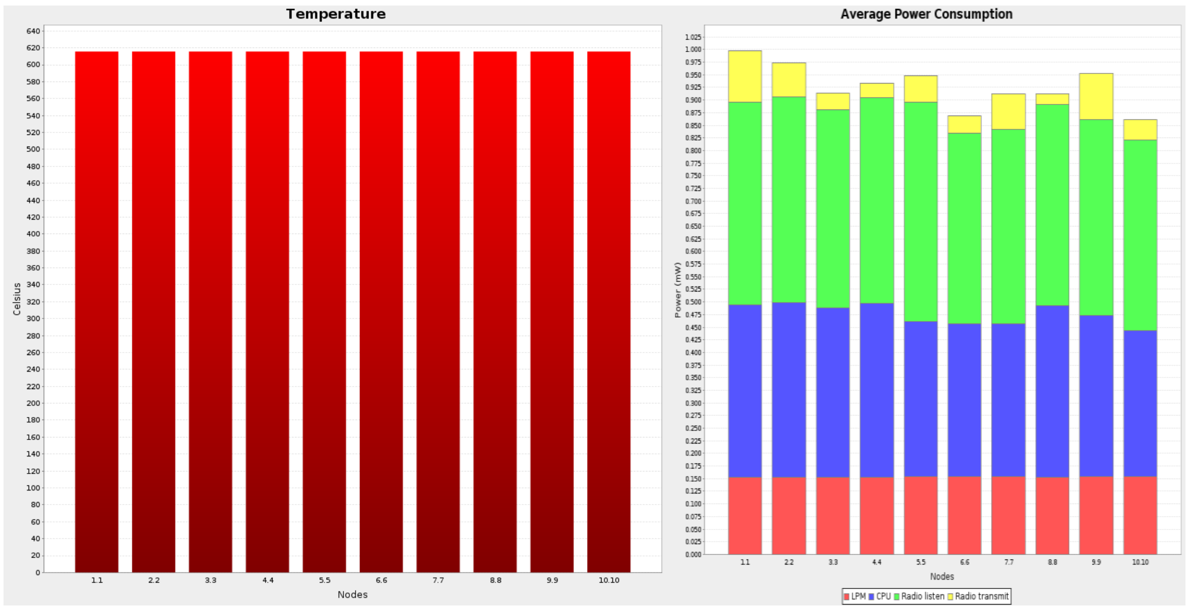


Figure 6.1a Average temperature Figure 6.1b Average Power Consumption

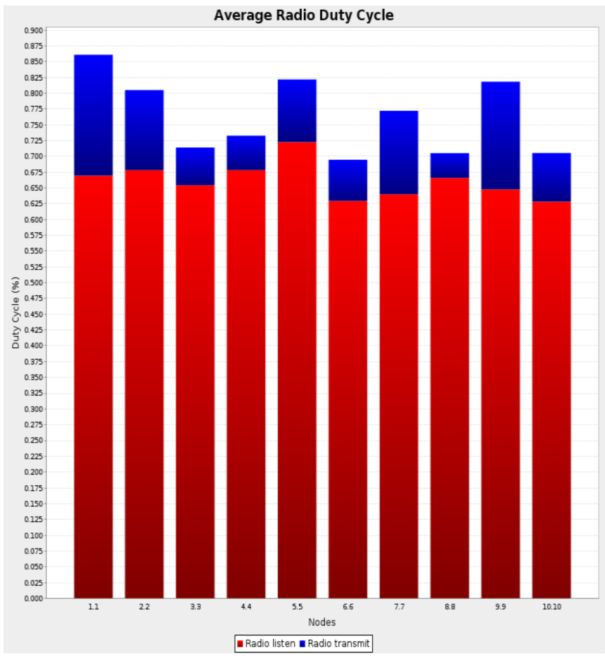


Figure 6.1c Average Radio Duty Cycle

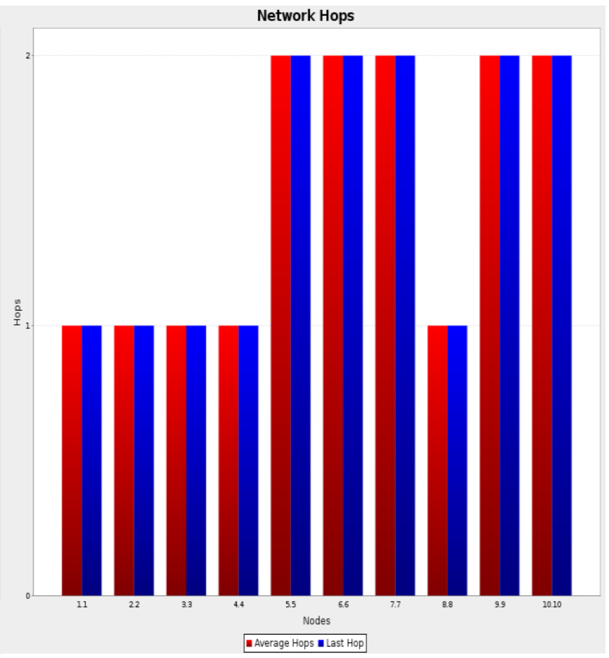


Figure 6.1d Network Hops

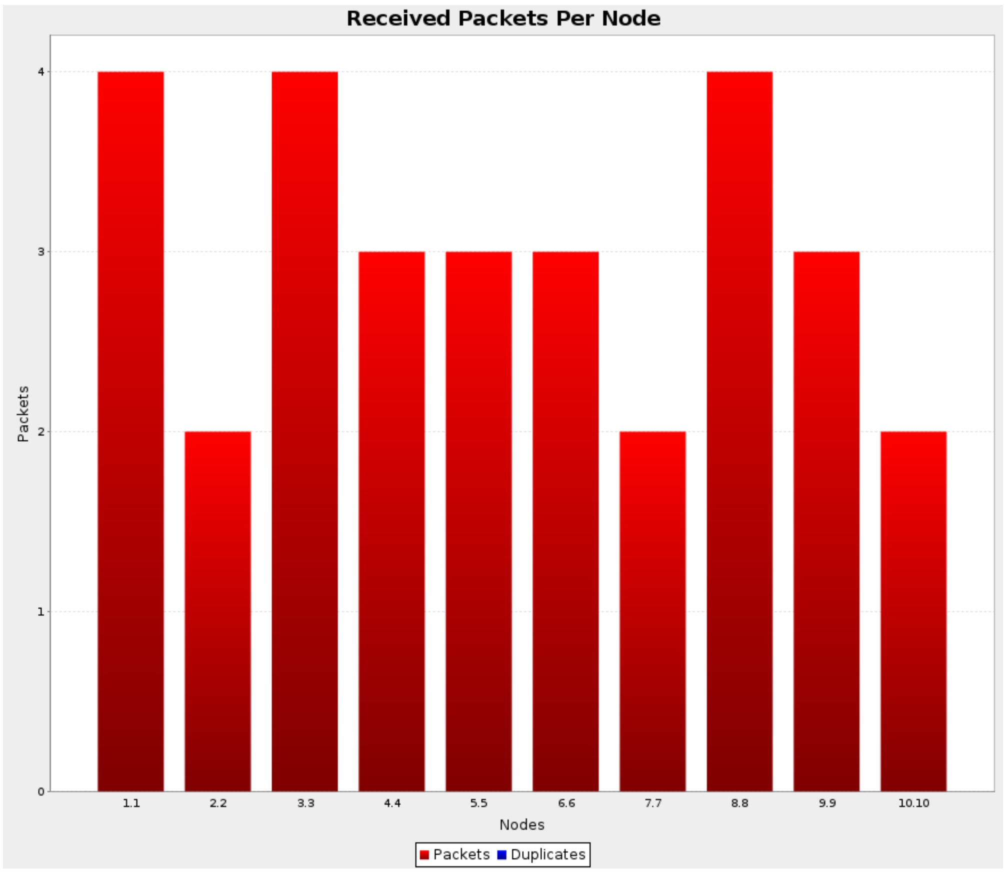


Figure 6.1e Received packets per node

6.2 CASE 2 RESULTS:

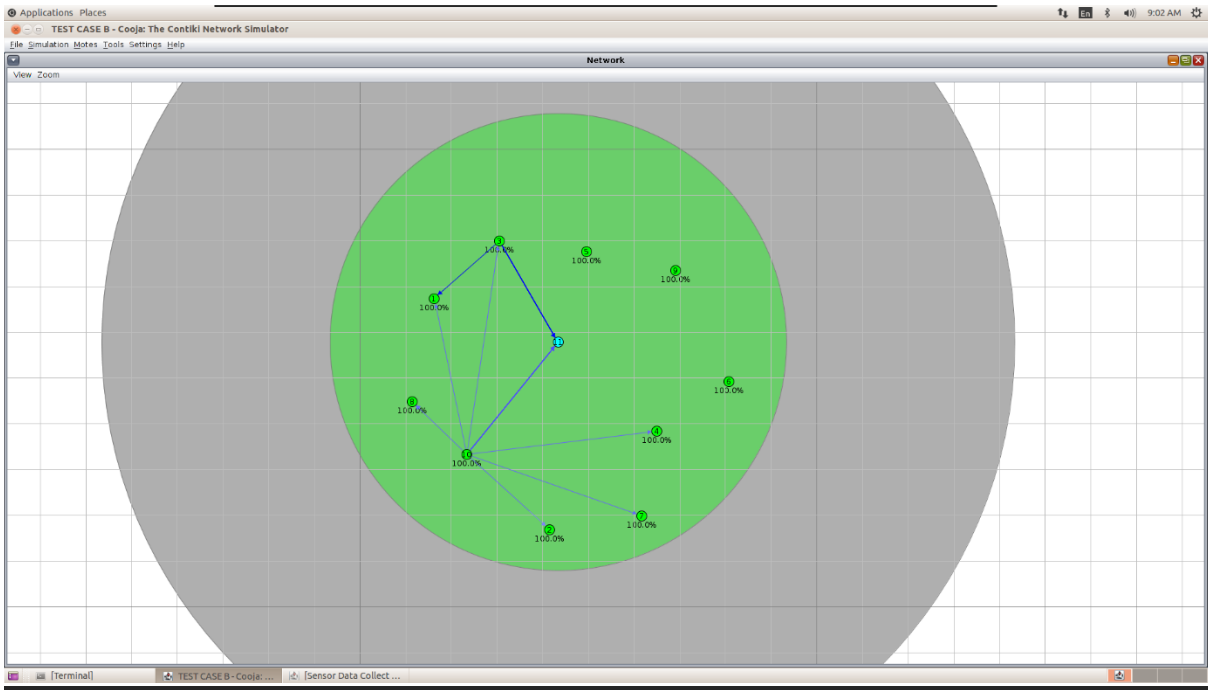


Figure 6.2 CASE 2 Network with 100% Nodes

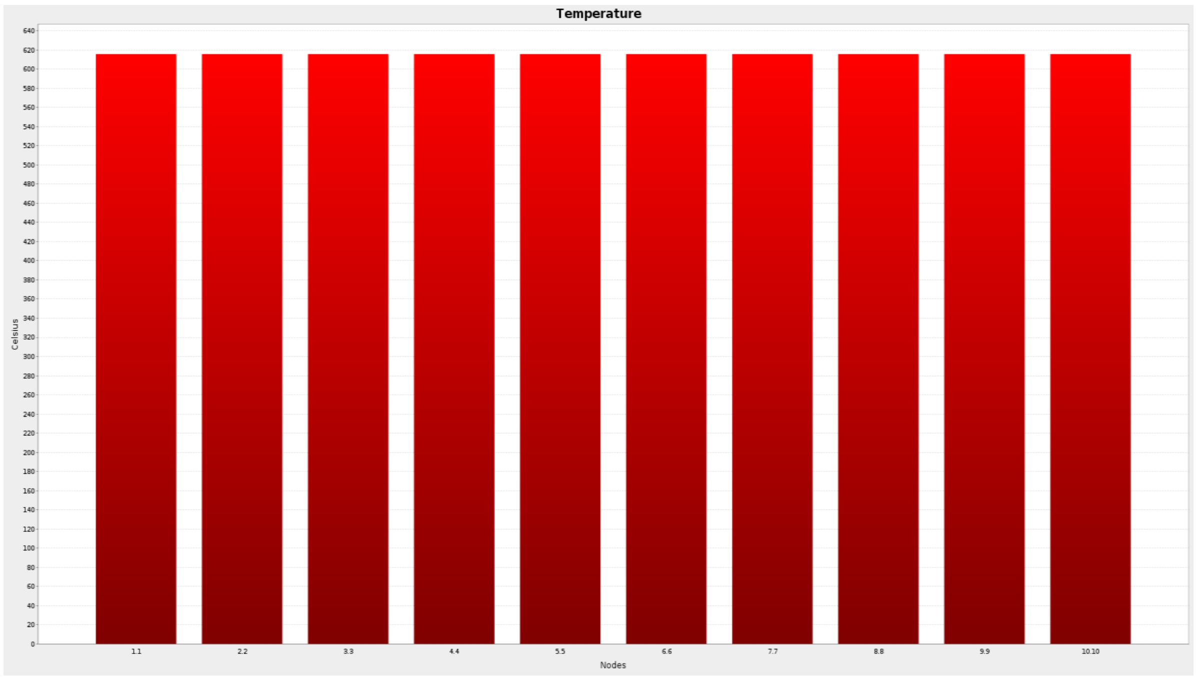


Figure 6.2a Average Temperature(CASE 2)

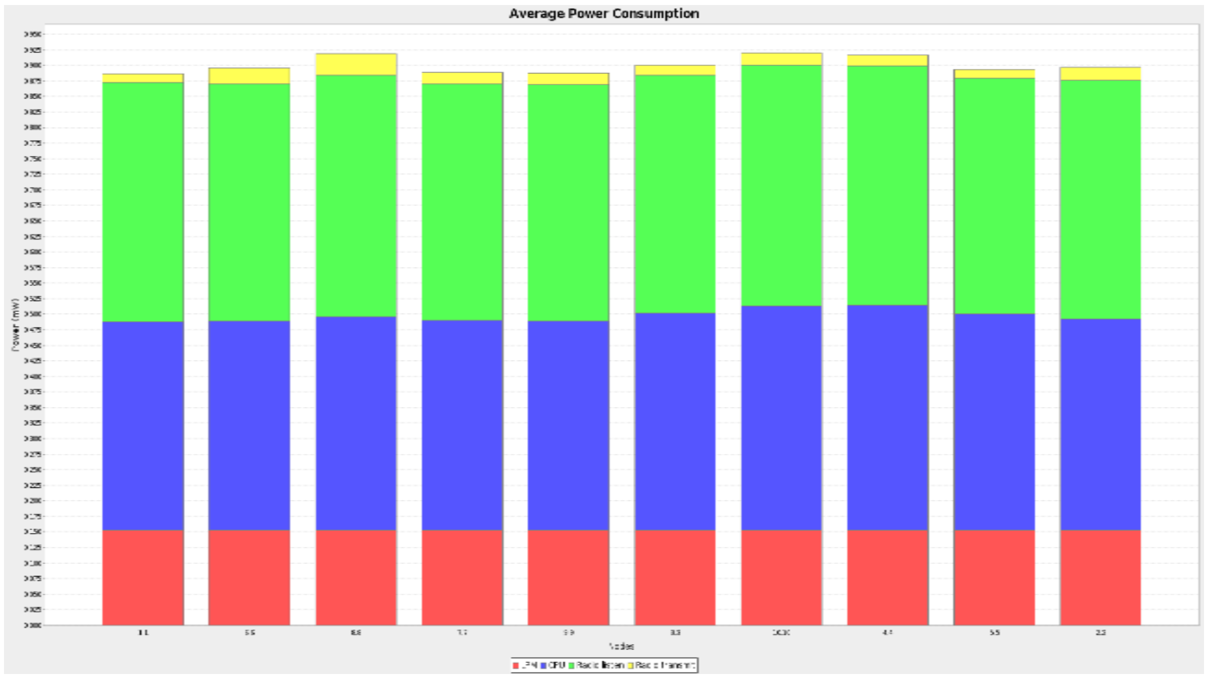


Figure 6.2b Average Power Consumption

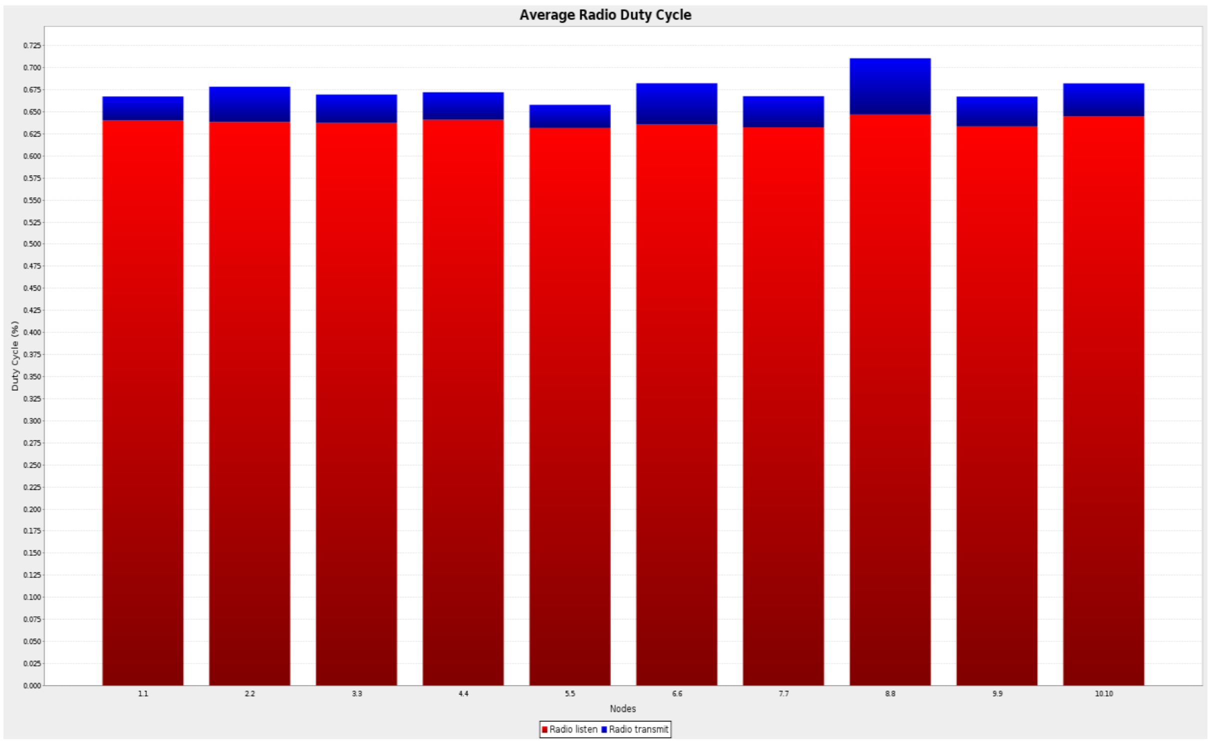


Figure 6.2c Average Radio Duty Cycle(CASE 2)

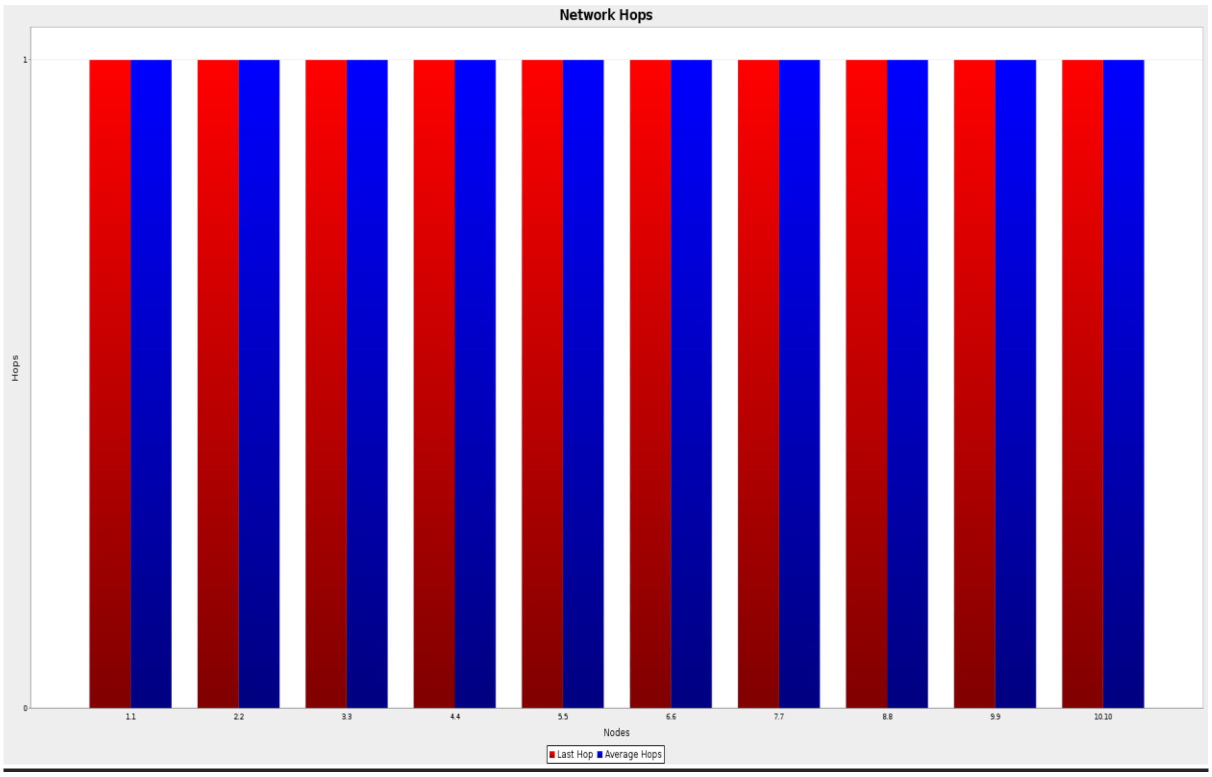


Figure 6.2d Network Hops(CASE 2)

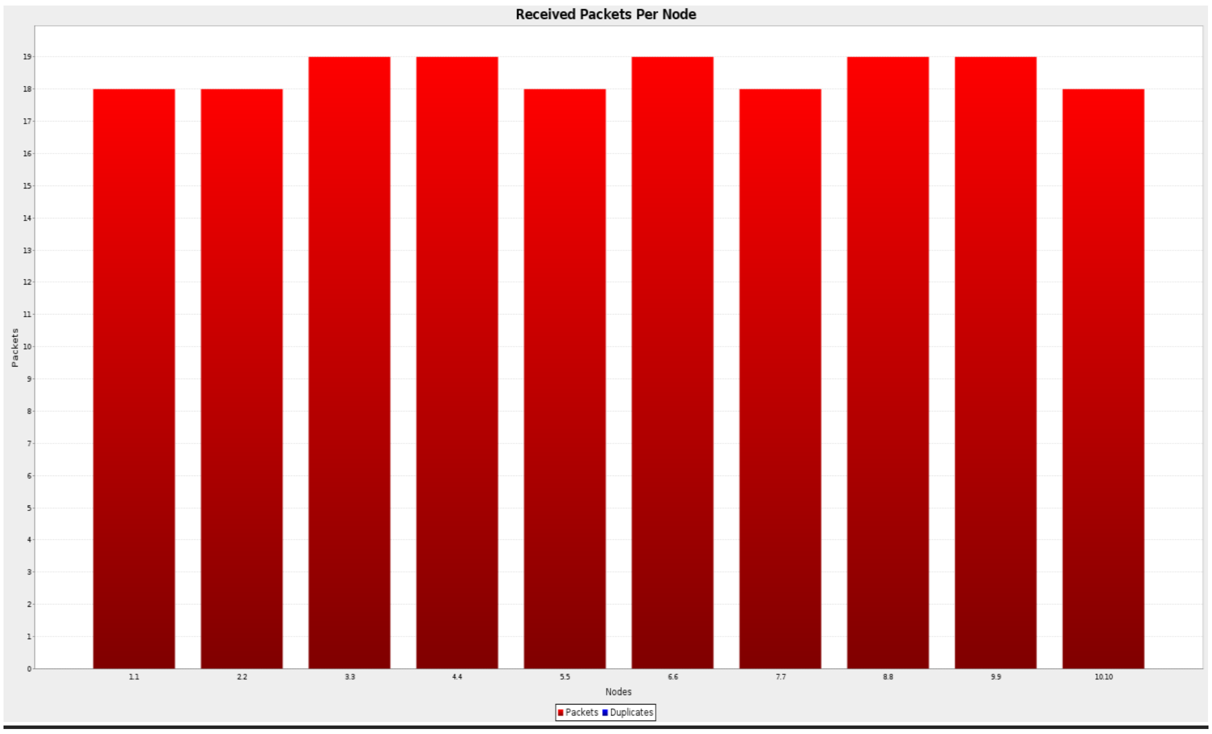


Figure 6.2e Received Packets per node(Case 2)

6.3 CASE 3 RESULTS:

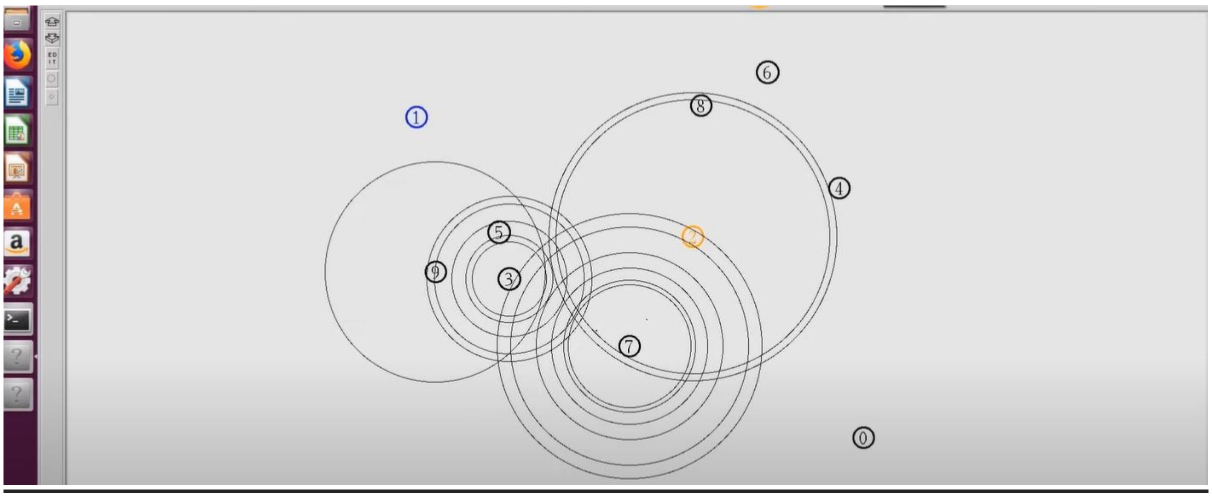


Figure 6.3a Network before malicious node

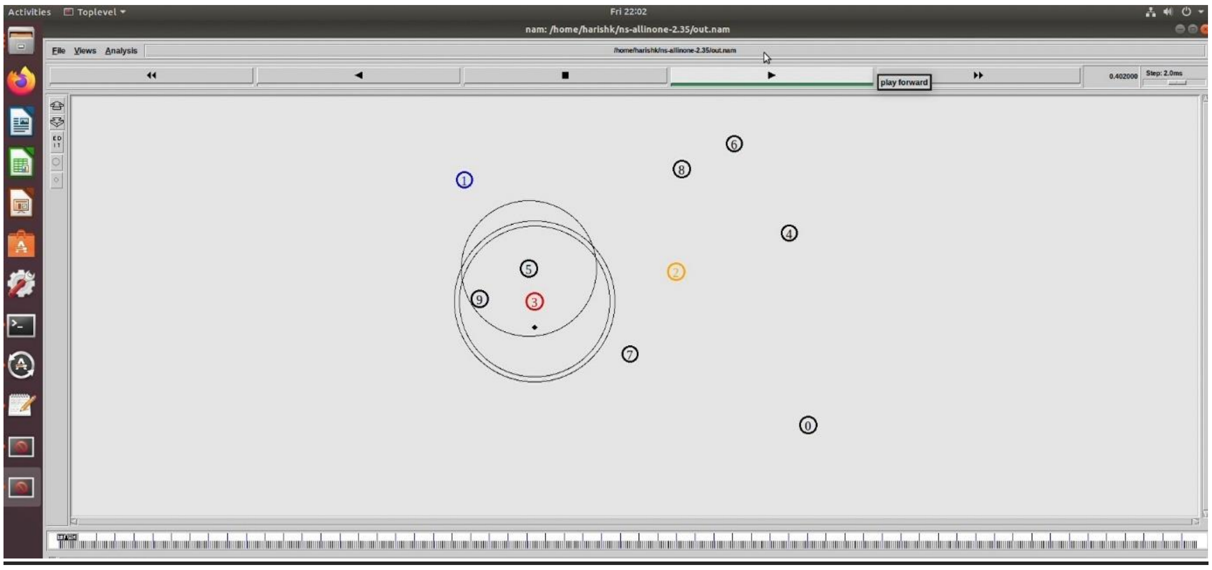
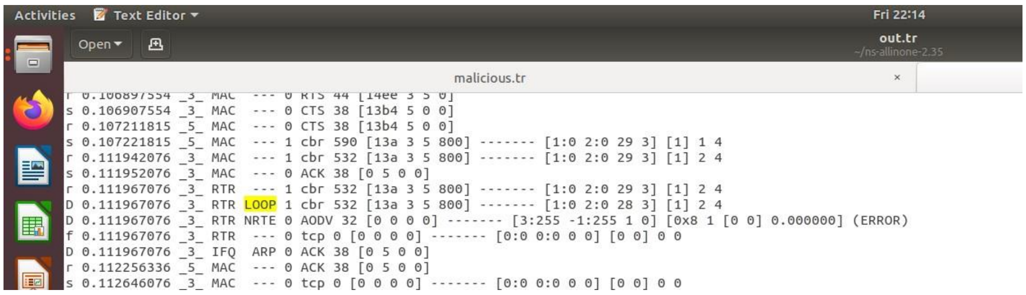


Figure 6.3b Network after adding malicious node and The packets are dropped.

Malicious node number is “3”.

Therefore, Node number 3 is not allowing the packets pass through it and dropping it from the Node.



```
r 0.106897554 _3_ MAC  --- 0 RTS 44 [14ee 3 5 0]
s 0.106907554 _3_ MAC  --- 0 CTS 38 [13b4 5 0 0]
r 0.107211815 _5_ MAC  --- 0 CTS 38 [13b4 5 0 0]
s 0.107221815 _5_ MAC  --- 1 cbr 590 [13a 3 5 800] ------- [1:0 2:0 29 3] [1] 1 4
r 0.111942076 _3_ MAC  --- 1 cbr 532 [13a 3 5 800] ------- [1:0 2:0 29 3] [1] 2 4
s 0.111952076 _3_ MAC  --- 0 ACK 38 [0 5 0 0]
r 0.111967076 _3_ RTR  --- 1 cbr 532 [13a 3 5 800] ------- [1:0 2:0 29 3] [1] 2 4
D 0.111967076 _3_ RTR LOOP 1 cbr 532 [13a 3 5 800] ------- [1:0 2:0 28 3] [1] 2 4
D 0.111967076 _3_ RTR NRTE 0 AODV 32 [0 0 0 0] ------- [3:255 -1:255 1 0] [0x8 1 [0 0] 0.000000] (ERROR)
f 0.111967076 _3_ RTR  --- 0 tcp 0 [0 0 0 0] ------- [0:0 0:0 0 0] [0 0] 0 0
D 0.111967076 _3_ IFQ  ARP 0 ACK 38 [0 5 0 0]
r 0.112256336 _5_ MAC  --- 0 ACK 38 [0 5 0 0]
s 0.112646076 _3_ MAC  --- 0 tcp 0 [0 0 0 0] ------- [0:0 0:0 0 0] [0 0] 0 0
```

Figure 6.3c Out.tr trace file analysis Malicious node found in the Drop event

CHAPTER 7
CONCLUSION

Here we conclude that RPL protocol justifies its purpose of forming a path to the server and client and the intermediate nodes between them forming a complete network. Out of Range Node in the network will take the help of intermediate node to connect with the server and process the Transmission and Reception .

Case 1 was having little difference compared to Case 2 because of the placement of nodes in the range of Sink node. Even though they are placed at different positions, the far away sender node is connected to the sink node with the help of intermediate nodes between them forming a route or path to the destination.

By using Contiki Cooja simulator we are able to find out the differences between each scenario and parameters. As Cooja simulator provides better understanding by displaying the outputs accurately .With the Help of Collect view application we are able to control the timing of the simulation .Not only for the purpose of analysing but also for performing networks related solution for the problem statements arising in IoT network field.

7.1 Future Scope:

There is a lot of scope for Contiki OS and Cooja simulator in simulating networks which are used to analyse the parameters of wireless sensor nodes and can decode the problem for dangerous network attacks and can provide a solution to them. Even though, there are a lot of other network simulator tools present Contiki Cooja simulator provides even better results close to accuracy .

REFERENCES

[1] Contiki - Wikipedia Contiki OS

[2] Running and Testing Applications for Contiki OS using Cooja Simulator. https://www.researchgate.net/publication/327238441_Running_and_Testing_Applications_for_Contiki_OS_Using_Cooja_Simulator

[3] A Performance Evaluation of RPL in Contiki https://www.diva-portal.org/smash/get/diva2:833572/FULLTEXT01.pdf

[4] Simulation of Routing protocol for Low power and Lossy Networks with Cooja simulator. https://www.ijcseonline.org/pub_paper/3-IJCSE-05797.pdf

[5] From Modelling with SysML to Simulation with Contiki Cooja simulator of Wireless Sensor Networks. https://ieeexplore.ieee.org/document/7471294

[6] A comparative analysis of simulation and experimental results on RPL performance. https://ieeexplore.ieee.org/document/8248996

[7] https://ieeexplore.ieee.org/document/411663
Level Sensor Network Simulation with COOJA

[8] https://ieeexplore.ieee.org/document/9122278
A New Method for Intrusion Detection on RPL Routing Protocol using Fuzzy logic.

[9] The Contiki Operating System. https://github.com/contiki-os/contiki

[10] The Cooja Network Simulator. https://github.com/contiki-ng/cooja

[11] The NS2 Based simulation and research on wireless sensor network route protocol. https://ieeexplore.ieee.org/document/5302699