A

Project Report

On

Simulation of Wireless Sensor network using NS3

Submitted In Fulfilment of the Requirement for the Award of the

T.Y.B.Tech in Information Technology

of

Walchand College of Engineering, Sangli

Submitted By:

Apsingkar Subhash (2015BIT210)

Khillare Ramdas (2015BIT219)

Zunjare Saurabh (2014BIT031)

Under the Guidance of:

Mrs.S.B.Kumar

DECLARATION

We, the undersigned, hereby declare that the project report entitled on Simulation of Wireless Sensor Network using NS3 written and submitted by us to **Walchand College of Engineering , Sangli** as a fulfillment for **Mini – Project II** under the guidance of **Mrs. S. B. Kumar** The or original work. empirical results in this project report are based on the data collected by us.

Date : 19th Nov, 2017
Place : WCE, Sangli

Forwarded through:

Mr. Apsingkar Subhash (2015BIT210)

Signature

Mr. Khillare Ramdas (2015BIT219)

Signature

Mr. Zunjare Saurabh (2014BIT031)

Signature

GUIDE :

Mrs.S.B.Kumar

Signature

ABSTRACT

Wireless sensor networks (WSNs) consist of a large number of interconnected distributed sensor nodes to monitor and measure physical and environment conditions, such as humidity, temperature, motion, light, etc. The measured data can be sent to different terminals or end users via a wireless network. Compared to conventional networks, WSNs are small in size and have been widely deployed for many applications. But they face constrained resource challenges in terms of storage, communication range, processing ability and energy. The potential uses of WSNs have been limited in particular due to the lack of energy, since sensor nodes in many application scenarios are required to run several months or years without human interference (e.g. battery changing or recharging), recent research in wireless sensor network has led to various new protocols which are particularly designed for sensor networks. To design these networks, the factors needed to be considered are the coverage area, mobility, power consumption, communication capabilities etc.

To understand the actual data transfer protocol followed and overall working of the Wireless Sensor Network we are going to simulate it by using NS3 and analyse WSN on several parameters.

In this project , we have shown the simulation of WSN . In this project we have created wireless sensor network in network simulator 3.

For this purpose we have used c++ language for script.Alongside,we have calculated the energy of the WSN in NS2.For the analysis of Energy Consumption more clearly we are represented it using graph with the help of gnuplot.

Contents

1.1 Introduction

1.1.1 Wireless sensor network

A wireless sensor network (WSN) is a wireless network consisting of spatially distributed autonomous devices using sensors to monitor physical or environmental conditions. A WSN system incorporates a gateway that provides wireless connectivity back to the wired world and distributed nodes . The wireless protocol you select depends on your application requirements.

Wireless sensor networks (WSNs) have gained worldwide attention in recent years. These sensors are small, with limited processing and computing resources, and they are inexpensive compared to traditional sensors. These sensor nodes can sense, measure, and gather information from the environment and, based on some local decision process, they can

transmit the sensed data to the user. Smart sensor nodes are low power devices equipped with one or more sensors, a processor, memory, a power supply, and a radio. A variety of mechanical, thermal, biological, chemical, optical, and magnetic sensors may be attached to the sensor node to measure properties of the environment. Since the sensor nodes have limited memory and are typically deployed in difficult-to-access locations, a radio is implemented for wireless communication to transfer the data to a base station (e.g., a laptop, a personal hand-held device, or an access point to a fixed infrastructure). Battery is the main power source in a sensor node. Secondary power supply that harvests power from the environment such as solar panels may be added to the node depending on the appropriateness of the environment where the sensor will be deployed.

1.1.2 Wireless sensor network Architecture

A WSN node contains several technical components. These include the radio, battery, micro-controller, analog circuit, and sensor interface. When using WSN radio technology, you must make important trade-offs. In battery-powered systems, higher radio data rates and more frequent radio use consume more power. Often three years of battery life is a requirement, so many of the WSN systems today are based on Zig Bee due to its low-power consumption. Because battery life and power management technology are constantly evolving and because of the available IEEE 802.11 bandwidth, Wi-Fi is an interesting technology.

The second technology consideration for WSN systems is the battery. In addition to long life requirements, you must consider the size and weight of batteries as well as international standards for shipping batteries and battery availability. The low cost and wide availability of carbon zinc and alkaline batteries make them a common choice.

To extend battery life, a WSN node periodically wakes up and transmits data by powering on the radio and then powering it back off to conserve energy. WSN radio technology must efficiently transmit a signal and allow the system to go back to sleep with minimal power use. This means the processor involved must also be able to wake, power up, and return to sleep mode efficiently. Microprocessor trends for WSNs include reducing power consumption while maintaining or increasing processor speed. Much like your radio choice, the power consumption and processing speed trade-off is a key concern when selecting a processor for WSNs. This makes the x86 architecture a difficult option for battery-powered devices.

2.1. Objectives of project….

- To learn the NS3 tool.
- To learn the working of the Wireless Sensor Network.
- Finding the Energy of the Node.
- Plot the energy graph.

2.2. Tools required:

1. NS3

2. NS2

3. Wireshark

4.GNUPlot

1. Network Simulator 3:

NS-*3* has been developed to provide an open, extensible network simulation platform, for networking research and education. In brief, NS-*3* provides models of how packet data networks work and perform, and provides a simulation engine for users to conduct simulation experiments. Some of the reasons to use NS-*3* include to perform studies that are more difficult or not possible to perform with real systems, to study system behavior in a highly controlled, reproducible environment, and to learn about how networks work. Users will note that the available model set in NS-*3* focuses on modeling how Internet protocols and networks work, but NS-*3* is not limited to Internet systems; several users are using NS-*3* to model non-Internet-based systems.

Many simulation tools exist for network simulation studies. Below are a few distinguishing features of NS-*3* in contrast to other tools.

- *NS-3* is designed as a set of libraries that can be combined together and also with other external software libraries. While some simulation platforms provide users with a single, integrated graphical user interface environment in which all tasks are carried out, NS-*3* is more modular in this regard. Several external animators and data analysis and visualization tools can be used with NS-*3*. However, users should expect to work at the command line and with C++ and/or Python software development tools.

- *NS-3* is primarily used on Linux systems, although support exists for FreeBSD, Cygwin (for Windows), and native Windows Visual Studio support is in the process of being developed.
- *NS-3* is not an officially supported software product of any company. Support for NS-*3* is done on a best-effort basis on the NS-3-users mailing list.

2.Network Simulator 2

For those familiar with *ns-2* (a popular tool that preceded *ns-3*), the most visible outward change when moving to *ns-3* is the choice of scripting language. Programs in *ns-2* are scripted in OTcl and results of simulations can be visualized using the Network Animator nam. It is not possible to run a simulation in *ns-2* purely from C++ (i.e., as a main() program without any OTcl). Moreover, some components of *ns-2* are written in C++ and others in OTcl. In *ns-3*, the simulator is written entirely in C++, with optional Python bindings. Simulation scripts can therefore be written in C++ or in Python. New animators and visualizers are available and under current development. Since *ns-3* generates pcap packet trace files, other utilities can be used to analyze traces as well. In this tutorial, we will first concentrate on scripting directly in C++ and interpreting results via trace files.

3. Wireshark

Wireshark is a network packet analyzer. A network packet analyzer will try to capture network packets and tries to display that packet data as detailed as possible.

You could think of a network packet analyzer as a measuring device used to examine what's going on inside a network cable, just like a voltmeter is used by an electrician to examine what's going on inside an electric cable (but at a higher level, of course).

In the past, such tools were either very expensive, proprietary, or both. However, with the advent of Wireshark, all that has changed.

Wireshark is perhaps one of the best open source packet analyzers available today.

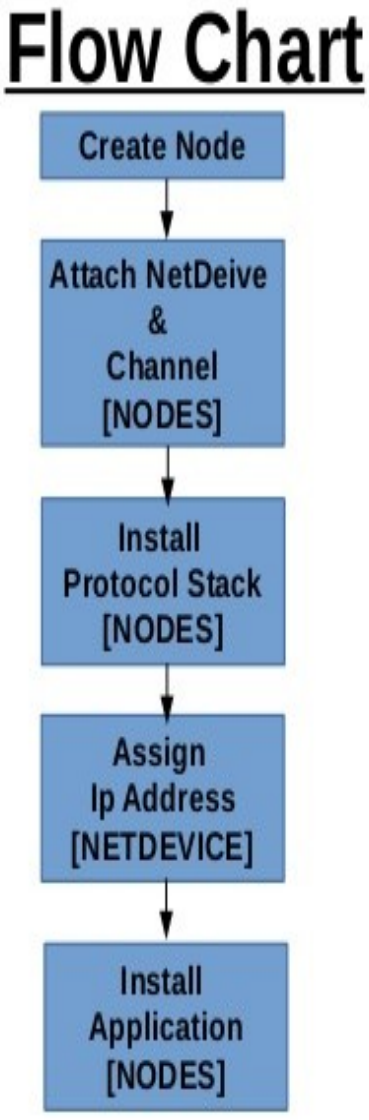
Features:

- *Open* files containing packet data captured with tcpdump/WinDump, Wireshark, and a number of other packet capture programs.
- *Capture* live packet data from a network interface.
- *Import* packets from text files containing hex dumps of packet data.
- Display packets with *very detailed protocol information*.
- *Save* packet data captured.
- *Export* some or all packets in a number of capture file formats.
- *Filter packets* on many criteria.

4. GNUPlot

GNUPlot is a command-line program that can generate two- and three-dimensional plots of functions, data and data fits. It is frequently used for publication-quality graphics as well as education. The program runs on all major computers and operating System Linux,Unix and others. It is a program with a fairly long history, dating back to 1986. Despite its name, this software is not distributed under the GNU General Public License (GPL), but its own free license.

2.3. Structure of project:



1 Node

In Internet jargon, a computing device that connects to a network is called a host or sometimes an end system. Because ns-3 is a network simulator, not specifically an Internet simulator, we intentionally do not use the term host since it is closely associated with the Internet and its protocols. Instead, we use a more generic term also used by other simulators that originates in Graph Theory — the node.

In ns-3 the basic computing device abstraction is called the node. This abstraction is represented in C++ by the class Node. The Node class provides methods for managing the representations of computing devices in simulations.

2.Net Device

It used to be the case that if you wanted to connect a computer to a network, you had to buy a specific kind of network cable and a hardware device called (in PC terminology) a peripheral card that needed to be installed in your computer.

If the peripheral card implemented some networking function, they were called Network Interface Cards, or NICs. Today most computers come with the network interface hardware built in and users don't see these building blocks. A NIC will not work without a software driver to control the hardware. In Unix (or Linux), a piece of peripheral hard-ware is classified as a device. Devices are controlled using device drivers, and

network devices (NICs) are controlled using network device drivers collectively known as net devices. In Unix and Linux you refer to these net devices by names such as eth0.

In ns-3 the net device abstraction covers both the software driver and the simulated hardware. A net device is “in-stalled” in a Node in order to enable the Node to communicate with other Nodes in the simulation via Channels.

Just as in a real computer, a Node may be connected to more than one Channel via multiple NetDevices.The net device abstraction is represented in C++ by the class NetDevice. The NetDevice class provides methods for managing connections to Node and Channel objects; and may be specialized by developers in the object-oriented programming sense. We will use the several specialized versions of the NetDevice called CsmaNetDevice, PointToPointNetDevice, and WifiNetDevice in this tutorial. Just as an Ethernet NIC is designed to work with an Ethernet network, the CsmaNetDevice is designed to work with a CsmaChannel; the PointToPointNetDevice is designed to work with a PointToPointChannel and a WifiNetNevice is designed to work with a WifiChannel.

3.NetDeviceContainer

At this point in the script, we have a NodeContainer that contains two nodes. We have a PointToPointHelper that is primed and ready to make PointToPoint Net Devices and wire PointToPointChannel objects be-tween them. Just as we used the NodeContainer topology helper object to create the Nodes for our simulation,we will ask the PointToPointHelper to do the

work involved in creating, configuring and installing our devicesfor us. We will need to have a list of all of the NetDevice objects that are created, so we use a NetDeviceContainer to hold them just as we used a NodeContainer to hold the nodes we created. The following two lines of code,

```
NetDeviceContainer devices;

devices = pointToPoint.Install (nodes);
```

will finish configuring the devices and channel. The first line declares the device container mentioned above and the second does the heavy lifting. The Install method of the PointToPointHelper takes a NodeContainer as a parameter. Internally, a NetDeviceContainer is created. For each node in the NodeContainer (there must be exactly two for a point-to-point link) a PointToPointNetDevice is created and saved in the device container. A PointToPointChannel is created and the two PointToPointNetDevices are attached. When objects are created by the PointToPointHelper, the Attributes previously set in the helper are used to initialize the corresponding Attributes in the created objects.

After executing the pointToPoint.Install (nodes) call we will have two nodes, each with an installed point-to-point net device and a single point-to-point channel between them. Both devices will be configured to transmit data at five megabits per second over the channel which has a two millisecond transmission delay.

4.InternetStackHelper

We now have nodes and devices configured, but we don't have any protocol stacks installed on our nodes. The next two lines of code will take care of that.

```
InternetStackHelper stack;

stack.Install (nodes);
```

The InternetStackHelper is a topology helper that is to internet stacks what the PointToPointHelper is to point-to-point net devices. The Install method takes a NodeContainer as a parameter. When it is executed, it will install an Internet Stack (TCP, UDP, IP, etc.) on each of the nodes in the node container.

5.Ipv4AddressHelper

Next we need to associate the devices on our nodes with IP addresses. We provide a topology helper to manage the allocation of IP addresses. The only user-visible API is to set the base IP address and network mask to use when performing the actual address allocation (which is done at a lower level inside the helper).

PCAP Tracing

The ns-3 device helpers can also be used to create trace files in the .pcap format. The acronym pcap (usually written in lower case) stands for packet capture, and is actually an API that includes the definition of a .pcap file format. The most popular program that can read and display this format is Wireshark (formerly called Ethereal). However, there are many traffic trace analyzers that use this packet format. We encourage users to exploit the many tools available for analyzing pcap traces. In this tutorial, we concentrate on viewing pcap traces with tcpdump.

Energy Model API

The energy model is used through the node-config API.

The following parameters are newly added:
-sleepPower: power consumption (Watt) in sleep state
-transitionPower: power consumption (Watt) in state transition from sleep to idle (active)
-transitionTime: time (second) used in state transition from sleep to idle (active)

Energy Analysis Through Trace Files

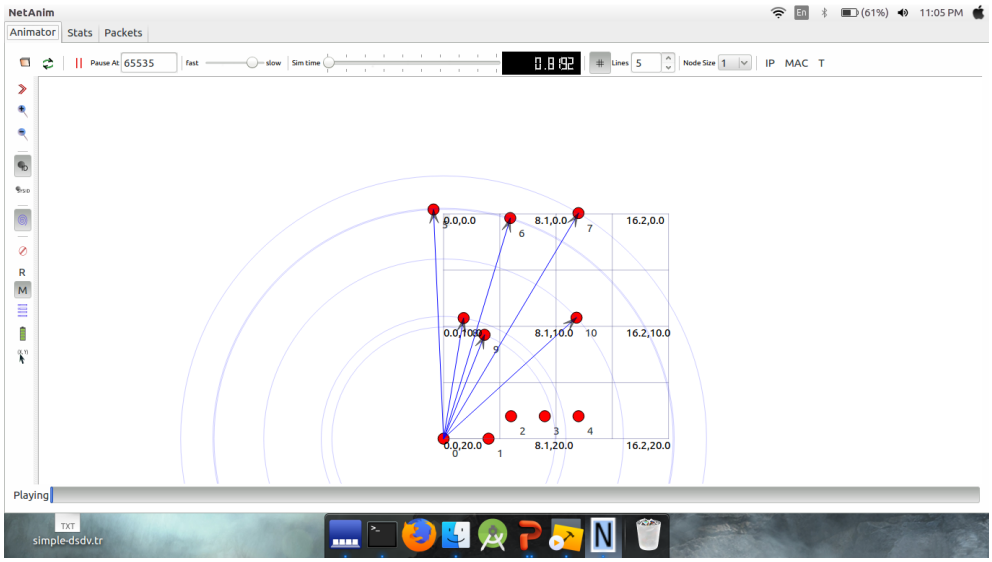
We have added energy breakdown in each state in the traces to support detailed energy analysis. In addition to the total energy, now users will be

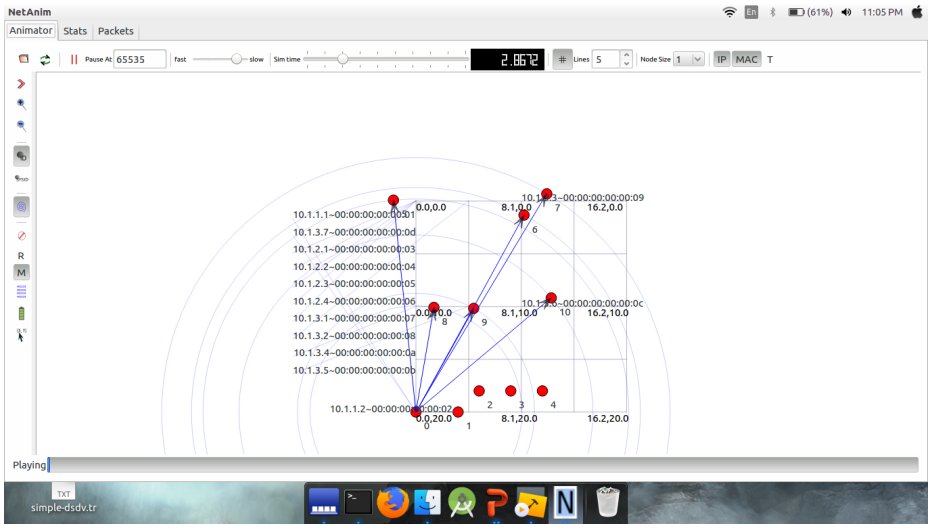
able to see the energy consumption in different states at a given time. Following is an example from a trace file on energy.

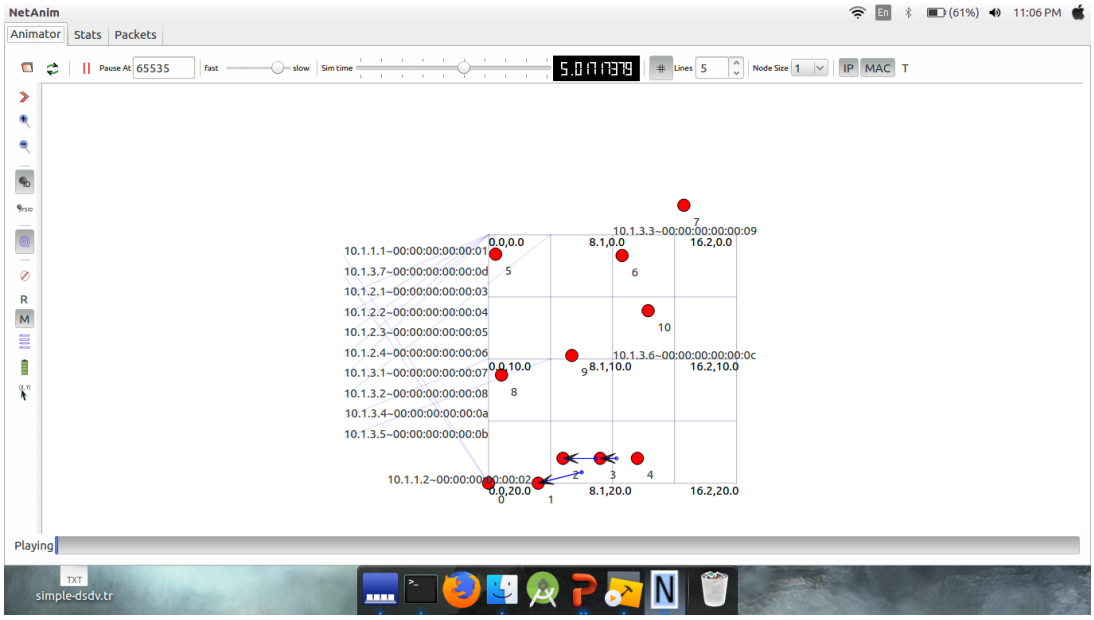
[energy 979.917000 ei 20.074 es 0.000 et 0.003 er 0.006]

The meaning of each item is as follows:
energy: total remaining energy
ei: energy consumption in IDLE state
es: energy consumption in SLEEP state
et: energy consumed in transmitting packets
er: energy consumed in receiving packets

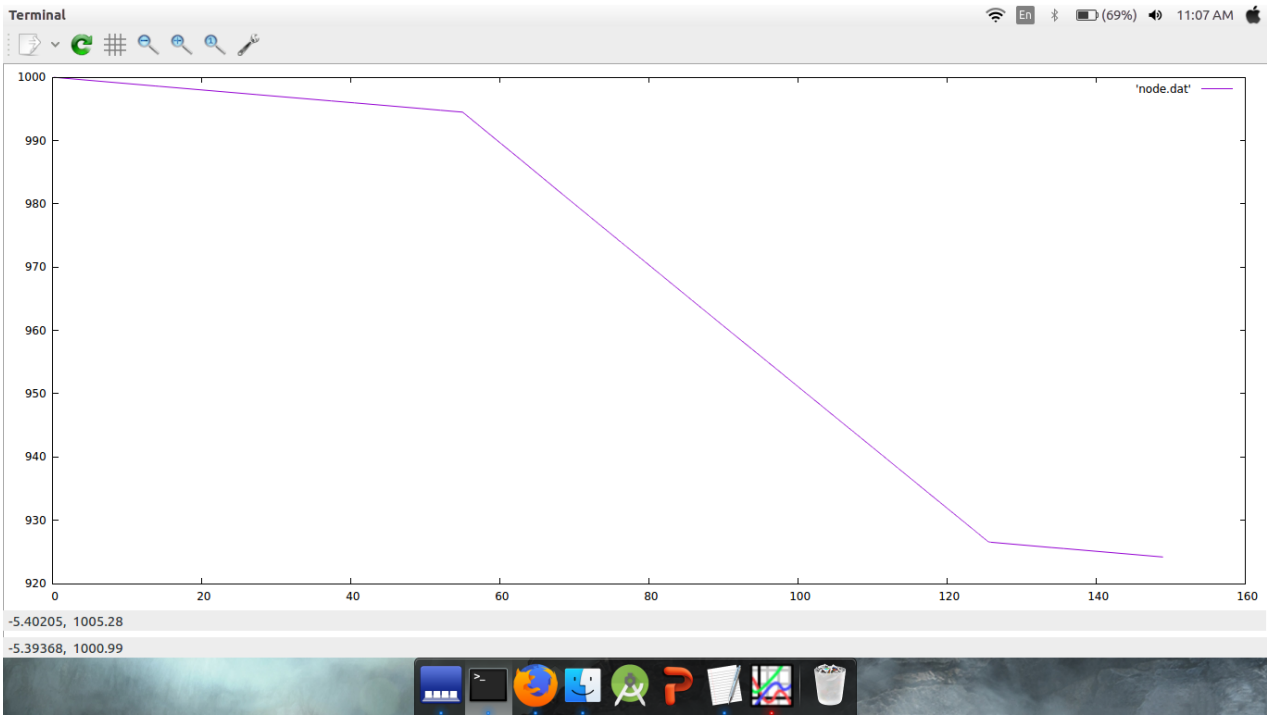
2.4. Simulation of Project







Energy Graph



3. Future Scope:

In Future we will be going to work how to reduce the energy consumption of the Wireless Sensor Network.

4. References

- https://www.google.co.in/search?client=ubuntu&channel=fs&q=WSN&ie=utf-8&oe=utf-8&gfe_rd=cr&ei=DBovWIbnD-uK8QeKzaPQDg
- https://www.nsnam.org/docs/tutorial/html/
- http://www.mashpy.me/2015/01/install-and-configure-network-simulator-ns3-on-ubuntu-14-04.html
- https://www.nsnam.org/docs/release/3.18/tutorial/ns-3-tutorial.pdf
- http://ns3simulation.com/ns3-simulation-examples/
- http://link.springer.com/chapter/10.1007%2F978-3-642-35864-7_5#page-1