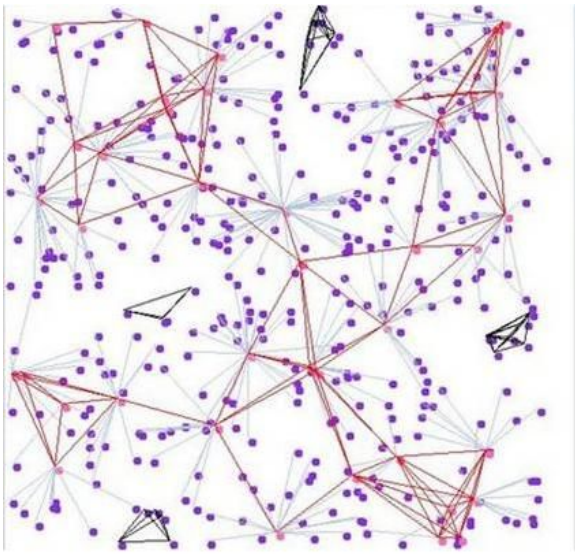
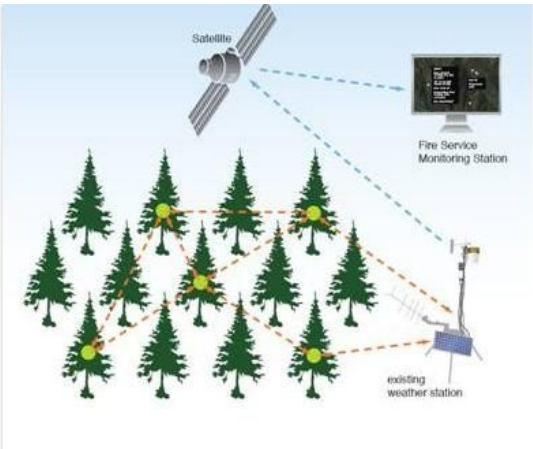
M.Sc. in Computer Science

DISSERTATION

PINAKAS ANTONIOS

Analysis and Simulation of Wireless Sensor Network Models and Growing Scenario Challenges





This dissertation is submitted in partial fulfilment of the requirements of Staffordshire University for the award of M. Sc. in Computer Science

January 2010

Abstract

Optimizing sensor networks involves addressing a wide range of issues steaming from limited energy reserves, computation power, communication capabilities, and self-managing sensor nodes. The ns-2 simulation environment is a flexible tool for network engineers to investigate how various protocols perform with different configurations and topologies [1].Also scientists are able to simulate a network and take the desired results before they implement to the desired area. Through the simulation they can create the environmental conditions that they want and spot possible problems that maybe occur. This project we will show how we will extend network simulator 2 framework so as to create our environmental conditions for the simulation. Also we will work on growing scenarios in a wireless sensor networks. We will add nodes in a wireless network and we will make changes to existing ns-2 code and protocols so as to succeed reduction of energy consumption per node.

Keywords: Network simulator 2, 802.11, zigbee, 802.15.4, environmental conditions, growing scenarios, node addition**.**

Acknowledgements

First of all, I would like to thank my advisor, Dr. Nikolaos Samaras, for his constant support and guidance throughout the course of this work. Also I would like to thank every person that advised me through forums such as www.ubuntuforums.com and of course students from around the world that are using network simulator 2.

I would like to express my thanksgiving to my family for supporting in order to complete my post graduate studies. I could not have succeeded without them.

Table of contents

List of figures

List of tables

1 Introduction

1.1 General background

Wireless sensor networks are useful tool in the hands of scientists so as to collect data of their desire from an area. Sensor networks that detect hazardous chemical or biological agents in complex urban infrastructures could be a killer application for homeland security. We can simulate a network before implement it. We will use network simulator 2 environment so as to simulate different topologies of sensors that communicate 802.15.4 protocols. Also the most important we will create C++ classes so as to extend the ns2 framework and create functions of the simulator that didnût exist through these classes we will describe a propagation model which name is okumura Hata. This model shows us the losses that the signal has when it moves through a dense forest with forest and some buildings. With the word model we describe in general the software code that simulates the communication between nodes in a different environment than the free space model.

1.2 Thesis Objectives

In this project a way to extend ns -2 is presented. As we described we will use several mathematical models that calculate environmental condition. The conditions of the environment, from where the signal passes, have direct effect to the power of the signal. Our will is to create a simulation of sensors in a dense forest with wind and rain. So the next step is to find the correct propagation model that corresponds to the behaviour of the signal while moving to this environment. Some of those models are described below.

It is logical that the signal is absorbed while moving to a not free space area but sometimes the absorption is responsible for bad communication into the network. Our purpose is to find ways to improve this communication because of the losses that our conditions create. A better communication is meaning less lost packets and better throughput values.

Afterwards, we will add some nodes into a network while the nodes that exists already are communicating and we will try to face a challenge that occurs on energy consumption .We will observe through the simulation that when we add nodes into a network (while simulation runs) and the protocol that the

networks use is the AODV , we have a big node energy consumption . This occurs because the criteria of the routing protocol are not the hop's cost of energy but the number of hops to the destination (the most important). In our project we will make some changes to the code of ns-2 ,that are describe below, so as to try to decrease the node consumption to a smaller level.

2 Tools and related work

2.1 Network Simulator 2 Overview

There are a lot of simulators such as omnet++, opnet etc. in our project we will use the ns2 environment. This tool offers great flexibility in investigating the characteristics of sensor networks. As we know it is already contain main important features such as propagation models, communication protocols, flexible models for energy constrained wireless networks. The wireless model also includes support for node movements and energy constraints. By leveraging the existing mobile networking infrastructure, we added the capability to simulate sensor networks. The network simulator (NS is a discrete event simulator for networks and it can supports TCP and UDP packet flows.

. The basic structure of NS-2 is:

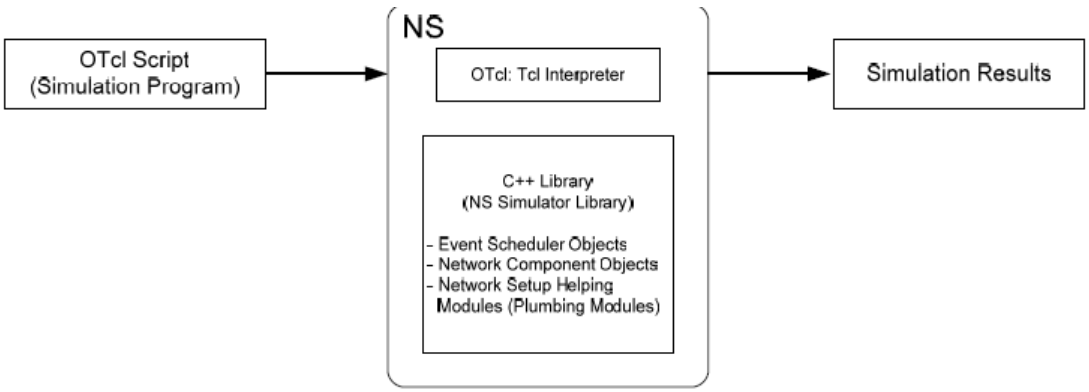


Fig.1 Basic structure of ns2.

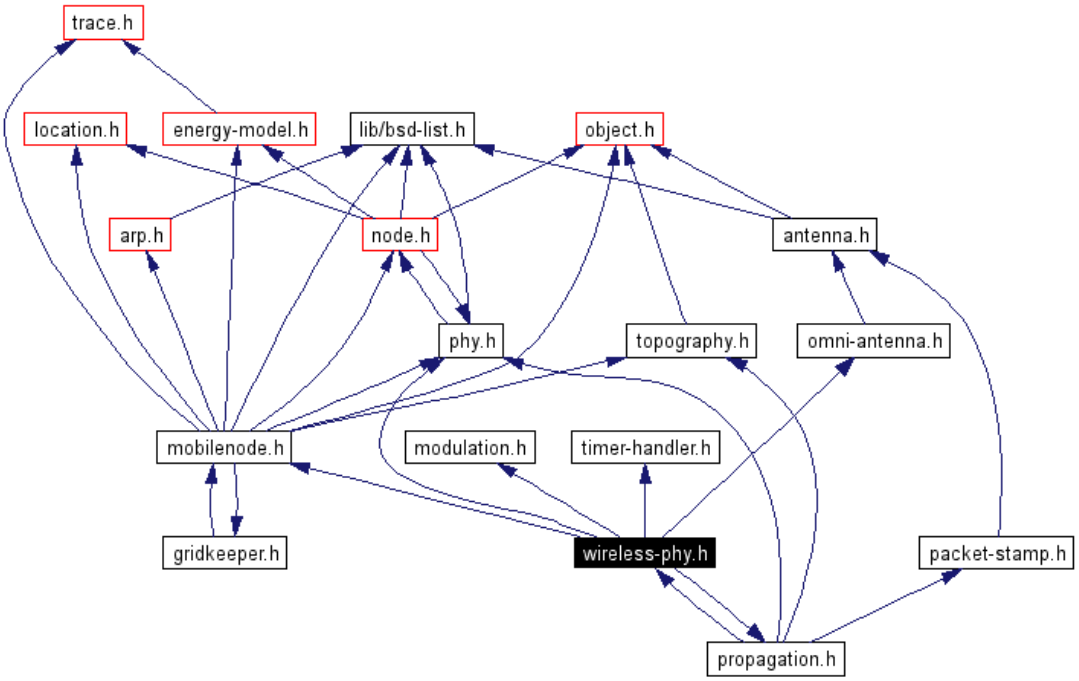


Fig.2 Ns2 dependency graph.

So as to create a simulation via ns2 we have to create a tcl script that describes the network and initiates the event scheduler. Once the user of ns2 creates the script, the interpreter takes the script creates the environment with the parameter and the characteristics that the user gives and ends the simulation. The simulator produces some files usually a trace files and a nam files. The first one contains all the packets that are sent, the time of everything happens, energy of the nodes and other results that the user may want. The second file contains traffics of the nodes. As we said, network simulator 2 is a group of C++ classes that are hierarchically connected and can be change for the purposes of the simulation.

2.2 Wireless sensor networks Communication protocols

Two main communication protocols standards that define a physical and MAC layer for wireless communications are 802.11 and 802.15.4 which are described in a few words in below paragraphs .

2.2.1 IEEE 802.11

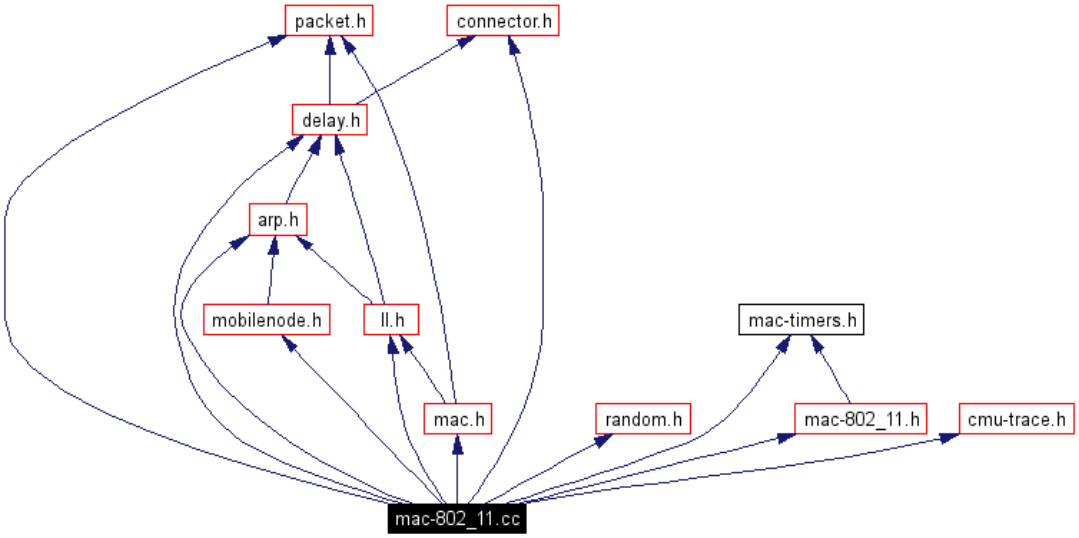


Fig.3 802.11 protocol class in ns2 structure.

802.11 protocol is supported from ns2.A wireless network communication protocol that is widely being used is the IEEE 802.11. When a number of nodes that are communicating by using 802.11, they are forming IBSSs (independent Basic Service Station). The minimum number of nodes in a IBSS is two. In an IBSS the nodes can communicate directly. The MAC layer of the 802.11. The fundamental access method for the 802.11 is the CSMA (carrier sense multiple access). The architecture of the MAC layer, includes the distributed coordination function (DCF), the point coordination function (PCF), and their coexistence [4]. 802.11 protocol provides asynchronous data service and many security features such as confidentiality, authentication and access control in conjunction with layer management.

A single first-in/first-out (FIFO) transmission queue is needed so as the 802.11 DCF work normally. The DCF CSMA/CA works as described below .A packet that arrives to the transmission queue is waiting the MAC so as to sense the medium and see if the channel is idle .If the medium is idle during a specific time interval (DIFS) then starts a backoff process which is a randomly selected counter. This counter is being decreased each time the medium is idle .When the counter reaches the zero number then the packet is transmitted .So as this process to be efficient each node have to be synchronized with the other by maintain a contention window . This is the mail function of the DCF CSMA/CA [3].

2.2.2 IEEE 802.15.4

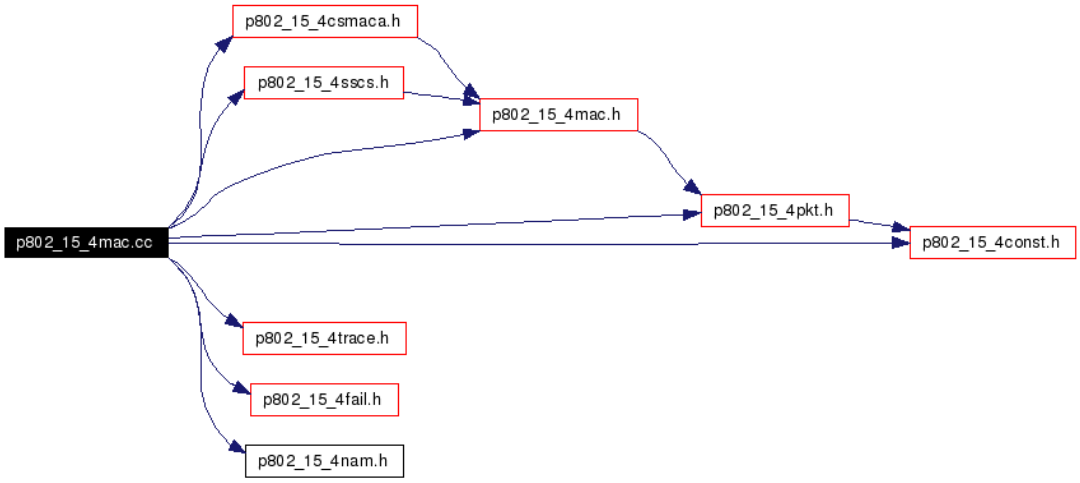


Fig.4 802.15.4 protocol structure into ns-2 .

The most important protocol is the IEEE 802.15.4 which comprises some interesting features such as power-efficiency, timeliness guarantees, scalability and it is fully handshake protocol so as to provide transfer reliability [9].Low rate personal area networks (PAN) are being defined by the new protocol. 802.15.4 defines the physical and the MAC access layers .Nodes in such a network communicate with each other at 2.4 GHz with a bit rate of 250 kbps. Other specification for the physical layer are at 915,868MHz and the data rates are decreased .Sensor and control applications are using ZigBee technology .This technology allows to sensor nodes that have different constructors to communicate without problem.

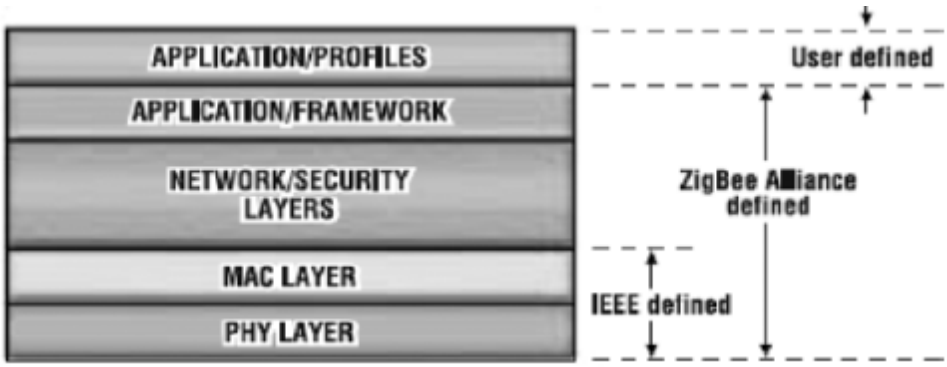


Fig.5 ZigBee layer specification [2].

2.3 Propagation Models

A simulation has two basic characteristics which are the topology of the sensor in an area and the signal that travels and transfer the data packets through the air from sensor to sensor. The signal is an electromagnetic wave that contains power and while travels the area losses power .Scientists have created models that shows the rate that signal losses its power . Some of those models are free space model, two ray ground model and okumura hataîs model. The first two models are already part of the ns2.

2.3.1 Free space model

The free propagation model is the ideal model for the signal, while it doesnît loose any power from the path. We assume that we have a flat area where there is not any obstacle between the source and the destination. It is clear that we have one line of sight path .The equation to find the receiving power of the destination node is described from the Friis eguation .

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L}$$

Where Pt is the transmitter power, Gt and Gr the gain of the transmitter and the receiver antenna respectively ,d is the distance between destination and source , L is constant that is 1 and ϟ is the wavelength [6] .

The free space model basically represents the communication range as a circle around the transmitter. If a receiver is within the circle, it receives all packets. Otherwise, it loses all packets .The OTcl interface for utilizing a propagation model (and using it in ns2 simulator) is the node-config command. One way to use it here is

$ns_ node-config -propType Propagation/FreeSpace

Another way is

```
set prop [new Propagation/FreeSpace]
$ns_ node-config -propInstance $prop
```

2.3.2 Two ray ground model

As we said above the line of sight path between two sensor nodes is ideal and happens only when the two sensors are very closely without any kind of obstacle between of them .Another model that considers not only the direct path but the reflection of the ground is the two ray ground model. This model is mole accurate when the distance between the two nodes is long without any obstacles between them .The equation that describes the model is below:

$$P_r(d) = \frac{P_t G_t G_r {h_t}^2 {h_r}^2}{d^4 L}$$

Where Pt is the transmitterûs power ,Gr and Gt the gain of the receiverûs and transmitterûs antennas respectively ,hr and ht is the height of the antennas ,L is constant equals to 1 and d the distance between the two nodes [5]. this equation is not good for small distances . As we can see the losses are bigger while the distance is bigger. Ns2 uses two ray model if we select this choice but for distances smaller than a limit (crossover distance) it uses free space model. The crossover distance is calculated from the ns2 through this equation:

$$d_c = (4\pi h_t h_r) / \lambda$$

Where λ is the wavelength, and hr, ht is the height of the receiver's and the transmitter's antenna respectively. Similarly, the OTcl interface for utilizing the two-ray ground reflection model is as follows [5].

```
$ns_ node-config -propType Propagation/TwoRayGround
```

Alternatively, the user can use :

```
set prop [new Propagation/TwoRayGround]
$ns_ node-config -propInstance $prop
```

2.3.3 Okumura Hata model

Okumura Hata model is a fully empirical method for calculating the path losses of the signal .It is based on measurements that scientists took in Tokyo city .It is the most widely used model in radio frequency propagation for predicting the behaviour of cellular transmissions in built up areas. Predictions are made through a number of graphs and the most important was from Hata. The model can be used for frequencies between 200 MHz and 2 GHz. This model incorporates the graphical information from Okumura model and develops it further to realize the effects of diffraction, reflection and scattering caused by city structures .The model has different equations for different types of areas .So ,the areas are divided into a series of clutter and terrain categories like open ,urban and suburban. An open area is a place that doesnôt have any trees or obstacles like buildings and is totally clear for more the 200 meters. A suburban area is an area that could be a small village with trees and other small obstacles. An urban area may be a place with a lot of big house like a small city with trees and other obstacles.

The Hataôs mathematical formulation is described below :

$$L_U = 69.55 + 26.16 \log f - 13.82 \log h_B - C_H + \left[44.9 - 6.55 \log h_B\right] \log d$$

For small cities with trees between the buildings CH is:

$$C_H = 0.8 + (1.1 \log f - 0.7)\, h_M - 1.56 \log f$$

For larger cities Ch is :

$$C_H = \begin{cases} 8.29\,(\log(1.54 h_M))^2 - 1.1 & \text{, if } 150 \le f \le 200 \\ 3.2\,(\log(11.75 h_M))^2 - 4.97 & \text{, if } 200 < f \le 1500 \end{cases}$$

Lu=Path loss in Urban Areas. Unit: decibel (dB)

Hb= Height of transmitting node Antenna. Unit: meter (m)

Hm= Height of receiving node Antenna. Unit: [meter(m)

F= Frequency of transmission. Unit: megahertz(MHz).

CH= Antenna height correction factor

[7]

d= Distance between the transmitting and receiving node sensors. Unit: kilometer (km).

2.3.4 Environmental Conditions and Path Loss

2.3.4.1 Vegetation Path Loss Modeling

As we said that when the signal moves towards the destination losses its power because of obstacles in its path. An obstacle can be a building or some trees. In our simulation we will support that our area is a small city in a place full of trees. Now we have to calculate not only the loss because of the buildings but the loss that occurs from the vegetation, too. Many studies have been carried out to characterize and model the effects of vegetation experimentally. They have been reviewed and summarized into several well-known through-vegetation loss models, such as Weissberger's. Weissberger's modified exponential decay model [8] . It is applicable in situations where propagation is likely to occur through a grove of trees rather than by diffraction over the top of the trees.

$$L_w\,(\mathrm{dB}) = \begin{cases} 1.33 \times f^{0.284} d^{0.588} & 14\,\mathrm{m} < d \leq 400\,\mathrm{m} \\ 0.45 \times f^{0.284} d & 0\,\mathrm{m} \leq d < 14\,\mathrm{m} \end{cases}$$

where *Lw* is the loss because of the vegetation in dB , *f* is the frequency in GHz, and

d is the depth of the trees in meter [8].

2.3.4.2 Rain and Wind Path Loss Model

Before we discussed about the power losses that the vegetation cause to the signal. Another significant loss while the signal travels an area can be caused from the weather conditions. The combined effects of wind and rain cause significant variation to the propagation channel. This variation in propagation channel can affect the reliability of modern communication systems when implemented for use within a forest environment [9]. So as to have the result scientist took measurements from an area of their desire and from the results

the made empirical formulation that calculate the path losses because of the weather phenomena such rain ,win or both .

ADDITIONAL ATTENUATION INDUCED BY VARYING WEATHER CONDITIONS ON THE PROPAGATING SIGNAL PER DISTANCE.

Frequency	Weather	No wind	Light Wind	Windy
240 MHz	No rain	0 dB/km	6.2 dB/km	8.5 dB/km
	Slight rain	1.5 dB/km	6.5 dB/km	8.1 dB/km
	Moderate rain	3.8 dB/km	12.7 dB/km	18.8 dB/km
700 MHz	No rain	0 dB/km	15.4 dB/km	22.6 dB/km
	Slight rain	11.1 dB/km	18.2 dB/km	40.2 dB/km
	Moderate rain	29.2 dB/km	30.6 dB/km	35.4 dB/km

Table1 .Attenuation induced by varying weather conditions.

Experimental results indicate that the wind and rain can impose an additional attenuation on the propagation signal within the forest environment. The additional attenuation increases as the strength of the wind and rain increases. This additional attenuation also increases as frequency increases. It is also observed that there is a large power variation and deep fades in the received signal as the strength of wind and the intensity of the rain increases [9].

2.4 AODV Protocol Description

Ad hoc On Demand Distance Vector (AODV)[12, 13] is source initiated, reactive protocol. It finds and keeps routes only if and when necessary. Route discovery works as follows. When the source requires a path to a particular destination, it sends to all nodes a route request (RREQ) packet in the ad hoc network. Nodes receiving RREQ record a *reverse* route back towards the source, using the node from which the RREQ was received as the next-hop, and then re-broadcasts the RREQ [14] .

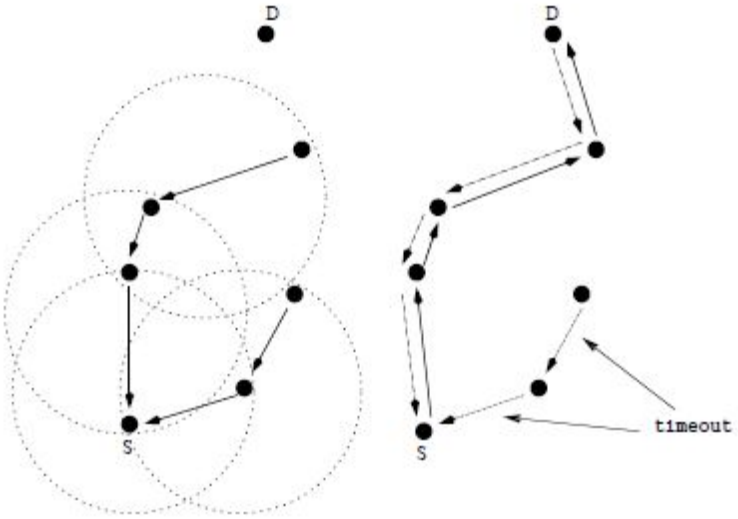


Fig.6 Reverse and forward route formation in AODV [14]

If the same RREQ is received more than once (via different routes), it is ignored. This way the RREQ packet is flooded to every node in the connected part of the network. When the RREQ packet reaches the destination, it sends a route reply (RREP) packet back to the source, using the reverse route. If an intermediate node has an up-to-date route to the destination, it may also send a RREP packet back to the source on behalf of the destination. As the RREP packet follows the path back to the source, the corresponding *forward* route is created at each intermediate node towards the destination (see Figure 2). Once the RREP packet reaches the source, data traffic can now flow along this forward route. To prevent routing loops, AODV maintains a sequence number on each node. Any routing information transmitted on routing packets or maintained on a node is tagged with the last known sequence number for the destination of the route [14]. AODV protocol guarantees the invariant that the destination sequence numbers in the routing table entries on the nodes along a valid route are always monotonically increasing. Other than preventing loops, sequence numbers also ensure freshness of routes. Given a choice of multiple routes, the one with a newer sequence number is always chosen. An important feature of AODV is maintenance of timer based states in each node, regarding utilization of individual routes. A route is ñexpiredò if not used recently. A set of predecessor nodes is maintained for each routing table entry, indicating a set of neighbouring nodes that use that entry to route

data packets. These nodes are notified with route error (RERR) packets when the next hop link breaks. Each predecessor node, in turn, forwards the RERR to its own set of predecessors, thus effectively erasing all routes using the broken link. This RERR is thus propagated to each source routing traffic through the failed link, causing the route discovery process to be reinitiated if routes are still needed [14].

3. Simulations and results

In below paragraphs we will show results of our simulations. We will describe and analyse them. The results that are gathered here are containing information that we need. All tracefiles and the rest files that are extracted from simulations can be found on the cd .

3.1 Adaptation of our source code to NS-2.

So as to create our conditions in ns-2 we have to build into network simulator the equations that describe the path loss of the electromagnetic wave .For this reason we have created several c++ classes that can cooperate with the already existent classes of the ns -2 and describe the losses of the signal from different environmental reasons. These files are Forest.cc, ForestModel.h, OkumuraForestModel.cc, OkumuraForestModel.h, RainWindModel.cc, RainWindModel.h . Forest .cc calculates the path losses of a wave that passes through a forest.

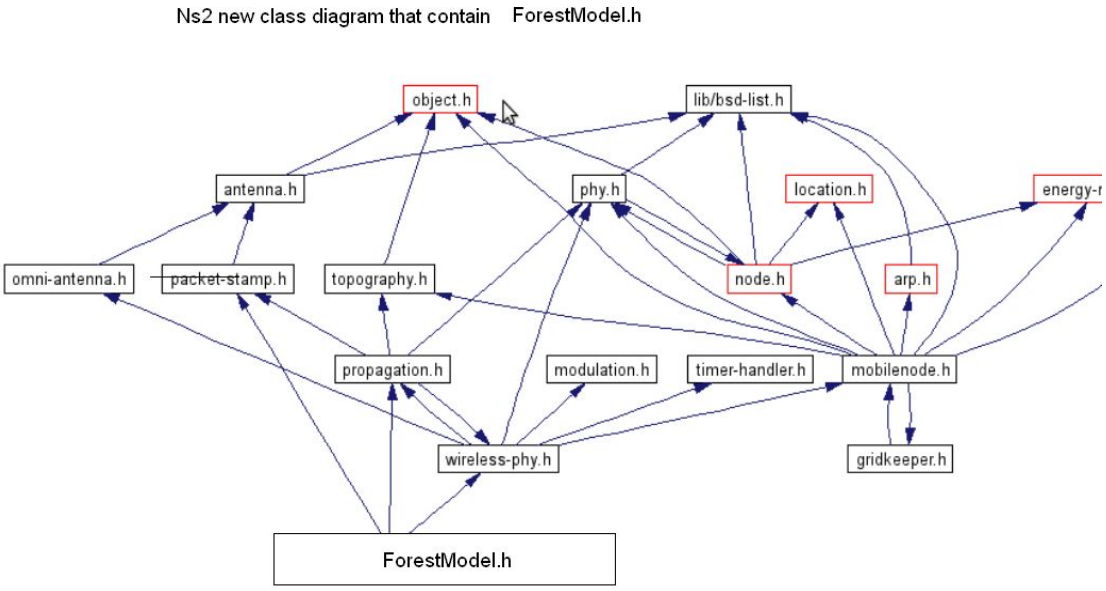


Fig.7 ForestModel.h library into ns-2 structure.

OkumuraForestModel .cc calculates the path losses of a wave that passes from an area with small buildings into a small town.

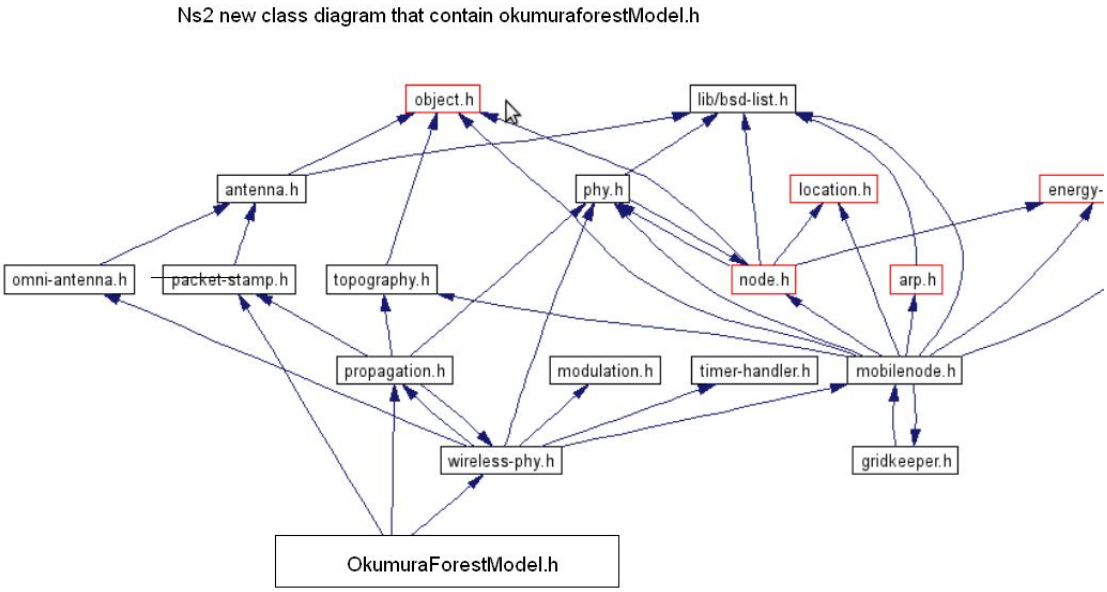


Fig.8 OkumuraForestModel.h library into ns-2 structure.

RainWindModel.cc calculates the path losses because of the rain.

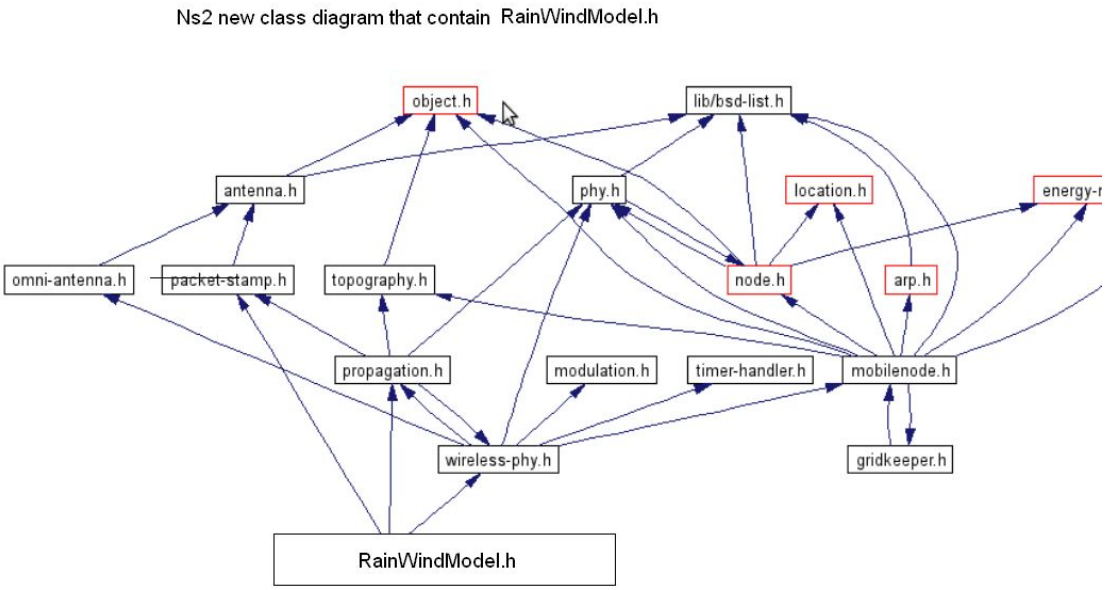


Fig.9 RainWindModel.h library into ns-2 structure.

These classes have similarities with the already existent propagation models of the ns-2 in their code. These similarities are existing so as to face problems and errors while debugging the new code into ns-2.So as to install that classes into ns-2 we have to write the code and inherit function from the other classes such as propagation.h, wireless-phy.h, packet-stamp.h. The new classes must be in the same directory with the two ray ground class. So as to embody the new class to ns2 we have to run some command at a terminal of Linux. Those commands are ñmake cleanò and ñmakeò. Several problems that may occur because of the lack of patches or version of ns2 and ubuntu can be faced from forums. Firstly, we will run several simulations that use the free space model .We will collect the trace files and by using the awk files we will take the desired results.

Table2 zigbee network/1 Mbps /No Move nodes/512 bytes				
protocol	Sent	Dropped	Received	Throughput
message	491	484	491	0,003822
AODV	6761	3953	51577	
total	7752	4437	52068	

Table3 zigbee network/5 Mbps /No Move nodes/512 bytes				
protocol	Sent	Dropped	Received	Throughput

messages	491	484	491	0,004022
AODV	6761	3953	51577	
total	7752	4437	52068	

Table4 zigbee network/5 Mbps /No Move nodes /64byte packet				
protocol	Sent	Dropped	Received	Throughput
messages	491	484	491	0,000503
AODV	6761	3953	51577	
total	7752	4437	52068	

Table 5 zigbee network/5 Mbps /With Move nodes /64 bytes				
protocol	Sent	Dropped	Received	Throughput
messages	491	484	491	0,000502
AODV	6761	3953	51577	
total	7752	4437	52068	

Table 6 zigbee network/5 Mbps /With Move nodes /512				
protocol	Sent	Dropped	Received	Throughput
messages	491	484	491	0,004031
AODV	6761	3953	51577	
total	7752	4437	52068	

Table 7 zigbee network/1 Mbps /With Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	491	484	491	0,004022
AODV	6761	3953	51577	
total	7752	4437	52068	

Table 8 zigbee network/1 Mbps /No Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	491	491	491	0,004022
DSR	204	2172	42173	
total	695	2663	42664	

Table 9 zigbee network/5 Mbps /No Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	490	484	490	0,004020
DSR	6761	3953	51577	
total	7751	4437	52067	

Table 10 zigbee network/1 Mbps /With Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	485	489	485	0,004019
DSR	204	1138	15846	
total	689	1627	16331	

Table 11 zigbee network/5 Mbps /With Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	491	491	491	0,004022
DSR	204	1138	15846	
total	695	1629	16337	

All tables contain information about packet losses .The type of the packets are messages and network packets , that are using to control the network (DSR packets).As we can observe above throughput is around 0,004020 in all cases except only when we have small packet size (64 Kbytes) ,where throughput is 0,000503 .

After the addition of our classes into the ns-2, we calculate again the losses and the throughput by processing the new trace files.

Table 12 zigbee network/1 Mbps /No Move nodes/512 bytes/Losses				
protocol	Sent	Dropped	Received	Throughput
message	415	536	415	0,003119
AODV	7295	3999	55650	
total	7710	4535	55065	

Table 13 zigbee network/5 Mbps /No Move nodes/512 bytes				
protocol	Sent	Dropped	Received	Throughput
messages	418	501	418	0,003123
AODV	7296	3953	55650	
total	7714	4454	55068	

Table 14 zigbee network/5 Mbps /No Move nodes /64 byte packet				
protocol	Sent	Dropped	Received	Throughput
messages	417	582	417	0,000415
AODV	7290	3954	51036	
total	7707	4536	51453	

Table 15 zigbee network/5 Mbps /With Move nodes /64 byte packet				
protocol	Sent	Dropped	Received	Throughput
messages	416	631	416	0,000377
AODV	6989	4122	53116	
total	7405	4753	53532	

Table 16 zigbee network/5 Mbps /With Move nodes /512 byte packet				
protocol	Sent	Dropped	Received	Throughput

messages	417	499	417	0,003122
AODV	7300	4010	55480	
total	7717	4519	55897	

Table 17 zigbee network/1 Mbps /With Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	417	483	417	0,003121
AODV	7411	4212	56740	
total	7828	4695	57157	

Table 18 zigbee network/1 Mbps /No Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	436	481	436	0,003415
DSR	205	1097	15908	
total	6641	1578	16344	

Table 19 zigbee network/5 Mbps /No Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	437	481	437	0,003416
DSR	205	1097	15908	
total	6642	1578	16345	

Table 20 zigbee network/1 Mbps /With Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	432	490	432	0,003410
DSR	335	1255	25996	
total	767	1745	26428	

Table 21 zigbee network/1 Mbps /With Move nodes / 512				
protocol	Sent	Dropped	Received	Throughput
messages	431	491	431	0,003411
DSR	335	1256	25996	
total	768	1746	26427	

Above we can see the results from the various simulations that we have done under ns-2, from which we can understand that the throughput is decreased because the number packets that are transmitted are smaller. In the start we run simulations without to add our files .the conditions are ideal for our nodes. In every simulation we use around 100 nodes .The ideal conditions for the nodes are to use free space propagation model .Wave doesnôt have important losses and the distance between the nodes can be very long (up to 20 meters) .We can observe this from the nam file. One important metric of a network is throughput which is the average rate of successful message delivery over a communication channel. We are calculating network throughput for every simulation in our project. As we can see, throughput doesnôt change significantly if the conditions are the same and we change only the bandwidth. When we add our environmental conditions in the simulation we can see several changes in the results. Throughput is decreasing by 10-20 % . Throughput much bigger if nodes have permanent positions and they donôt change while the conditions of the simulations change .After the addition of our classes, we observe that, the maximum distance between two nodes is less 4-5 meters so as the network to work normal. We can observe this by reading the nam file that is producing from the ns-2 .Of course it isnôt logical to put data from this file to this paper as it is difficult to read without a tool and extremely big. if positions are the same as before the addition and not automatically change by ns 2 so as nodes to be closer, neither one of the packets would reach the destination . Also we can observe that the motion of the nodes affects the network because nodes have to send packets (AODV, DSR) more frequent so as to send their position to other nodes.

So as to increase the effectiveness of the network we have to increase throughput and of course the distance of the nodes. One way to increase the distance of the nodes is by increasing the radius of each node. This is simple but it consumes more energy than before .We can succeed it by changing the energy for transmission for every node through ns-default.tcl file that exist in ns-2 directory and keeps the default values for the most of the parameters in ns-2 tool .Below we can see the part that describes the transmission power where we can increase Pt parameter.

```
00632 Phy/WirelessPhy set RXThresh_ 3.652e-10
00633 Phy/WirelessPhy set bandwidth_ 2e6
00634 Phy/WirelessPhy set Pt_ 0.28183815
00635 Phy/WirelessPhy set freq_ 914e+6
```

A way to increase throughput is to decrease velocity of the nodes. A way to increase throughput in my simulation was to make the packets that a node send, bigger .Also because of the of the equation that we have ,that is affected from the height ,we can add another parameter that can increase the receiving power .This parameter is the height of the antenna. The part that controls the height of the node antenna is the Z_ .This parameter exists in ns-default.tcl from where we can modify it.

```
00623 Antenna/OmniAntenna set Y_ 0
00624 Antenna/OmniAntenna set Z_ 1.5
00625 Antenna/OmniAntenna set Gt_ 1.0
```

The results from a simulation into our environmental conditions are showing below.

Table 22 zigbee network/1 Mbps /No Move nodes / 768				
protocol	Sent	Dropped	Received	Throughput
messages	441	490	431	0,004105
DSR	354	1254	27470	

total	795	1744	27901	

In that simulation we have increased the height of the antenna to 2,5 meters .There is no motion for the nodes .The transmitted power is increased by 10 % so as to increased the radius of the node .The simulation runs under our environmental conditions .As we can see the received packets are around 10% more than without the modifications .Also ,throughput is improved.

3.2 Energy growing scenario challenge

The growing scenario of a wireless sensor network takes place when one or more nodes try to join the network. Our network ,after communicating with all the nodes that want to join ,embodies them .A challenge that occurs when the nodes are moving and take their position is that the energy that consume is becoming greater. Also when new nodes come to the network a bigger region is covered .As a result new routing paths are created and nodes have to be informed of this changes so as to select the correct path. In our simulation nodes are using AODV algorithm so as to find the path. In order to decrease energy that a node is consuming we will make some changes to the protocol. We will add some code so as the algorithm not only to find the path by calculation the minimum number of hops but to find the path with the smaller energy cost for the node. The files that need some changes are aodv.cc, aodv.h, aodv_packet.cc, aodv_packet.h [15]. The difficult part of this modification is not the changes but to make the code to run into ns2. After a lot of work I change ns-2 2.31 to ns-2 2.29 because of some new bugs that I had to face because of the version. The most of the changes are in aodv.cc file where we put a new parameter of the energy cost of the path .Some of the changes are described below .The first change changes the way that the routing table is being updated by putting the energy cost as an additional parameter.

```
void
AODV::rt_update(aodv_rt_entry *rt, u_int32_t seqnum, u_int16_t metric,
                nsaddr_t nexthop, double expire_time,double energyCo) {
```

```
    rt->rt_seqno = seqnum;
    rt->rt_hops = metric;
    rt->rt_flags = RTF_UP;
    rt->rt_nexthop = nexthop;
    rt->rt_expire = expire_time;
    rt->rt_energyCo = energyCo;
}
```

So as to find the energy of the node we have to acquire that price while the simulation runs .

```
Node* currentNod = Node::get_node_by_address(index);
double currentEnergy;
if (currentNod->energy_model())
        {
                currentEnergy = currentNod->energy_model()->energy();
        }
double thisCost;
int   this_max_que = 0;
this_max_que = rqueue.len_;
```

The nodes must be inform for the energy state of their neighbour and this will happen from any communication that have with other nodes .So, the function which is suitable for this job is sendreply() function where we put a update there . The rq->rq_cost) is what we have add to that function .

```
sendReply(rq->rq_src,          // IP Destination
        1,                  // Hop Count
        index,              // Dest IP Address
        seqno,              // Dest Sequence Num
        MY_ROUTE_TIMEOUT,     // Lifetime
        rq->rq_timestamp,
```

```
            rq->rq_cost);[15]
```

Another sendreply() function ,that Inserts nexthops to RREQ source and RREQ destination in the precursor lists of destination and source respectively ,is described below with her update .

```
  sendReply(rq->rq_dst,
            rq->rq_hop_count,
            rq->rq_src,
            rq->rq_src_seqno,
              (u_int32_t) (rt->rt_expire - CURRENT_TIME),
                    rq->rq_timestamp,
            rq->rq_energyCo); //energy is considered here
```

One of the most important changes is the way that the routing table is being updated. Before the changes the node was trying to find if there was a fresher sequence number or smaller number of hops so as to do the update .Now a new criteria is the energy cost of that hop.

```
if ( (rt->rt_seqno < rp->rp_dst_seqno ) ||
          ((rt->rt_seqno == rp->rp_dst_seqno) &&
           (rt->rt_energyCo > rp->rp_energyCo)) ) {//the criteria of energy
   rt_update(rt, rp->rp_dst_seqno, rp->rp_hop_count, rp->rp_src,
                CURRENT_TIME + rp->rp_lifetime ,rp->rp_energyCo);
```

Also so as to make the protocol to work normal we have to make some changes to the request and reply packets so as to include the energy information.

```
struct hdr_aodv_request {
        u_int8_t        rq_type; // Packet Type
```

```
u_int8_t        reserved[2];
u_int8_t        rq_hop_count;   // Hop Count
u_int32_t       rq_bcast_id;
nsaddr_t        rq_dst;         // Destination IP Address
u_int32_t       rq_dst_seqno;   // Destination Sequence Num
nsaddr_t        rq_src;         // Source IP
u_int32_t       rq_src_seqno;   // Source Sequence Num
double          rq_timestamp;   // when REQUEST sent;

  double          rq_less_energy; //energy information
int             rq_max_que;
  Double          rq_energyCo; é  é .} [15]
```

The new reply packet structure is being shown below.

```
struct hdr_aodv_reply {
u_int8_t        rp_type;        // Packet Type
u_int8_t        reserved[2];
u_int8_t        rp_hop_count;           // Hop Count
nsaddr_t        rp_dst;                 // Destination IP        u_int32_t
rp_dst_seqno;           // Destination Sequence Num
nsaddr_t        rp_src;                 // Source IP
double          rp_lifetime;            // Lifetime
double          rp_timestamp;
double          rp_min_energy; //energy information
int             rp_max_que;
  double rp_cost; é .} [15]
```

After the modifications at the code we have to run make clean and make into the ns2 main directory .Below we will show results from simulations .These

results show us the average energy consumption of the nodes. The results can be acquired by applying the findEnergy.awk file to the trace files that produced from the ns2 after running simulations. Table 1 shows the average energy f each node in the network before the addition of the new nodes.

Table 23. Energy consumption without the addition of extra nodes		
Initial Energy	Energy used/node consumption during sim.	Energy used (percent) %
97.976286	58.737972	58.740122

So as to have node addition to the network we create through the tcl file 20 nodes that are moving to the network from far away .When the nodes come closer to the initial nodes ,they communicate and join the network . Because of the process of joining a lot of new AODV packets are created and new path that consume more or less energy are created, too. Table 2 shows the average energy of each node of the simulation that includes the additions of nodes.

Table 24. . Energy consumption with the addition of extra nodes.		
Initial Energy	Energy used/node consumption during sim.	Energy used (percent) %
97.976286	75.827767	75.830542

In order to make this addition to consume less energy we apply several small changes to AODV protocol so as make nodes to consume less energy. By this technique we increase the operational life time of our network . We run again the simulation with the addition of the new nodes and we have the results of table 3.

Table 25. . Energy consumption with the addition of extra nodes after the addition of our code.		
Initial Energy	Energy used/node consumption during sim.	Energy used (percent) %
97.976286	65.362789	65.463477

As we can observe the average energy that is consumed is around 10 % less than before. Now the node is selecting the path by concerning not only the number of hops but the cost of every hop. The AODV protocol and all its files can be found at the ns2 directory and has been created by university of Berkeley.

4. Conclusions

Classes that create environmental conditions so as to run a simulation into a network simulator are very useful. We can observe the parameters that affect the energy of the nodes of a network. We can control the cost of the network by knowing the distance that a node must have from the other nodes so as to cover a bigger region with the smaller number of nodes.

We have encountered a lot of problems so as to create and more problems to put that code into ns-2. Ns -2 is hard to be modified, especially for a beginner .Nevertheless the results was of our techniques was are favourable and encouraging. The longetivity to a network that we add new nodes can be extended by applying the changes in the initial protocol.

The technique that we have use is very generic, it is being used in other protocols too.

5 Future Work

There are many directions in the future for research on wireless sensor networks.

1. First of all, I have to make the added code inside ns2 to work more stable without to create so many errors. In the most simulations that I have tried the code created errors.

2. Check if there is any other way so as to make the consumption of the nodes smaller.
3. Examine how the environmental conditions can affect the creation of the routing path in a wireless sensor network.

6 References

[1]. . Ian Downard .Simulating sensor networks in ns2 Naval Research Laboratory , downard@itd.nrl.navy.mil .

[2] Nitaigour P. Mahalik , Sensor Networks and Configuration Fundamentals, Standards, Platforms,and Applications . Institute of Science and Technology Department of Mechatronic Republic of South Korea .

[3] Ananda Mun Choon , Chan Wei Tsang . Mobile wireless, and sensor networks technology, applications, and future directions. University of Singapore Ieee Press , A hohn Wiley & Sons, inc., publication.

[4] IEEE Computer Society ANSI/IEEE ,wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Sponsor LAN MAN Standards Committee of the Std 802.11, 1999 Edition Information technology Telecommunications and information exchange between systems Local and metropolitan area networks

[5] Two ray ground model, Berceley University
http://www.isi.edu/nsnam/ns/doc/node218.html.

[6] Free space model, Berkeley University,
http://www.isi.edu/nsnam/ns/doc/node217.html.

[7] Hata ,Empirical Formula for Propagation Loss in Land Mobile Radio Services., M IEEE Trans. Vehicular Technology, VT-29, pp. 317 - 325, 1980 .

[8] Y. S. Meng, Y. H. Lee, and B. C. Ng 2009 . Study of propagation loss prediction in forest environment . *Progress In Electromagnetics Research B, Vol. 17, 117{133}*

Technological University *Progress In Electromagnetics Research B, Vol. 17, 117{133, 2009*

[9] Yu Song Meng ,Yee Hui Lee .The Effects of Tropical Weather on Radio Wave Propagation over Foliage Channel IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, VOL. X, NO. X, XXX 2009 1

[10] Elmurod Talipov .Topology generator for Network Simulator.

[11]http://en.pudn.com/downloads99/sourcecode/others/detail405086_en.html

[12] Charles Perkins and Elizabeth Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications*, pages 90ï100, Feb 1999.

[13] Charles Perkins, Elizabeth Royer, and Samir R. Das. Ad hoc on demand distance vector (AODV) routing. http://www.ietf.org/internet-drafts/ draft-ietf-manet-aodv-07.txt, Nov 2000. IETF Internet Draft (work in progress) .

[14] . Nishant Gupta and Samir R. Das Energy-Aware On-Demand Routing for Mobile Ad Hoc Networks . OPNET Technologies.

[15] Optimization of AODV protocol
http://en.pudn.com/downloads30/sourcecode/unix_linux/network/detail96026_en.html

Appendix

SECTION 1

In section 1 there is included all ns-2 code that we have created. Code below contains not only classes and library but also files that is written in awk programming language and help us to take results. AWK files can process trace files that are created from the ns2 after a simulation. Of course awk files in this paper have a general form and need some modifications so as to

process a trace file created with AODV or DSR .For example we need to change some ñifò conditions so as to check if a packet is cbr or tcp or AODV or DSR. It depends of the protocol that we use .The original propagation models of two ray ground model and the free space model can be found at ns-2 installation (if you have installed it) . The last firstzb.tcl file is just a simple tcl code that creates a topology of nodes. We use several times this file to create the network by changing only some parameters such as the protocol (AODV, DSR), the position of the nodes, the type of the packets or the propagation model (free space model, two ray , our model).

1. ForestModel.cc

```
#include <math.h>

#include <delay.h>

#include <packet.h>

#include <phy.h>

#include <packet-stamp.h>

#include <antenna.h>

#include <mobilenode.h>

#include <propagation.h>

#include <wireless-phy.h>

#include <ForestModel.h>


static class ForestModelClass: public TclClass {

public:

	ForestModelClass() : TclClass("Propagation/ForestModel") {}

	TclObject* create(int, const char*const*) {

		return (new ForestModel);

	}

} class_ForestModel;
```

```
ForestModel::ForestModel()
{
  last_hr = last_ht = 0.0;
  crossover_dist = 0.0;
}



double ForestModel::through2Forest(double PowerTrans, double GainTrans,
double GainRec, double ht, double hr, double L, double d)
{
        /*
         *  It shows us how to find the receiving power of the     receiver
         *  when signal passes from forest .Our terrain have a lot of trees
         *   and buildings too.
         *
         *
         *
         *
         */
double noproblem;
noproblem=PowerTrans+GainTrans+GainRec+ht+hr+L+d; //Just i used them
so as not to have indication for unused variables
double w,x;
double frequency,ForestLossesdB ;
double PowerRec;
double result1;
```

```
frequency=868; //Mhz of zigbee in europe.




result1=pow(frequency,0.284);


 ForestLossesdB=0.45*result1*distance ;

//Path losses are in db so we have to convert them.


x=ForestLossesdB/10;


w=pow(10,x);

 //w variable shows us the pathlosses in Watt


PowerRec=PowerTrans-w;

// the receiving in watts is colculating here.

//return PowerTrans * GainTrans * GainRec * (hr * hr * ht * ht) / (d * d * d * d *
L)

  return PowerRec;

}
```

```
double
ForestModel::Pr(PacketStamp *t, PacketStamp *r, WirelessPhy *ifp) //is
inherited from Propagation.h so we do not change it
{
  double ton;
  double hr, ht;                // height of recv and xmit antennas
  double rX, rY, rZ;            // location of receiver
  double tX, tY, tZ;            // location of transmitter
  double d;                             // distance
  double L = ifp->getL();                       // system loss
  double PowerRec;                      // received signal power
  ton=2;

  double lambda = ifp->getLambda();      // wavelength

  r->getNode()->getLoc(&rX, &rY, &rZ);
  t->getNode()->getLoc(&tX, &tY, &tZ);

  rX += r->getAntenna()->getX();
  rY += r->getAntenna()->getY();
  tX += t->getAntenna()->getX();
  tY += t->getAntenna()->getY();

  d = sqrt((rX - tX) * (rX - tX)
           + (rY - tY) * (rY - tY)
           + (rZ - tZ) * (rZ - tZ));
```

```
/* We're going to assume the ground is essentially flat.
   This empirical two ground ray reflection model doesn't make
   any sense if the ground is not a plane. */


hr = rZ + r->getAntenna()->getZ();
ht = tZ + t->getAntenna()->getZ();

 // if (hr != last_hr || ht != last_ht)
 // {
 // crossover_dist = (4 * PI * ht * hr) / lambda;
 //last_hr = hr; last_ht = ht;
 //}


double GainTrans = t->getAntenna()->getTxGain(rX - tX, rY - tY, rZ - tZ,
                                              t->getLambda());
double GainRec = r->getAntenna()->getRxGain(tX - rX, tY - rY, tZ - rZ,
                                              r->getLambda());

   PowerRec = through2Forest(t->getTxPowerRec(), GainTrans, GainRec, ht,
hr, L, d);

   printf("Through2Forest  %e\n",PowerRec);

   return PowerRec;
```

```
}

double ForestModel::getDist(double PowerRec, double PowerTrans, double
GainTrans, double GainRec, double hr, double ht, double L, double lambda)
{
	/* A way to find distance */
double x=0.0;
double distance=0.0;
x=sqrt(PowerTrans * GainTrans* GainRec * (hr * hr * ht * ht) / PowerRec);
distance=sqrt(x);
return distance;
}
```

2.ForestModel.h

```
#ifndef __OkumuraForestModel_h__
#define __OkumuraForestModel_h__

#include <packet-stamp.h>
#include <wireless-phy.h>
#include <propagation.h>
#include <antenna.h>
#include <phy.h>

class ForestModel: public Propagation {
```

```
public:
  ForestModel();
  virtual double Pr(PacketStamp *tx, PacketStamp *rx, WirelessPhy *ifp);
  virtual double getDist(double Pr, double Pt, double Gt, double Gr,
                         double hr, double ht, double L, double lambda);

protected:
  double trought2Forest(double Pt, double Gt, double Gr, double ht, double hr,
double L, double d);//our function to find Pr
  double last_hr, last_ht;
  double crossover_dist;
};


#endif
```

3. OkumuraForestModel.cc

```
#include <math.h>
#include <delay.h>
#include <packet.h>
#include <phy.h>
#include <packet-stamp.h>
#include <antenna.h>
#include <mobilenode.h>
#include <propagation.h>
#include <wireless-phy.h>
#include <OkumuraForestModel.h>
```

```
static class OkumuraForestModelClass: public TclClass {

public:

	OkumuraForestModelClass() :
TclClass("Propagation/OkumuraForestModel") {}

	TclObject* create(int, const char*const*) {

		return (new OkumuraForestModel);

	}

} class_OkumuraForestModel;


OkumuraForestModel::OkumuraForestModel()

{

  last_hr = last_ht = 0.0;

  crossover_dist = 0.0;

}


// use Friis at less than 5 m distance

// use our OkumuraForest model at more than 5 m distance


double OkumuraForestModel::throughForest(double Pt, double Gt, double Gr,
double ht, double hr, double L, double d)

{

	/*

	 *  Okumura Hata Propagation for Forest enviroment model.

	 *

	 *

	 *

	 *
```

```
        *
        *
        */
double noproblem;
noproblem=Pt+Gt+Gr+ht+hr+L+d; //Just i used them so as not to have
indication for unused variables
double w,x;
double frequency,Ch,PathlossesdB ;
double Pr;

frequency=868; //Mhz of zigbee in europe.
//Colculation of the pathlosses by using Okumura's Hata model.
//Ch=0.8+(1.1 * log(frequency)-0.7) * hr - 1.56 * (log(frequency)) ;

Ch=2*((log ((frequency)/28)(log ((frequency)/28))+5.4) ;

 PathlossesdB=69.55+26.16*log(frequency)-13.82*log(ht)-Ch+(44.9-
6.55*log(ht))*log(d);
//Path losses are in db so we have to convert them.

x=PathlossesdB/10;

w=pow(10,x);
 //w variable shows us the pathlosses in Watt

Pr=Pt-w;
// the receiving in watts is colculating here.
  return Pr;
```

```
}

double
OkumuraForestModel::Pr(PacketStamp *t, PacketStamp *r, WirelessPhy *ifp)
//is inherited from Propagation.h so we do not change it
{
  double ton;
  double hr, ht;                // height of recv and xmit antennas
  double rX, rY, rZ;            // location of receiver
  double tX, tY, tZ;            // location of transmitter
  double d;                             // distance
  double L = ifp->getL();                       // system loss
  double Pr;                    // received signal power
  ton=2;

  double lambda = ifp->getLambda();      // wavelength

  r->getNode()->getLoc(&rX, &rY, &rZ);
  t->getNode()->getLoc(&tX, &tY, &tZ);
```

```
  rX += r->getAntenna()->getX();
  rY += r->getAntenna()->getY();
  tX += t->getAntenna()->getX();
  tY += t->getAntenna()->getY();

  d = sqrt((rX - tX) * (rX - tX)
           + (rY - tY) * (rY - tY)
           + (rZ - tZ) * (rZ - tZ));

  hr = rZ + r->getAntenna()->getZ();
  ht = tZ + t->getAntenna()->getZ();

  if (hr != last_hr || ht != last_ht)
    {
// Here we colculate  cross-over distance
     /* in case that maybe we will use it.
                 4 * PI * hr * ht
          d = ----------------------------
                     lambda
           *
           *
      */

     crossover_dist = (4 * PI * ht * hr) / lambda;
     last_hr = hr; last_ht = ht;
#if DEBUG > 3
     printf("TRG: %e.10 hr %f ht %f\n",
```

```
                crossover_dist, hr, ht);
#endif
    }

  /*
   *  If the transmitter is smaller then 5 meters range ,then we use the
   *  Free Space Loss Model.  Otherwise,system  uses the our
OkumuraForestModel
   *   model.
   */

  double Gt = t->getAntenna()->getTxGain(rX - tX, rY - tY, rZ - tZ,
                                          t->getLambda());
  double Gr = r->getAntenna()->getRxGain(tX - rX, tY - rY, tZ - rZ,
                                          r->getLambda());

#if DEBUG > 3
  printf("TRG %.9f %d(%d,%d)@%d(%d,%d) d=%f xo=%f :",
          Scheduler::instance().clock(),
          t->getNode()->index(), (int)tX, (int)tY,
          r->getNode()->index(), (int)rX, (int)rY,
          d, crossover_dist);

#endif

  if(d < 5) {//in a forest 5 metres is enough so as to have absorption
    Pr = Friis(t->getTxPr(), Gt, Gr, lambda, L, d);
```

```
#if DEBUG > 3
    printf("Free Space Model %e\n",Pr);
#endif
    return Pr;
  }
  else {
    Pr = throughForest(t->getTxPr(), Gt, Gr, ht, hr, L, d);
#if DEBUG > 3
    printf("ThroughForest  %e\n",Pr);
#endif
    return Pr;
  }
}


double OkumuraForestModel::getDist(double PowerRec, double PowerTrans,
double GainTrans, double GainRec, double hr, double ht, double L, double
lambda)
{
      /* A way to find distance */
double x=0.0;
double distance=0.0;
x=sqrt(PowerTrans * GainTrans* GainRec * (hr * hr * ht * ht) / PowerRec);
distance=sqrt(x);
return distance;
}
```

4. OkumuraForestModel.h

```
#ifndef __OkumuraForestModel_h__
#define __OkumuraForestModel_h__

#include <packet-stamp.h>
#include <wireless-phy.h>
#include <propagation.h>
#include <antenna.h>
#include <phy.h>

class OkumuraForestModel: public Propagation {
public:
  OkumuraForestModel();
  virtual double Pr(PacketStamp *tx, PacketStamp *rx, WirelessPhy *ifp);
  virtual double getDist(double Pr, double Pt, double Gt, double Gr,
                         double hr, double ht, double L, double lambda);

protected:
  double troughtForest(double Pt, double Gt, double Gr, double ht, double hr,
double L, double d);//our function to find Pr
  double last_hr, last_ht;
  double crossover_dist;
};

#endif
```

5. RainWindModel.cc

```
#include <math.h>
#include <delay.h>
#include <packet.h>
#include <phy.h>
#include <packet-stamp.h>
#include <antenna.h>
#include <mobilenode.h>
#include <propagation.h>
#include <wireless-phy.h>
#include <RainWindModel.h>

static class RainWindModelClass: public TclClass {
public:
	RainWindModelClass() : TclClass("Propagation/RainWindModel") {}
	TclObject* create(int, const char*const*) {
		return (new RainWindModel);
	}
} class_RainWindModelModel;

RainWindModel::RainWindModel()
{
  last_hr = last_ht = 0.0;
  crossover_dist = 0.0;
}

double findLossesRW(double d){
```

```
double w,x;

double frequency,RainWindLossesdB ;

double PowerRec;

double result1,loss;

frequency=868; //Mhz of zigbee in europe.



//As we know attenuation at 700 Mhz is 40dB so we suppose that
attennuation

//for 868 Mhz that is zigbee will be from 39 to 45 db .So we take a random
number

//to be more accurate .The attenuation is not stable between all the nodes .

//and changes little from time to time

result1=rand(39,45);

//Result1 variable describes the losses of the signal for a distance of 1000
meters

//We have to found the losses for the distance that we have between the 2
nodes

loss=(d*result1)/1000;



 RainWindLossesdB=loss ;

//Path losses from rain are in db so we have to convert them.

x=RainWindLossesdB/10;


w=pow(10,x);

 //w variable shows us the pathlosses in Watt
```

```
return w;
}

double RainWindModel::through2RainWindModel(double PowerTrans, double
GainTrans, double GainRec, double ht, double hr, double L, double d)
{
	/*
	 *  It shows us how to find the receiving power of the receiver
	 *  when signal passes from forest .Our terrain have a lot of trees
	 *   and buildings too.
	 *
	 *
	 *
	 *
	 */
double noproblem;
noproblem=PowerTrans+GainTrans+GainRec+ht+hr+L+d; //Just i used them
so as not to have indication for unused variables
double w,x;
double frequency,RainWindLossesdB ;
double PowerRec;
double result1,loss;
frequency=868; //Mhz of zigbee in europe.


//As we know attenuation at 700 Mhz is 40dB so we suppose that
attennuation
```

```
//for 868 Mhz that is zigbee will be from 39 to 45 db .So we take a random number

//to be more accurate .The attenuation is not stable between all the nodes .

//and changes little from time to time

result1=rand(39,45);

//Result1 variable describes the losses of the signal for a distance of 1000 meters

//We have to found the losses for the distance that we have between the 2 nodes

loss=(d*result1)/1000;



 RainWindLossesdB=loss ;

//Path losses from rain are in db so we have to convert them.

x=RainWindLossesdB/10;


w=pow(10,x);

 //w variable shows us the pathlosses in Watt


PowerRec=PowerTrans-w;

// the receiving in watts is colculating here.

//return PowerTrans * GainTrans * GainRec * (hr * hr * ht * ht) / (d * d * d * d * L)

  return PowerRec;

}
```

6. RainWindModel.h

```
#ifndef __RainWindModel_h__
```

```
#define __RainWindModel_h__

#include <packet-stamp.h>
#include <wireless-phy.h>
#include <propagation.h>
#include <antenna.h>
#include <phy.h>

class RainWindModel: public Propagation {
public:
  RainWindModel();
  virtual double Pr(PacketStamp *tx, PacketStamp *rx, WirelessPhy *ifp);
  virtual double getDist(double Pr, double Pt, double Gt, double Gr,
                         double hr, double ht, double L, double lambda);

protected:
  double trought2RainWind(double Pt, double Gt, double Gr, double ht, double
hr, double L, double d);//our function to find Pr
  double findLossesRW(double d);
  double last_hr, last_ht;
  double crossover_dist;
};

#endif
```

7. FindLoss.awk

```
 BEGIN {
        DroppedCbr = 0;
        SentCbr = 0;
        SentDSR=0;
        AckReceived=0;
        AckSent=0;
         AckD=0;
       MessagesSent=0;
        MessagesRec=0;
        messagesDropped=0;

}
{
   action = $1;
   time = $2;
   type = $7;
   pktsize = $8;
   packet_id = $8;




   if (type=="cbr" && action == "s")

                SentCbr++;
 if (type=="cbr" && action == "r")

                AckSent++;
if (type=="cbr" && action == "D")

                DroppedCbr++;
```

```
if (type=="DSR" && action == "r")

            AckReceived++;
if (type=="DSR" && action == "D")

            AckD++;
 if (type=="DSR" && action == "s")

            SentDSR++;




if ( action == "D")
            messagesDropped++;
if ( action == "s")
            MessagesSent++;
if ( action == "r")
            MessagesRec++;

}

END {
 printf("--------------------Network Results-----------------------------\n\n");
printf("number of cbr packets sent:%d lost:%d receive:%d                    |\n",
SentCbr, DroppedCbr, AckSent);
printf("The loss rate is : %d percent ."
,(DroppedCbr*100)/(DroppedCbr+SentCbr) );

printf("------------------------------------------------------------------ \n");
printf("number of DSR packets sent : %d      |\n", SentDSR);
printf("number of DSR packets  dropped :%d   |\n", AckD);
printf("number of DSR packets received::%d   |\n", AckReceived);
printf("The loss rate is : %d percent ." ,(AckD*100)/(SentDSR+AckD));
```

```
 printf("---------------------------------------------------------------- \n");

printf("Total packets dropped:%d |\n",messagesDropped);
printf("Total packets received:%d |\n",MessagesRec);
printf("Total packets sent:%d |\n",MessagesSent);

 printf("----------------------------------------------------------------");

}
```

8. FindThroughput.awk

```
BEGIN {
        PacketSizeGen = 0;
        SimulationTime = 0;
        p_s= 0; #size
}
{

        event          = $1;     #; Event
        sim_time         = $2;     #; Time
        node           = $3;     #; Node
        trace_type      = $4;     #; Trace type
        error     = $5;
        PacketId        = $6;     #; Event ID :
        pkt_type        = $7;     #; , Data = cbr , ACK
        pkt_size        = $8;     #;  size



        # pick the part of number among "node" variables
```

```
        node = substr ( node, 2, length(node) - 2 );





node == end_node
        if ( event == "s"  && pkt_type == "cbr" ) {
                PacketSizeGen = PacketSizeGen + pkt_size;
        }
        SimulationTime = time;
}
END {

        throughput_0 = PacketSizeGen * 8 / ( SimulationTime * 1000000 );
        printf ( "%d %f\n", PacketSizeGen ,throughput);

}
```

9. firstzb.tcl

```
set val(chan)         Channel/WirelessChannel      ;# channel type
set val(netif)        Phy/WirelessPhy/802_15_4     ;# network ZIGBEE

set val(prop)         Propagation/FreeSpace     ;# propagation types
set val(mac)          Mac/802_15_4              ;# MAC type

set val(ifq)          Queue/DropTail/PriQueue      ;# interface queue type
set val(ll)           LL                           ;# link layer type
            ####ANTENNA TYPES#############3
set val(ant)          Antenna/OmniAntenna          ;# antenna model
set val(ifqlen)       150                          ;# max packet in ifq
```

```
set val(nn)          100                        ;# number of mobilenodes
            #### Protocols ################
set val(rp)          AODV                       ;
                        ###### We define Topography of terrain and nodes
#################
set val(x)           1000                       ;# X dimension of topography
set val(y)           1200                       ;# Y dimension of topography

set val(stop)        500                        ;# simulation period
                       #####ENERGY MODEL ANd Nodes Energy at start of
simulation#############

set val(energymodel)   EnergyModel                  ;# Energy Model

set val(initialenergy) 100                      ;# value


set ns                 [new Simulator]
set tracefd            [open trace-zb.tr w]
set namtrace           [open zb.nam w]
$ns trace-all $tracefd
$ns namtrace-all-wireless $namtrace $val(x) $val(y)

set dist(5m)  7.59113e-06
set dist(9m)  2.29381e-06
set dist(10m) 1.83278e-06
set dist(11m) 1.59182e-06
set dist(12m) 1.34437e-06
set dist(13m) 1.12989e-06
set dist(14m) 9.80099e-07
set dist(15m) 8.54789e-07

set dist(16m) 7.52997e-07
set dist(20m) 4.80696e-07
```

```
set dist(25m) 3.10015e-07
set dist(30m) 2.13234e-07

set dist(35m) 1.55675e-07
set dist(40m) 1.35675e-07
set dist(45m) 1.11007e-07
set dist(50m) 0.99984e-07
set dist(55m) 0.80981e-07
set dist(60m) 0.50437e-07
set dist(65m) 9.20987e-08
Phy/WirelessPhy set CSThresh_ $dist(40m)

Phy/WirelessPhy set RXThresh_ $dist(40m)

# set up topography object
set topo       [new Topography]
$topo load_flatgrid $val(x) $val(y)


create-god $val(nn)
# configure the 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) \
             -channel [new $val(chan)] \
             -topoInstance $topo \
             -agentTrace ON \
```

```
            -routerTrace ON \
            -macTrace  OFF \
            -movementTrace OFF \

            -energyModel $val(energymodel) \
            -initialEnergy $val(initialenergy) \

            -rxPower 50.28e-3 \
            -txPower 55.32e-3 \
              -idlePower 712e-6 \
              -sleepPower 144e-9
         #   -errProc NormalErr



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

        set mnode_($i) [$ns node]
}


##Create geographical groups to the topography
# 17 nodes will be at the first region

for {set i 3} {$i < 20 } { incr i } {

          $mnode_($i) set X_ rand(250.0) #[ expr {$val(x) * rand()} ]

          $mnode_($i) set Y_ rand(999.0) #[ expr {$val(y) * rand()} ]
          $mnode_($i) set Z_ 0
}
#60 nodes will be in the medium second central region
for {set i 20} {$i < $val(nn)-19 } { incr i } {
          $mnode_($i) set X_ rand(250.0,750.0) #[ expr {$val(x) * rand()} ]
```

```
        $mnode_($i) set Y_ rand(999.0) #[ expr {$val(y) * rand()} ]
        $mnode_($i) set Z_ 0
}
#20 nodes will be at the last region
for {set i 80} {$i < $val(nn)  } { incr i } {

        $mnode_($i) set X_ rand(750.0,999.0) #[ expr {$val(x) * rand()} ]

        $mnode_($i) set Y_ rand(999.0) #[ expr {$val(y) * rand()} ]

        $mnode_($i) set Z_ 0
}
# Position of Sink
$mnode_(0) set X_ 999.0

$mnode_(0) set Y_ 999.0
$mnode_(0) set Z_ 0.0
$mnode_(0) label "S0"
$mnode_(1) set X_ 0.0
$mnode_(1) set Y_ 999.0
$mnode_(1) set Z_ 0.0
$mnode_(1) label "S1"
$mnode_(2) set X_ 999.0
$mnode_(2) set Y_ 0.0
$mnode_(2) set Z_ 0.0
$mnode_(2) label "A2"

for {set i 0} {$i < $val(nn)} { incr i } {
        $ns initial_node_pos $mnode_($i) 10
}
```

```
#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $mnode_(10) $udp

set udp2 [new Agent/UDP]
$ns attach-agent $mnode_(2) $udp2
set sink [new Agent/Null]
$ns attach-agent $mnode_(0) $sink

set sink1 [new Agent/Null]

$ns attach-agent $mnode_(1) $sink1
#set em [new ErrorModel]
#$em set rate_ 0.1
#$em unit pkt
#$em ranvar [new RandomVariable/Uniform]
#$em drop-target [new Agent/Null]
#$ns link-lossmodel em $mnode_(10) $mnode_(0)
$n connect $udp $sink
$ns connect $udp2 $sink1
$udp set fid_ 2
$udp2 set fid_ 2

#Setup a CBR over UDP connection
set cbr2 [new Application/Traffic/CBR]
$cbr2 attach-agent $udp2

$cbr2 set type_ CBR
$cbr2 set packet_size_ 512
$cbr2 set rate_ 5.0Mb
$cbr2 set interval_ 2

set cbr [new Application/Traffic/CBR]
```

```
$cbr attach-agent $udp

$cbr set type_ CBR
$cbr set packet_size_ 512
$cbr set rate_ 5.0Mb
$cbr set interval_ 2
$ns at 5.0 "$cbr start"
$ns at 8.0 "$cbr2 start"
$ns at [expr $val(stop) - 5] "$cbr stop"
# move the central region to a destination
#for {set i 20} {$i < $val(nn)-20 } { incr i } {
#$ns at 50.0 "$mnode_(1) setdest 320 1.0 10.0"
#}
# Telling nodes when the simulation ends
for {set i 0} {$i < $val(nn) } { incr i } {
   $ns at $val(stop) "$mnode_($i) reset;"
}
# ending nam and the simulation
$ns at $val(stop) "$ns nam-end-wireless $val(stop)"

$ns at $val(stop) "stop"
$ns at [expr $val(stop) + 0.01] "puts \"The end of Experiment\"; $ns halt"
proc stop {} {
   global ns tracefd namtrace
   $ns flush-trace
   close $tracefd
   close $namtrace
}
$ns run
```

[10]

11 findEnergy.awk

```
BEGIN {
```

```
    # Records the total traffic packets sent, received and dropped.
    total_AGTpackets_sent = 0;
    total_AGTpackets_received = 0;
    total_AGTpackets_dropped = 0;

    signal=0;

    no_of_nodes=0;
    route_reqs=0;

    no_of_flows = 0;

    # FOR EACH ZB PKT TYPE: CM1, CM2, CM3, ...


    # Variables defining the reason for the dropped packets
    APS = 0;
    LQI = 0;

;

    hashes_ = 0;
    others_ = 0;
}
{
    event = $1;
    time = $2;
    node = $3;
    type = $4;
    reason = $5;
    node2 = $5;
    packetid = $6;
    mac_sub_type=$7;
```

```
    size=$8;
    source = $11;
    dest = $10;
    energy=$14;
}
END {
for (i=0; i<no_of_nodes-1; i++)
        {
            initial_energy[0] = initial_energy[0]+initial_energy[i+1];
            energy_used[0] = energy_used[0] + energy_used[i+1];
                    percent_energy_used[0]    =    percent_energy_used[0]    +
percent_energy_used[i+1];
        }

    initial_energy[0] = initial_energy[0]/no_of_nodes;
    energy_used[0] = energy_used[0]/no_of_nodes;
    percent_energy_used[0] = percent_energy_used[0]/no_of_nodes;

    printf("%f / %f / %f /",
            initial_energy[0],
            energy_used[0],
            percent_energy_used[0],
            >> "energy.txt";
}[11]
```

SECTION II

In this section there is information on what you will find on the cd that I provided. First of all ,cd includes all the trace files that are results of the simulations. With the trace files there are also NAM files that showing the positions of the nodes. These can be used for any purpose we want .In our project we used them only to find the throughput, the packet losses and the energy consumption. Also, in the cd there are several awk files that are

useful to manipulate the trace files so as to take the desirable results. Awk files are running only in Linux operating system. The operating system that I have used is ubuntu Linux 9.04 .The version of the network simulators that exist in the cd are of version ns -2.29 and 2.31 .I made the environmental condition files on ns-2.31 and continue with the changes to the AODV on 2.29. I have used both versions, because by trying to change some parts of AODV I had to face some errors. I informed from some forums that this error may be a bug of the ns-2 version 2.31. Also, so as to run ns-2 on Linux operating system we have to make a lot of updates of the operating system .These files can be found on internet .n order ns-2 to work normal need a lot of updates too. Also, ns-2 needs patches which can be found from internet. The configuration of ns-2 needs a lot of effort and is not described in my cd. Other files that exist in the cd are the c++ files and libraries such as Forest.cc, ForestModel.h, OkumuraForestModel.cc, OkumuraForestModel.h, RainWindModel.cc, and RainWindModel.h, Forest.cc. Model such as tworayground.cc and .h and free space model can be found at the ns-2 directory. Parts of those files have code from the tworayground.cc and .h files so as to decrease the possibility of errors while the compilation. Also the aodv.cc, aodv_packet .cc and h files with the changes exist in the cd .The original files of the AODV protocol can be found to the ns2 directory. The process to embody new code to ns2 is difficult and hides a lot of errors that have to be faced. Information can be found in several forums, but big personal effort and experiments are needed.