Tree based collection protocol for WSN.

Report submitted for the partial fulfillment of the requirements for the degree of Bachelor of Technology in
Information Technology

Submitted by

Ajay Kumar IT/2014/068

Arghyadeep Das IT/2014/070

Ayaz Zafar IT/2014/085

Under the Guidance of Mrs. Moumita Deb

RCC Institute of Information Technology
Canal South Road, Beliaghata, Kolkata – 700 015
[Affiliated to West Bengal University of Technology]

Acknowledgement

We would like to express our sincere gratitude to Mrs. Moumita Deb of the department of Information Technology, whose role as project guide was invaluable for the project. We are extremely thankful for the keen interestshe took in advising us, for the books and reference materials provided for the moral support extended to us.

Last but not the least we convey our gratitude to all the teachers for providing us the technical skill that will always remain as our asset and to all non-teaching staff for the gracious hospitality they offered us.

Place:RCCIIT, Kolkata

Date:

……

……

……

Department of Information Technology
RCCIIT,Beliaghata,
Kolkata – 700 015,
West Bengal, India

Approval

This is to certify that the project report entitled “Tree based collection protocol for WSN.” prepared under my supervision by Ajay Kumar (IT/2014/068), Arghyadeep Das (IT/2014/070), Ayaz Zafar (IT/2014/085), be accepted in partial fulfillment for the degree of Bachelor of Technology in Information Technology.

It is to be understood that by this approval, the undersigned does not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn thereof, but approves the report only for the purpose for which it has been submitted.

……..	……
Dr. Abhijit Das	Asst Prof.. Moumita Deb
H.O.D.	Asst. Professor
Department of Information Technology	Department of Information Technology
RCCIIT	RCCIIT

INDEX

Contents	***Page Numbers***
1. Introduction	5-6
2. Problem Definition	7
3. Literature Survey	8
4. SRS (Software Requirement Specification)	9
5. Planning	10
6. Design	11-14
7.Working	15-17
8. Results and Discussion	18
9. Coding	18-32
10. Conclusion and Future Scope	33
11. References / Bibliography	34
12. List of Figures	35

1. INTRODUCTION

WSN- Wireless Sensor Networks can be defined as a network of sensor which can communicate with each other wirelessly[1]. Large numbers of nodes are present in any wireless sensor network (WSN). Each of these nodes collect data and then forwards collected data to sink. Various characteristics of wireless sensor networks are 1.Low cost, 2.Ability to handle node failure, 3. Heterogeneity Of nodes. Due to these characteristics WSN can be used in weather monitoring, Disaster management, target tracking, homeland Security. WSN can be also used in Environmental Monitoring and Battlefield Surveillance.

Data collection is a basic task in Wireless Sensor Networks. In data collection sensor nodes measures the attributes of nodes and send to sink. Data is mainly collected in three stages: 1.Deployment stage: this stage deals with how deployment is done in sensing environment. 2.Data delivery stage: It includes how sensed data from each node is forwarded to the sink. 3.Control message dissemination stage: this is the last stage where collection commands or control message are disseminated from sink to all sensor nodes.

In many WSN applications, the deployment of sensor nodes is performed in an ad hoc fashion without careful planning and engineering. Once deployed, the sensor nodes must be able to autonomously organize themselves into a wireless communication network. Sensor nodes are battery-powered and are expected to operate without attendance for a relatively long period of time. In most cases it is very difficult and even impossible to change or recharge batteries for the sensor nodes. WSNs are characterized with denser levels of sensor node deployment, higherunreliability of sensor nodes, and sever power, computation, and memory constraints. Thus, the unique characteristics and constraints present many new challenges for the development and application of WSNs. Due to the severe energy constraints of large number of densely deployed sensor nodes, it requires a suite of network protocols to implement various network control and management functions such as synchronization, node localization, and network security. The traditional routing protocols have several shortcomings when applied to WSNs, which are mainly due to the energy-constrained nature of such networks[2].

Even if WSNs were originally thought to have static networking infrastructure, recent applications require sensing nodes to be mounted on mobile entities (human beings, mobile robots ,etc...).Think of the case of Wireless Body Area Networks (WBANs) consisting of a set of wear able to implant sensing devices which can communicate among themselves and or transmit data from the body to external traffic sinks .WBANs can be indeed useful whenever there is the need to monitor track nomadic people ,e.g. ,to monitors the battle field(military applications) ,patient sin nursing institutes (e-health applications),fire brigade sand policemen(security/safety applications). Whatever application environment, the use of WBANs and mobile sensors in general, brings into the world of WSNs the problem of effectively supporting the mobility of single nodes and or groups of nodes. Namely the mobile sensors need to be continuously connected to the external network to deliver theirs ensured data and vice versa ,an external control point may need to seamlessly contact them mobile sensors.

Paying attention to the various problems related to data collection in WSN we take a simple approach to connect the sensor nodes of a WSN in a bi-directional way in a tree like structure to accomplish successful data collection in a wide sensor environment.

We visualize the nodes of the network as the vertices of a tree and its edges as the connection links.

PROBLEM DEFINITION

As energy required in communication plays a major issue in energy depletion of the sensor node, we should minimize the number of transmissions along with efficient routing to achieve extended system lifetime [1]. We consider a wireless sensor system where nodes are homogeneous and sensed data are highly correlated. A sensor network for continuous monitoring is a typical example of such a system. In continuous monitoring application, energy constraint sensors periodically sense the environment and send the observe data to the base station(sink) with aggregation. In this paper, we propose a tree base routing technique for transferring the data in minimum hops. We assume that all nodes perform in network data aggregation. Our proposed approach generates a transmission schedule which contains a collection of routing paths. A routing path forms a tree that spans all the sensor nodes. A transmission schedule denotes how data is collected from each sensor and propagated to base station[sink]. It represents a collection of routing paths that network will follow to maximize lifetime.

LITERATURE SURVEY

Application for localization, tracking and monitoring of objects and people in in do or environments ,usually resort to hybrid sensor networks compose do fixed and mobile nodes. Hence mobility-aware routing protocols are required to support samples communication from to them mobile nodes and the data sinks.

The most common approach in the literature to handle mobility in WSNs([4],[5],[6]) consists in modifications of the Low Energy Adaptive Cluster Hierarchy (LEACH) protocol[10].In LEACH sensor nodes reorganized into local clusters with on node acting as Cluster-Head(CH) .The CH is responsible to deliver all the data coming from non-cluster-head nodes to the PAN coordinator (traffic sink) .Since non-cluster-head nodes have a TDMA schedule computed from their CH ,they can be switched on only in their time slot ,thus reducing energy consumption .On the contrary ,since cluster heads must be always active in order to receive data from the cluster and forward it to the PAN coordinator, their lifetime is limited .To avoid the death of a fixed set of sensor nodes, LEACH introduces a randomized rotation of the cluster heads in order to distribute the energy consumption among all nodes in the network.

In literature so far many hybrid metrics have been proposed [8][9]. We studied their merits and demerits. The first and foremost point is that all these metrics are combined with at most two estimators whether RSSI and PRR or SNR and LQI. Except SNR the noise floor is not taken into account. As discussed before a hybrid metric should closely relate to PRR and without PRR this can’t be achieved. In general LEACH and its modified versions supporting mobile nodes are based on single hop communication ,so they work under the assumption that all the nodes in the network can reach directly the Sink with the help of the path that has been created. This assumption is seldom realistic especially in do or environments where walls, furniture and people limit the radio range of wireless devices and multi-hop routing is unquestionably necessary.

As we are to deal with a flat bed routing protocol we also studied protocols like the AODV[10] to compare our results and analyze.

SRS (Software Requirement Specification)

OMNET++ is an object-oriented modular discrete event network simulation framework. It has a generic architecture, so it can be (and has been) used in various problem domains:

• modelling of wired and wireless communication networks

• Protocol modelling

• modelling of queuing networks

• modelling of multiprocessors and other distributed hardware systems

• validating of hardware architectures

• evaluating performance aspects of complex software systems

• in general, modelling and simulation of any system where the discrete event approach is suitable, and can be conveniently mapped into entities communicating by exchanging messages.

OMNET++ itself is not a simulator of anything concrete, but rather provides infrastructure and tools for writing simulations. One of the fundamental ingredients of this infrastructure is component architecture for simulation models. Models are assembled from reusable components termed modules. Well-written modules are truly reusable, and can be combined in various ways like LEGO blocks.

Castalia is a simulator for Wireless Sensor Networks (WSN), Body Area Networks and generally networks of low-power embedded devices. It is based on the OMNET++ platform and used by researchers and developers to test their distributed algorithms and/or protocols in a realistic wireless channel and radio model, with a realistic node behavior especially relating to access of the radio. Castalia uses the lognormal shadowing model as one of the ways to model average path loss, which has been shown to explain empirical data in WSN.[1] It also models temporal variation of path loss in an effort to capture fading phenomena in changing environments (i.e., the nodes or parts of the environment are moving). Castalia's temporal variation modeling is designed to be fitted to measured data instead of making specific assumptions on the creation of fast fading.

Planning

The basic structure of a connected Wireless Sensor Network is formed similar to the figure of a tree. So to achieve this type of a structure and connect nodes with each other so that the best possible and fast connection is achieved. Various routing protocols make use of different tree based structure to connect the nodes.

Here we plan to construct a routing protocol for a collection of nodes in a particular environment. We are to use different factors on which wireless data transmission can be achieved in an optimized manner. Whenever a node is planted it runs on a cell or battery that has to be used wisely to ensure its long battery life as a battery replacement at all distant nodes are not possible. There are various estimators which give us data about the link or broadcast quality of the nodes.

We plan to consider the remaining energy of each node that makes a transaction with another node. This measure of the remaining energy of the node would be used for parent selection by any random node. We studied whenever a transaction of data packets takes place energy is lost and link quality depreciates. Thus keeping an eye to these factors we combine the concept of remaining energy with this. The sink behaves as the root node which has got the virtue of becoming a root node for all tree like structures that can be formed on the given network topology.

We are considering a network layout or topology that has several nodes and a sink. The sink initiates the first event of the routing algorithm. It broadcasts a sink message, whenever a node receives this message it becomes the first level nodes. The first level nodes basically act like cluster heads which can connect to the sink directly. These nodes then start the Topology setup process by broadcasting their level. The nodes first follow an algorithm to set the level of each node in the hierarchy. This leveling helps the nodes to decide its parent at time of tree building. This parent selection or tree building is done on the basis of the leveling already created. Furthur the best quality parent is selected from a set of higher level nodes by comparing them on the basis of their remaining energy.

DESIGN

Considering various tree based protocols and the methods used by them a new algorithm based on tree topology was written. To implement this algorithm in a network stimulation environment firstly a network stimulator "OMNET ++" was used as the base to run all stimulations. As we have used parameters that relate to the remaining energy of the nodes we used the Energy Manager- module of the "Green Castalia" framework to implement our algorithm. To code and stimulate the algorithm on a simulator mainly three different files had to be created, first one is the ".ned" or Network Description file that has all the data regarding the topology of the network. The second one is code which defines the work procedure of an individual node. The algorithm is coded on C++ language. This file is known as the source file of the stimulation. The last requirement of the implementation process is the stimulation environment setup file, in OMNET++ a file named "omnetpp.ini" is responsible for configuring the environment and setting up all the parameters.

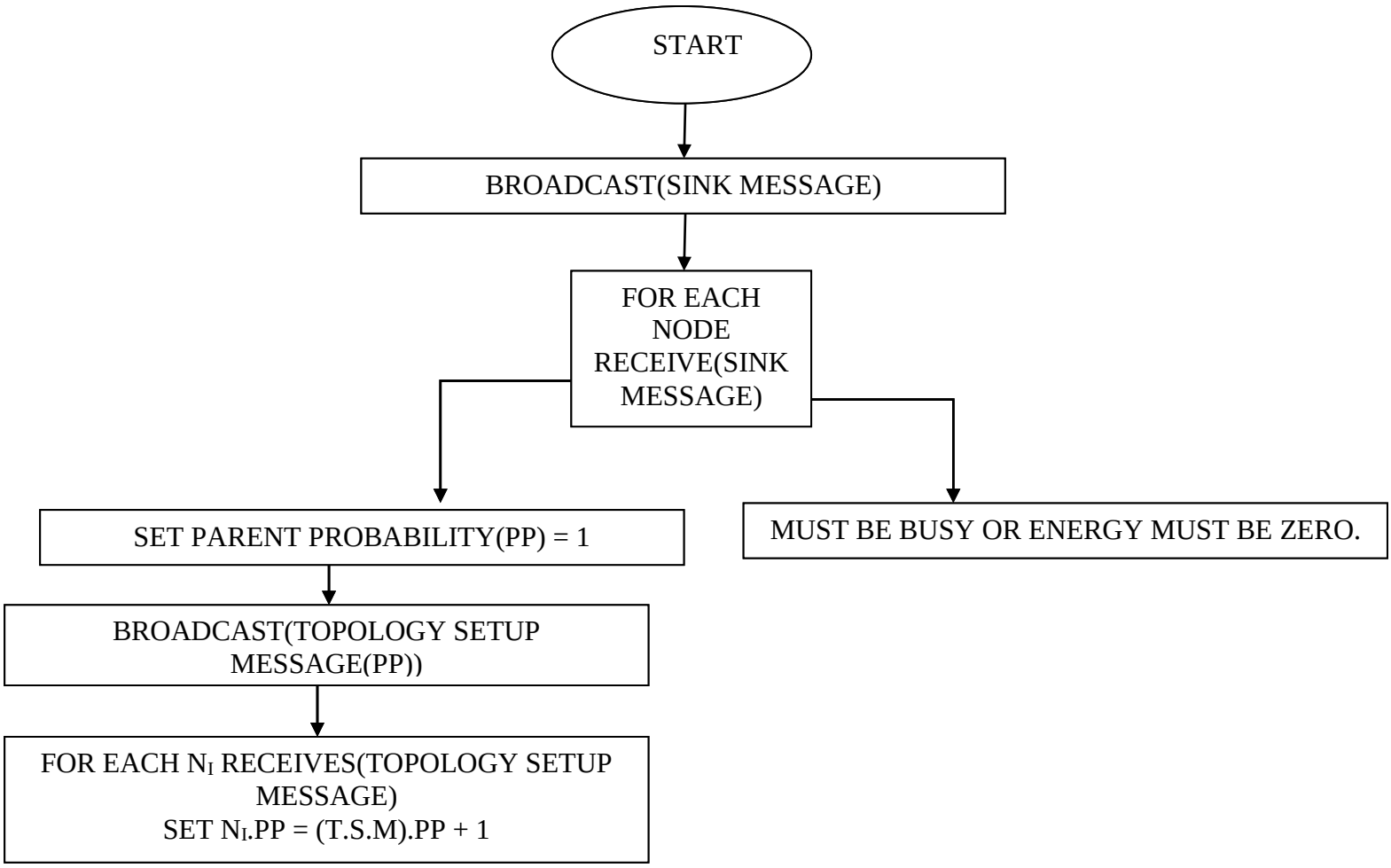

Fig1: Flowchart for level

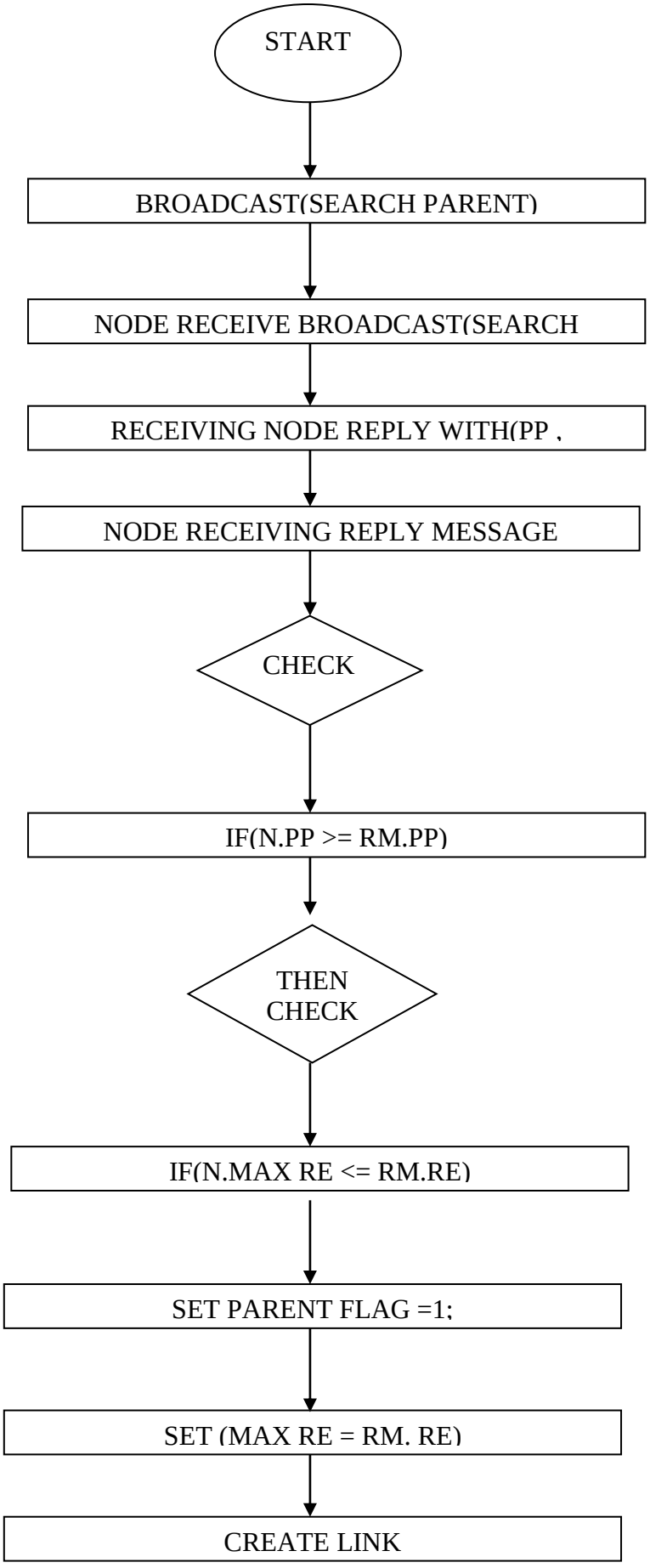

Fig2: Flowchart for tree creation.

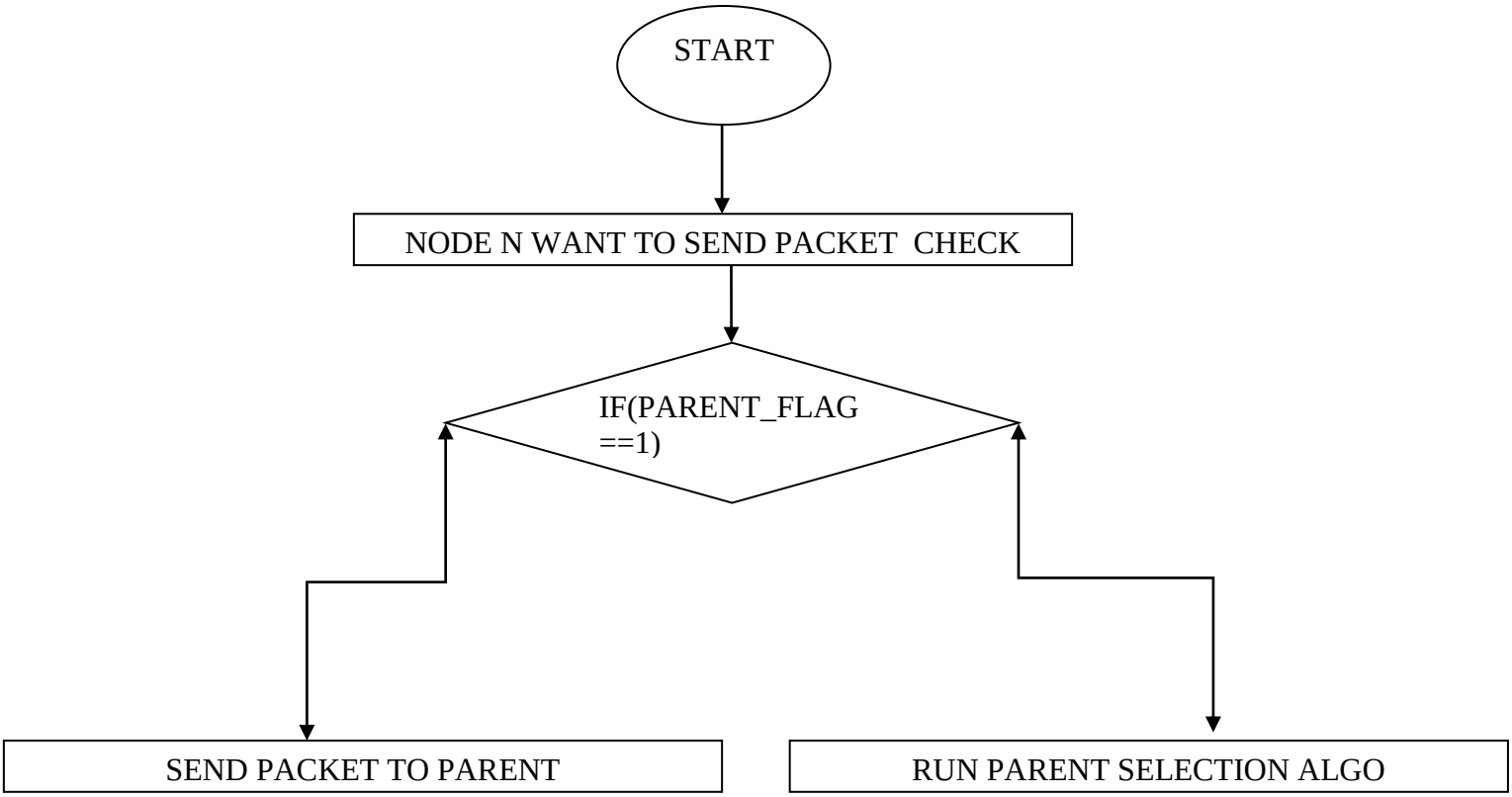

Fig3: Packet Sending Flowchart

ALGORITHM FOR NODE LEVEL SETTING.

{
SN = SINK NODE ,
N = SET OF NODES ,
SM = SINK MESSAGE ,
PP = PARENT PROBABILITY ,
T_M_S = TOPOLOGY SETUP MESSAGE
}
BEGIN

1. SN -> Broadcast(SM)
2. For each N_i € N receive (SM)
   {
   Set PP = 1
   N_i -> Broadcast(T_S_M (PP))
   }
3. For each N_i € N receive (T_S_M)
   {
   Set N_i.PP = (T_S_M).PP + 1
   }

ALGORITHM FOR PARENT SELECTION AND TREE BUILDING.

{
SP = SEARCH PARENT MESSAGE ,
N = SET OF NODES ,
RE = REMAINING ENERGY ,
PP = PARENT PROBABILITY ,
}
BEGIN

1. Node N_i -> Broadcast(SP)
2. For each N_j € N receives(SP)
   {
   Reply with RM(PP , RE)
   }
3. For each N_j € N receives(RM) check
   If(N_j.PP >= RM.PP)
   Then check
   If(N_j. Max RE <= RM.RE)
   Set parent flag=1;
   Set max RE = RM.RE;
   Create link;

ALGORITHM FOR PACKET SENDING.

BEGIN

1. For any node N_j € N which want to send packet check
   If(parent_flag == 1)
   {
   Send packet to parent
   }
   Else
   {
   Run parent selection algorithm
   }
2. For any node N_k € N which receive packet check
   If
   {
   N_k.PP == 0;
   Break;
   }
   Else if(N_k.PP > 0)
   ➔ Goto 1.

WORKING:

Node leveling:

The routing protocol works on the above said algorithm, at the start all the nodes initialize and the sink node Broadcasts a message tagged as “sink message”. Whenever this message is received by a primary node its Parent Probability (PP) is set as 1. This means that all the nodes that receive the “sink message” are the closest to the sink or the first level parents. These first level nodes act like cluster heads and start broadcasting a message tagged as TOPOLOGY_SETUP_MESSAGE (T_S_M) which contains the PP of the node. Any node in the network that receives this type of a message sets its PP one more than the PP of the node from which the message was received. This process completes the leveling of the nodes according to their probable levels in the tree.

Parent selection and tree building:

After the leveling of all the nodes are complete the nodes now start following an algorithm to select their parent and establish the link. To do this the nodes broadcast a message tagged as “Search Parent (SP)”. Whenever a node gets such a message it replies back with a message called the “Reply message (RM)”, this message contains the PP of the replying node and its Remaining Energy (RE).

When the node which had sent the search parent message accepts the first reply message its sets the replying node as its parent if the PP of the replying node is smaller than the PP of the sending node. This ensures that a node distant from the sink always selects a node more closer to the sink than itself. The remaining energy received with the reply message is stored in a variable named as “Max_RE” Now in a much bigger scenario where the node density is higher a node broadcasting the search parent message receives more than one reply from nodes of higher levels. The final parent selection from the set of all replying nodes is done on the basis of the remaining energy of the replying node. After the first replying node is set as parent if any other reply message is received with lesser “PP” value then the remaining energy values of the current parent and the new reply message are compared, the node with the greater remaining energy value is set as parent and the link is established. This completes the procedure of parent selection. This results in a tree like linkage pattern where the sink acts as the root node.

Packet sending:

Once the tree is created and every node is linked with its most suitable parent the packet sending procedure starts. Whenever a node has a packet ready to send it checks if its parent has already been set, if true it passes the packet to the destination or parent node otherwise runs the parent selection algorithm. This parent node now acts as a node ready to transfer its packet to a higher level. This procedure of packet handover is carried out until the packet reaches the sink or reaches to a level 0 node.

The following figures (Fig1, Fig2) explain two different steps in the tree building procedure the first one is the leveling scenario and the second one shows the formation of the tree. The figures clearly explain how the network topology is created and the tree like structure is found. After the creation of this tree all the sensor nodes have a fixed parent to which it sends its packet.

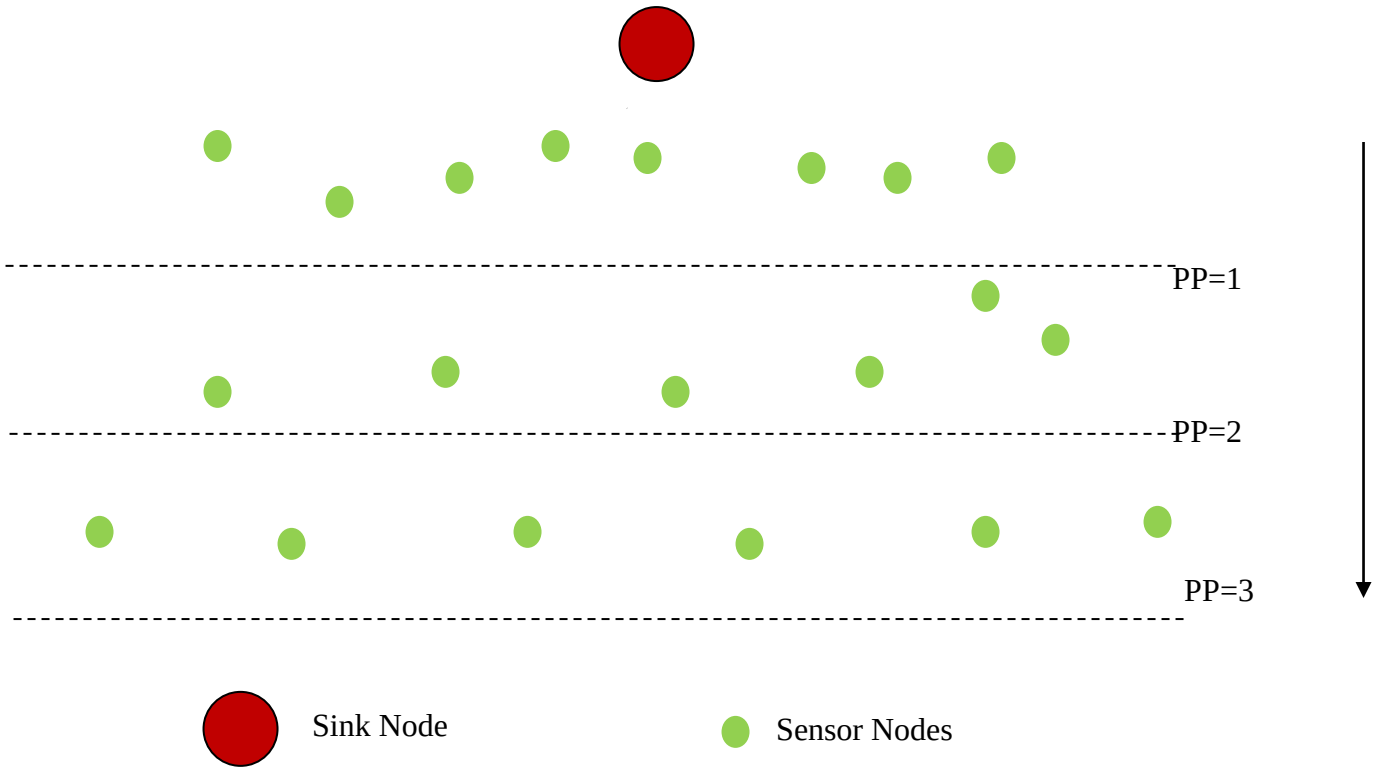

Fig 4: Leveling Scenario.

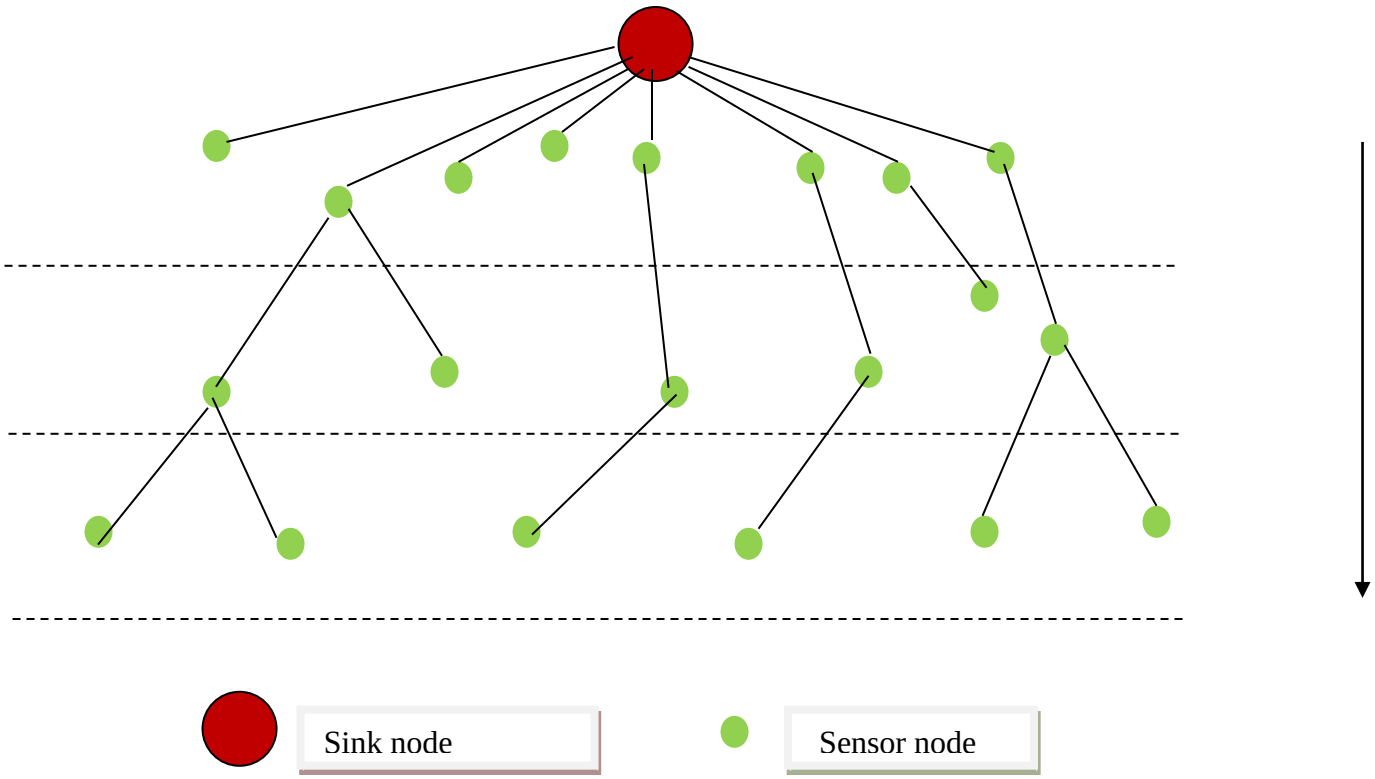

Fig 5: Tree topology setup

RESULT

Simulation experiments were conducted to analyze the performance of our tree routing protocol using the Castalia Framework [8], which is a widely used network simulator for WSNs based on OMNET++ simulator [9]. The simulations were carried out and repeated 20 times with different random seed numbers in order to obtain a confidence interval of 95%. Due to limited available nodes in the testbed, simulation experiments were conducted to evaluate the Tree routing protocol and compare its performance with AODV [10] in terms of energy-efficiency, latency and data delivery in a large-scale scenario, i.e., using a monitoring area of 100 m × 100 m and up to 100 nodes. AODV is a standard reference, which can be considered as a benchmark solution for flat routing protocols and scenarios. The simulation environment was created and the topology setup was successful. Simulation results were compared with other widely accepted protocols that are generally used for communication routing processes and it was found to be working fine on various topologies defined by us. The protocol runs smoothly without missing out any node in the topology defined for that simulation. The screenshots of one fully completed simulation are attached with this documents. The result is formed as a trace file in the simulations folder every time the simulation is run and completed. These trace files are then analyzed and data about the efficiency of the protocol is extracted from it.

Screenshots

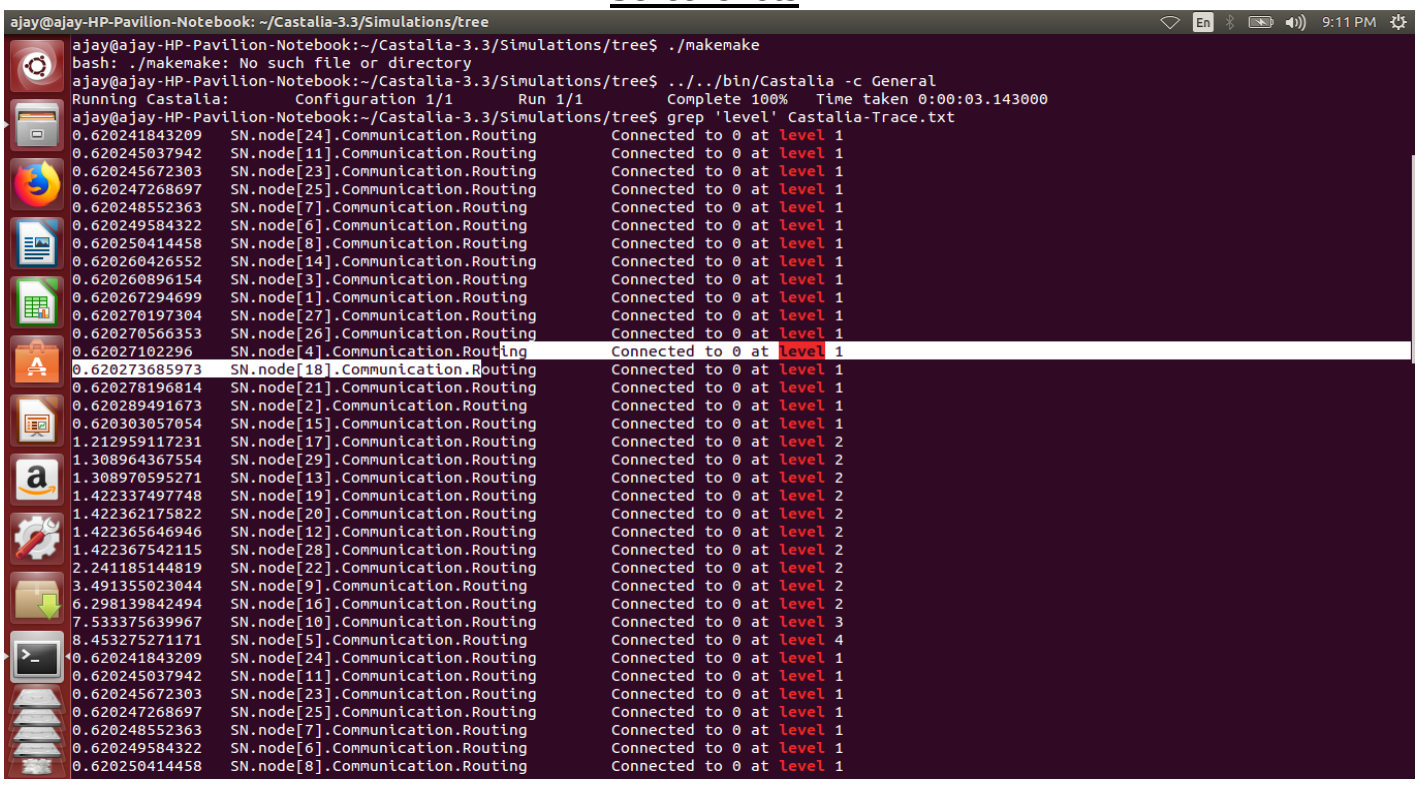

Fig6: Screenshot (i)

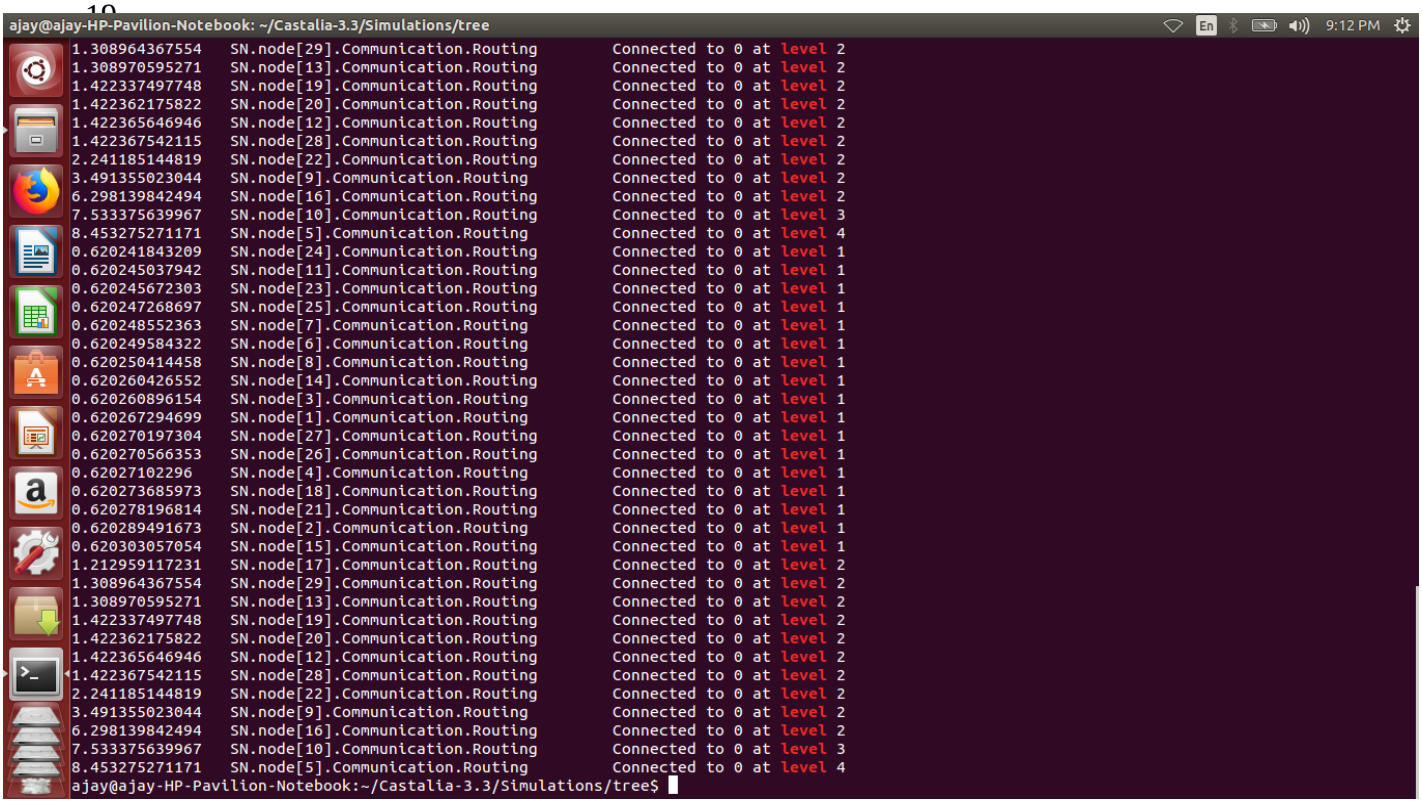

Fig7: Screenshot (ii)

Code: Source code for actions of an individual node.

```
#include "TreeRouting.h"
#include "ResourceManager.h"
#include "VirtualEnergyPredictor.h"
#include "VirtualEnergyManager.h"
#include "VirtualEnergyStorage.h"
#include "Battery.h"

Define_Module(TreeRouting);

void TreeRouting::startup()
{
        /*--- The .ned file's parameters ---*/
        //percentage = par("percentage");
        //roundLength = par("roundLength");
        isSink = par("isSink");
        //slotLength = par("slotLength");
```

```
        advPacketSize = par("advPacketSize");
        joinPacketSize = par("joinPacketSize");
        tdmaPacketSize = par("tdmaPacketSize");
        dataPacketSize = par("dataPacketSize");
        applicationID = par("applicationID").stringValue();
        t = par("t");

        /*--- Class parameters ---*/
        //CHcandidates.clear();
        //clusterMembers.clear();
//AddM
//      CHCompetitors.clear();
        //CHNeighbors.clear();
//End
        //roundNumber=0;
//      probability = 0;
        isCH = false;
        endFormClus = false;
        //isCt = false;
        remEnergy = 0;
        //probability = 0.40;
        //RComp = 0;
        //CHCompetitors.clear();
        //bool found = false;
        //flag = 0;
        //den = 0;
        //timer = 0;
//AddM To access remaining Energy
        engyMgr =check_and_cast<VirtualEnergyManager*>(getParentModule()-
>getParentModule()->getSubmodule("ResourceManager")-
>getSubmodule("EnergySubsystem")->getSubmodule("EnergyManager"));
//      remEnergy = engyMgr->getCurrentEnergy();
//      nodeC = engyMgr->getNodeCategory();
//AddM To access Max energy of Battery
        //VirtualEnergyStorage* engyStore
=check_and_cast<VirtualEnergyStorage*>(getParentModule()->getParentModule()-
>getSubmodule("ResourceManager")->getSubmodule("EnergySubsystem")-
>getSubmodule("EnergyStorage")->getSubmodule("RechBatteries",0));

        //maxEnergy=engyStore->getMaxEnergy();
//trace() << "MAx energy is" << maxEnergy;
//      maxEnergy=1800;
//AddM To access amount of predicted Energy
```

```
//	VirtualEnergyPredictor* predictorModule =
check_and_cast<VirtualEnergyPredictor*>(getParentModule()->getParentModule()-
>getSubmodule("ResourceManager")->getSubmodule("EnergySubsystem")-
>getSubmodule("EnergyPrediction"));

//	predTime = roundLength;
//	trace() << "prediction Time" << predTime;
//	predHarvPwr = predictorModule->getPrediction( predTime );
	//predHarvPwr = 0.015;

//check how to set timer
//	if(!isSink) setTimer(START_ROUND, netSetupTimeout);

//AddM

	cModule *appModule = getParentModule()->getParentModule()-
>getSubmodule("Application");
	if (appModule->hasPar("isSink"))
		isSink = appModule->par("isSink");
	currentLevel = tmpLevel = isSink ? 0 : NO_LEVEL;
	currentSinkID = tmpSinkID = isSink ? self : NO_SINK; //whether necessary

	isConnected = (isSink) ? true : false; //check
	isScheduledNetSetupTimeout = false;		//Relation between isconnected????
	currentSequenceNumber = 0;			//what is the use
	isJoin = false;
	knwRelay = false;
	if (isSink && (roundNumber == 0))
		sendTopologySetupPacket();

	//readXMLparams();
}
// AddM

void TreeRouting::sendTopologySetupPacket()
{
	TreeRoutingPacket *setupPkt =
	   new TreeRoutingPacket("Tree routing setup packet", NETWORK_LAYER_PACKET);
	setupPkt->setTreeRoutingPacketKind(TREE_ROUTING_TOPOLOGY_SETUP_PACKET);
	setupPkt->setSource(SELF_NETWORK_ADDRESS);
	setupPkt->setDestination(BROADCAST_NETWORK_ADDRESS);
	setupPkt->setSinkID(currentSinkID);
	setupPkt->setSenderLevel(currentLevel);
	toMacLayer(setupPkt, BROADCAST_MAC_ADDRESS);
```

```
}



void TreeRouting::fromApplicationLayer(cPacket *pkt, const char *destination) // Add control
packet
{
	if(!isSink)			//if its not a sink
	{
		string dst(destination);
		TreeRoutingPacket *netPacket = new TreeRoutingPacket("Tree routing data
packet", NETWORK_LAYER_PACKET);
		netPacket->setSource(SELF_NETWORK_ADDRESS);
		netPacket->setDestination(destination);
		encapsulatePacket(netPacket, pkt);
		toMacLayer(netPacket, resolveNetworkAddress(destination));  //??
		/*if (!isCH && endFormClus)
		{
			CHInfo info = *CHcandidates.begin();
			stringstream buffer;
			buffer << info.src;
			string dst = buffer.str();
			netPacket->setDestination(dst.c_str());
			bufferPacket(netPacket);
		}
		else if (!isCH && !endFormClus)
		{
			tempTXBuffer.push(netPacket);
		}

		else if (isCH)
		{
			bufferAggregate.push_back(*netPacket);
		}*/
	}
}

void TreeRouting::fromMacLayer(cPacket *pkt, int macAddress, double rssi, double lqi){
	TreeRoutingPacket *netPacket = dynamic_cast <TreeRoutingPacket*>(pkt);

	if (!netPacket)
		return;

	switch (netPacket->getTreeRoutingPacketKind()) {
```

```
// AddM
                case TREE_ROUTING_TOPOLOGY_SETUP_PACKET:{
                        if (isSink)
                                break;
                        if (!isScheduledNetSetupTimeout) {
                                isScheduledNetSetupTimeout = true;
                                setTimer(TREE_ROUTING_TOPOLOGY_SETUP_TIMEOUT, 0.5);
                                tmpLevel = NO_LEVEL;
                                tmpSinkID = NO_SINK;
                        }
                        if (tmpLevel == NO_LEVEL || tmpLevel > netPacket->getSenderLevel()) {
                                tmpLevel = netPacket->getSenderLevel();
                                tmpSinkID = netPacket->getSinkID();
                        }
                        break;
                }

//check

        }
}

void TreeRouting::timerFiredCallback(int index)
{
        switch (index) {
//AddM for checking

        case TREE_ROUTING_TOPOLOGY_SETUP_TIMEOUT:
        {
                if (index != TREE_ROUTING_TOPOLOGY_SETUP_TIMEOUT)
                return;
           isScheduledNetSetupTimeout = false;
            if (tmpLevel == NO_LEVEL) {
                setTimer(TREE_ROUTING_TOPOLOGY_SETUP_TIMEOUT, 0.6);
                isScheduledNetSetupTimeout = true;
                }
                else if (currentLevel == NO_LEVEL) {
                //Broadcast to all nodes of currentLevel-1
                currentLevel = tmpLevel + 1;
                currentSinkID = tmpSinkID;

                if (!isConnected) {
                        isConnected = true;
```

```
//                      sendControlMessage(MPRINGS_CONNECTED_TO_TREE);
                        trace() << "Connected to " << currentSinkID << " at level " <<
currentLevel;
                        setTimer(REFRESH_HELLO_TIMER,1.75);
//                      if (!TXBuffer.empty())
//                              processBufferedPacket();
                }else {
//                      sendControlMessage(MPRINGS_TREE_LEVEL_UPDATED);
                        trace() << "Reconnected to " << currentSinkID << " at level " <<
currentLevel;
                }
                sendTopologySetupPacket();
        }

                tmpLevel = isSink ? 0 : NO_LEVEL;
                tmpSinkID = isSink ? self : NO_SINK;
            break;
       }
// End
        }

}
```

Code: Definition of user defined header files used'

```
#ifndef _TREEROUTING_H_
#define _TREEROUTING_H_

#include <list>
#include <map>
#include <queue>
#include <vector>
#include <omnetpp.h>
#include <algorithm>
#include <string>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include <sstream>
#include "VirtualRouting.h"
#include "VirtualApplication.h"
#include "TreeRoutingPacket_m.h"
#include "ApplicationPacket_m.h"
```

```
#include "NoMobilityManager.h"
#include "ResourceManager.h"

#define NO_LEVEL  -110
#define NO_SINK   -120

using namespace std;

enum TreeRoutingTimers {
        START_ROUND = 1,
        SEND_ADV = 2,
        JOIN_CH = 3,
        MAKE_TDMA = 4,
        START_SLOT = 5,
        END_SLOT = 6,
        TREE_ROUTING_TOPOLOGY_SETUP_TIMEOUT = 7,
        ADJACENT_CH_ADVERTISEMENT = 8,
        START_COMPETITION = 9,
        FINISH_COMPETITION = 10,
        SEND_QUIT = 11,
        TREE_ROUTING_DENSITY_CAL_TIMER = 12,
        REFRESH_HELLO_TIMER = 13,
        CANCEL_HELLO_TIMER = 14,

        };

struct CHInfo
{
        int src;
        double rssi;
        double currentLevel;
};
//AddM
struct CHCompete{
        int src;                        //sorgente (ID)
        double RComp;
        double currentLevel;            //raggio di competenza

};

struct CHAdjacent{
        int src;
        double remainingEnergy;
        double currentLevel;
        };
```

```
struct Neighbor{
        int src;
        int density ;
        };

class VirtualEnergyManager;
class VirtualEnergyPredictor;
class VirtualEnergyStorage;
class TreeRouting : public VirtualRouting {

private:

        string applicationID;
        int advPacketSize;
        int tdmaPacketSize;
        int dataPacketSize;
        int joinPacketSize;
// For topology setup
        double netSetupTimeout;
        double denSetupTimeout;
        double maxPower;
        double sensibility;
        double aggrConsumption;

        double slotLength;
        int clusterLength;
        double percentage;
        double probability;

        int den;
        int roundNumber;
        int dataSN;

        bool isCH;
        bool isSink;
        bool isCt;
        bool endFormClus;
        int flag;
        double t ;
        bool isJoin;
        bool knwRelay;
        vector<RoutingPacket> bufferAggregate;
        vector<int> powers;
        queue <cPacket *> tempTXBuffer;
        vector <int> clusterMembers;
        list <CHInfo> CHcandidates;
```

```
        list <Neighbor> NeighborList;
//AddM
        ResourceManager* resMgrModule;
        VirtualEnergyManager* engyMgr;
        VirtualEnergyStorage* engyStore;
        VirtualEnergyPredictor* predictorModule;
        double remEnergy;
        double maxEnergy;
        char nodeC;
        simtime_t predTime,roundLength;
        double predHarvPwr;
        double timer;
        bool found;
        bool newTentativeCH;
        double RComp;
        double rNumber;

        int currentSinkID;
        int currentLevel;
        int tmpSinkID;
        int tmpLevel;
                        //is a .ned file parameter of the Application module
        bool isConnected;       //attached under a parent node
        bool isScheduledNetSetupTimeout;
        list <CHCompete> CHCompetitors;                 //aggiunto da EEUC
        list <CHAdjacent> CHNeighbors;
        CHAdjacent nextRelay;
        CHAdjacent nextRelayForward;
//      CHInfo nextR;
        CHInfo nextRe;
        map<string, int> hmeluc;
protected:

        void startup();
        void fromApplicationLayer(cPacket *, const char *);
        void fromMacLayer(cPacket *, int, double, double);
        void timerFiredCallback(int);
        void processBufferedPacket();
        void sendTopologySetupPacket();
        void sendHelloMessage();
        void print_neighbor();
        void sendAggregate();
        void setPowerLevel(double);
        void setStateSleep();
        void setStateRx();
        void levelTxPower(int);
```

```
        void readXMLparams();

};
bool cmpaRssi(CHInfo a, CHInfo b);
bool compLevel(CHAdjacent a, CHAdjacent b);

#endif
```

Code: Definition of Topology Setup Message.

```
cplusplus {{
#include "RoutingPacket_m.h"
}}

class RoutingPacket;


enum TreeRoutingPacket_Type
{
        TREE_ROUTING_ADV_PACKET  = 1;
        TREE_ROUTING_JOIN_PACKET  = 2;
        TREE_ROUTING_TDMA_PACKET         = 3;
        TREE_ROUTING_DATA_PACKET         = 4;
        TREE_ROUTING_TOPOLOGY_SETUP_PACKET = 5;
        TREE_ROUTING_AGGREGATED_DATA_PACKET = 6;
        TREE_ROUTING_ADJACENTCH_PACKET = 7;
        TREE_ROUTING_COMPETECH_PACKET = 8;
        TREE_ROUTING_FINALHEAD_PACKET = 9;
        TREE_ROUTING_QUITELECTION_PACKET = 10;
        TREE_HELLO_MESSAGE_PACKET = 11;
};

packet TreeRoutingPacket extends RoutingPacket
{
        int TreeRoutingPacketKind enum (TreeRoutingPacket_Type);
        int schedule[];
//      char nodeCategory;
        int sinkID;                     // 2 bytes
        int senderLevel;
        double remainingEnergy;
        double RComp;
};
```

Code: Simulation Environment Configuration File.

```
//network
include ../Parameters/Castalia.ini
include ../Parameters/MAC/CSMA.ini
sim-time-limit = 100s
SN.field_x = 50      #40
SN.field_y = 50     #10
SN.numNodes = 30

SN.deployment = "[0..29]->uniform"
//SN.node[0].xCoor = 50
//SN.node[0].yCoor = 70

//SN.node[1].xCoor = 10
//SN.node[1].yCoor = 10

SN.node[2].xCoor = 35
SN.node[2].yCoor = 10

//SN.node[3].xCoor = 15
//SN.node[3].yCoor = 20

//SN.node[4].xCoor = 27
//SN.node[4].yCoor = 20

//SN.node[5].xCoor = 50
//SN.node[5].yCoor = 17

//SN.node[6].xCoor = 36
//SN.node[6].yCoor = 30

//SN.node[7].xCoor = 70
//SN.node[7].yCoor = 24

//SN.node[8].xCoor = 80
//SN.node[8].yCoor = 15

//SN.node[9].xCoor = 90
//SN.node[9].yCoor = 27
```

```
//SN.node[10].xCoor = 15
//SN.node[10].yCoor = 40

//SN.node[11].xCoor = 30
//SN.node[11].yCoor = 45

//SN.node[12].xCoor = 55
//SN.node[12].yCoor = 40

//SN.node[13].xCoor = 70
//SN.node[13].yCoor = 45

//SN.node[14].xCoor = 95
//SN.node[14].yCoor = 45

//SN.node[15].xCoor = 80
//SN.node[15].yCoor = 55

//SN.node[16].xCoor = 55
//SN.node[16].yCoor = 58

//SN.node[17].xCoor = 27
//SN.node[17].yCoor = 60

//SN.node[18].xCoor = 15
//SN.node[18].yCoor = 60

//SN.node[19].xCoor = 44
//SN.node[19].yCoor = 50

//SN.node[20].xCoor = 89
//SN.node[20].yCoor = 65

//SN.node[21].xCoor = 70
//SN.node[21].yCoor = 66

//SN.node[22].xCoor = 38
//SN.node[22].yCoor = 70

//SN.node[23].xCoor = 52
//SN.node[23].yCoor = 79

//SN.node[24].xCoor = 20
//SN.node[24].yCoor = 75
```

```
//SN.node[25].xCoor = 30
//SN.node[25].yCoor = 85

//SN.node[26].xCoor = 12
//SN.node[26].yCoor = 88

//SN.node[27].xCoor = 55
//SN.node[27].yCoor = 90

//SN.node[28].xCoor = 70
//SN.node[28].yCoor = 95

//SN.node[29].xCoor = 80
//SN.node[29].yCoor = 85




SN.node[*].Communication.Radio.mode = "normal"

// Traces
SN.wirelessChannel.collectTraceInfo = false
SN.node[*].Communication.Radio.collectTraceInfo = false
SN.node[*].Communication.MAC.collectTraceInfo = false
SN.node[*].Communication.Routing.collectTraceInfo = true
SN.node[*].Application.collectTraceInfo = true
SN.node[*].SensorManager.collectTraceInfo = false
SN.node[*].ResourceManager.collectTraceInfo = false


//MAC


#-----CSMA-CA-----#

SN.node[*].Communication.MACProtocolName = "TunableMAC"
SN.node[*].Communication.MAC.listenInterval = 10
SN.node[*].Communication.MAC.dutyCycle = 0.1
SN.node[*].Communication.MAC.beaconIntervalFraction = 1.0
SN.node[*].Communication.MAC.phyDataRate = 250
SN.node[*].Communication.MAC.phyFrameOverhead = 6
SN.node[*].Communication.MAC.macPacketOverhead = 9

//Routing
```

```
SN.node[*].Communication.RoutingProtocolName = "TreeRouting"
SN.node[*].Communication.Routing.netBufferSize = 1000
SN.node[0].Communication.Routing.isSink = true
SN.node[*].Communication.Routing.slotLength = 0.2
SN.node[*].Communication.Routing.roundLength = 30s
SN.node[*].Communication.Routing.percentage = 0.05
SN.node[*].Communication.Routing.powersConfig = xmldoc("powersConfig.xml")


// Application

SN.node[*].ApplicationName = "ThroughputTest"
SN.node[*].Application.packet_rate = 1
SN.node[*].Application.constantDataPayload = 2000


// Wireless Channel

SN.wirelessChannel.onlyStaticNodes = true
SN.wirelessChannel.sigma = 0
SN.wirelessChannel.bidirectionalSigma = 0
SN.wirelessChannel.pathLossExponent = 2.0    # Free Space



// Radio

SN.node[*].Communication.Radio.RadioParametersFile = "../Parameters/Radio/CC2420.txt"
SN.node[*].Communication.Radio.TxOutputPower = "-10dBm"
```

Conclusion and Future Scope

This article has presented a tree based routing protocol together with load balance scheme based on Remaining energy for applications on Wireless Sensor network, such as automation of comfortable homes and offices, healthcare, environmental monitoring, and smart parking. This tree routing protocol combines a reliable scheme for route discovery and load balance mechanism, which provides high reliability, QoS-awareness and energy-efficiency. Moreover, it proposes an end-to-end route selection scheme based on cross-layer information with a minimal overhead. Nodes become energy efficient by sending the residual energy to their neighboring nodes with the aid of a piggyback and on-demand scheme.

Reference

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, “A survey on sensor networks,” Communications Magazine, IEEE, vol. 40, no. 8, pp. 102 – 114, aug. 2002.

[2] Routing protocol in WSN – A Survey by Shio Kumar Singh, Mk Singh, Dk singh.

[3] LVM- Designing a Link Quality Estimator for Sensor Nodes by combining available Estimators- Moumita Deb, Subhobrata Roy, Bidushi Saha, Paromita Das, Moumita Das

[4] D.-S. Kim and Y.-J. Chung, “Self-Organization Routing Protocol Sup- porting Mobile Nodes for Wireless Sensor Networks,” in Computer and Computational Sciences, 2006. IMSCCS ’06. First International Multi- Symposiums on, 2006.

[5] G. Santhosh Kumar, M. V. Vinu Paul, and K. Poulose Jacob, “Mobility Metric based LEACH-Mobile Protocol,” in Advanced Computing and Communications, 2008. ADCOM 2008. 16th International Conference on, 2008

[6] L. Nguyen, X. Defago, R. Beuran, and Y. Shinoda, “An Energy Efficient Routing Scheme for Mobile Wireless Sensor Networks,” in Wireless Communication Systems. 2008. ISWCS ’08. IEEE International Sym- posium on, 2008.

[7] R. Michele, Junaid Ansari, Janne Riihijarvi. April 1, 2008. “Designing a Reliable and Stable Link Quality Metric for Wireless Sensor Networks Department of Wireless Networks”, REALWSN’08, April 1, 2008

[8] . Boulis, A. Castalia, A Simulator for Wireless Sensor Networks and Body Area Networks, Version 2.2. User’s Manual; NICTA: Canberra, Australia, 2009.

[9] . Varga, A. The OMNeT++ Discrete Event Simulation System. In Proceedings of the European Simulation Multiconference (ESM 2001), Prague, Czech Republic, 6–9 June 2001.

[10] AODV ROUTING PROTOCOL WORKING PROCESS Asma Ahmed, A. Hanan, Izzeldin Osman

LIST OF FIGURES

Figure number	Figure name
Fig1	Flowchart for leveling.
Fig2	Flowchart for tree creation.
Fig 3	Packet Sending Flowchart.
Fig4	Leveling scenario.
Fig5	Tree Topology Setup
Fig6	Screenshot(i)
Fig7	Screenshot (ii)