import java.util.ArrayList;

import java.util.LinkedList;

import java.util.Random;

import java.util.Queue;

import java.util.Date;

public class Node extends Thread

{

//temp sum

public double sum = 0;

public Node()

{

name = "";

idle = true;

waiting = false;

sentRTS = false;

sentCTS = false;

receivedRTS = false;

receivedCTS = false;

backoff = 0.0;

sendAttempts = 0;

//Start time

Date date = new Date();

setTime(date.getTime());

//iniitalize total data amount

totalData = 0;

files = new LinkedList<Double>();

nodesInRange = new ArrayList<Node>();

//Default packets come in 2 per second

arrivalRate = 0.5;

//Set number of files node will send

Random rand = new Random();

numOfFiles = rand.nextInt(10);

this.setx\_Cord(rand.nextInt(200) - 100);

this.sety\_Cord(rand.nextInt(200) - 100);

//Send the file to random size from 1 to 1000

for(int i = 0; i < numOfFiles;i++)

{

files.offer(rand.nextDouble() \* 1000.0);

}

this.setRun(true);

}

public void run()

{

Random r = new Random();

double num\_frames = PACKET\_NUM + r.nextInt(5000);

this.setcurrentFile(0);

if(this.getNodesInRange().size() == 0)

{

System.out.println(this.getNodeName() + " has no node in range");

run = false;

}

while(getRun())

{

for(int j = this.getcurrentFile(); j < num\_frames;j++)

{

if(this.getWaiting() == false)

{

try {

sentRTS(j);

reset();

this.setcurrentFile(j);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

}

if(this.getcurrentFile() >= PACKET\_NUM - 1)

{

double throughput;

run = false;

Date date = new Date();

sum = ((double)date.getTime() - timeElapsed) /1000;

throughput = this.getTotalData()/sum;

System.out.println("DONE " + this.getNodeName() + ", time spend= " +sum + " with a throughput of " + throughput);

}

}

}

public void AddInRangeNode(Node inRange)

{

this.nodesInRange.add(inRange);

}

public boolean isRange(Node receiver)

{

if((Math.pow(receiver.getx\_Cord()-x\_Cord,2) + Math.pow(receiver.gety\_Cord()-y\_Cord,2)) <= RANGE\*RANGE)

{

return true;

}

return false;

}

public double distance(Node receiver)

{

return(Math.sqrt((Math.pow(receiver.getx\_Cord()-x\_Cord,2) + Math.pow(receiver.gety\_Cord()-y\_Cord,2))));

}

/\*\*

\* @param j

\* @throws InterruptedException

\*/

public void sentRTS(int j) throws InterruptedException{

Random rand = new Random();

int receiverIndex = 0;

if(nodesInRange.size() != 0) //if there are nodes in range

{

//Randomly generate a receiver from nodes in Range

receiverIndex = rand.nextInt(nodesInRange.size());

Node temp = this.nodesInRange.get(receiverIndex);

int backOfAttempt = 0;

//Random Packet Size

double packetSize = 1000.0 \* (rand.nextInt((int) PAYLOAD\_MAX\_SIZE) + FRAME\_OVERHEAD\_SIZE);

double distance = distance(temp);

//prop delay

double propDelay = distance/speedOfLight;

sleep(0,(int) (propDelay\*1000000000.0));

for(Node i: this.nodesInRange)

{

if(!i.equals(temp))

{

i.receiveRTSBackOff(propDelay);

}

}

boolean failed = false; //field to see if packet had to back off before

double lastBackOffTime = 0;

while(temp.receiveRTS(this, j, failed, packetSize, lastBackOffTime) == false)

{

failed = true;

backOfAttempt++;

//Calculation for back off in nanoseconds

double BackOffValue = BACKOFF\_VALUE \* 1000000000; double upperbound = Math.pow(2, backOfAttempt); //will not -1 cause might get 1 and get 0

//randomly generate

double BackOffNum = (double)rand.nextInt((int) upperbound);

lastBackOffTime = BackOffValue\*BackOffNum;

//Back off for a certain time

sleep((int)(BackOffValue\*BackOffNum)/1000000,(int) (BackOffValue\*BackOffNum%1000000.0));

if(backOfAttempt > MAX\_BACKOFF\_ATTEMPTS)

{

System.out.println("fail :" + this.getNodeName() + j + " to " + temp.getNodeName() + ", idle status: " + temp.getIdle());

failed = true;

this.setTotalData(this.getTotalData() + packetSize);

this.setcurrentFile(this.getcurrentFile() - 1);

break;

}

}

}

else //if no nodes in range then can't send or receive

{

run = false;

}

}

public void receiveRTSBackOff(double backOffTime) throws InterruptedException

{

setWaiting(true);

sleep((int) (backOffTime\*1000),(int) backOffTime%1000000);

setWaiting(false);

}

public boolean receiveRTS(Node sendingNodeName,int j,boolean i,double packSize,double lastBackTime) throws InterruptedException

{

//if busy or if another node got here first or if node is current receiving

if(getIdle() == false || this.getReceivedCTS() || this.getSentCTS())

{

return false;

}

else

{

setIdle(false);

setReceivedRTS(true);

sendingNodeName.setIdle(false);

sentCTS(sendingNodeName,j,packSize);

//If a node had to backoff but then got through

if(i == true)

{ sendingNodeName.setAverageBackOff((sendingNodeName.getAverageBackOff() + lastBackTime)/2);

}

return true;

}

}

public void sentCTS(Node nodeToCTS,int j,double packetSize) throws InterruptedException

{

this.setSentCTS(true);

//if(idle) System.out.println("ERROR");

for(Node i: nodesInRange){

if(i.equals(nodeToCTS) == true)

{

double distance = (int) distance(nodeToCTS);

//prop delay

double propDelay = distance/speedOfLight;

sleep(0,(int) (propDelay\*1000000000.0));

i.receiveCTS(this ,j,packetSize);

}

else

{

i.setWaiting(true);

}

}

}

public void receiveCTS(Node receiverName,int j,double packetSize) throws InterruptedException{

double distance = distance(receiverName);

//prop delay

double propDelay = distance/speedOfLight;

double transmitDelay = packetSize/BAND\_WIDTH;

double delay = propDelay + transmitDelay;

sum = sum + delay;

//Add frame size to total

this.setTotalData(this.getTotalData() + packetSize);

//System.out.println((int) (delay \* 1000.0));

this.setReceivedCTS(true);

sleep((int) (delay\*1000.0),(int) ((delay)\*1000000000.0)%1000000);

//System.out.println("Packet #" + j + "From " + this.getNodeName() + " Done Packet");

requestAck(receiverName, delay);

}

public void requestAck(Node receivingNode, double delay) throws InterruptedException

{

//receivingNode.reset();

for(Node i:nodesInRange)

{

if(i.getNodeName().equals(receivingNode.getNodeName()))

{

if(i.getAck(receivingNode,delay) == true)

{

//sucessfully sent frame?

}

else

{

//if failed (in this simulation there are never any packets lost for now

//decrement the currentFile

this.setcurrentFile(this.getcurrentFile() - 1);

}

}

}

}

public boolean getAck(Node sendingNodeName, double delay) throws InterruptedException

{

this.reset();

for(Node i: nodesInRange)

{

if(i.equals(sendingNodeName) == false)

{

i.setWaiting(false);

}

}

//received data and return an ACK

double distance = (int) distance(sendingNodeName);

//prop delay

double propDelay = distance/speedOfLight;

sleep(0,(int) (propDelay\*1000000000.0));

return true;

}

public void reset()

{

this.idle = true;

this.waiting = false;

this.sentRTS = false;

this.sentCTS = false;

this.receivedRTS = false;

this.receivedCTS = false;

this.backoff = 0.0;

this.sendAttempts = 0;

}

//Getters Setters

}