Changes in Simulation:

public void feed(Queue<Double> p)

{

while(p.peek()!=null)

{

files.offer(p.poll());

}

}

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;

}

//If async then run packet shooter

if(sync == 0)

{

PacketFeeder p = new PacketFeeder(this);

p.start();

}

while(getRun())

{

if(sync == 1)

{

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

{

if(this.getWaiting() == false)

{

try {

sentRTS();

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);

}

}

else

{

while(files.peek()!=null)

{

if(this.getWaiting() == false)

{

try {

sentRTS();

reset();

this.setcurrentFile(0);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

if(done && files.peek() == null)

{

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 + " total date: " + this.getTotalData() );

}

}

}

}

}

public void sentRTS() 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;

double packetSize = 0;

//Random Packet Size

if(sync == 1)

{

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

}

else

{

packetSize = 1000.0 \* (files.poll() + FRAME\_OVERHEAD\_SIZE);

}

class PacketFeeder extends Thread

{

private static final double AVERAGE\_PER\_SEC = 100.0;

private static final int TIME = 10;

public Queue<Double> p;

public Node Node\_;

public double ctt = 0;

public PacketFeeder(Node node)

{

Node\_ = node;

}

public void run()

{

double actualRate = poisson(AVERAGE\_PER\_SEC);

ctt = ctt + actualRate;

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

{

//give x ammount of packets

p = new LinkedList<Double>();

Random rand = new Random();

for(int j = 0; j < (int) actualRate; j++)

{

p.offer(1/exponential(AVERAGE\_PER\_SEC));

//p.offer(1000.0 \* (rand.nextInt((int) 18) + .2));

}

Node\_.feed(p);

try

{

sleep(1000);

}

catch(Exception e){}

}

Node\_.done = true;

}

private int poisson(double rate)

{

double sum = 0;

int count = 0;

while(sum < 1.0)

{

sum = sum + exponential(rate);

count++;

}

return count - 1;

}

//rate is avg x pkt/sec and 1/exponential = actual packet size

// @return 1/(time until next event aka actual pkt size)

private double exponential(double rate)

{

Random rand = new Random();

double U = rand.nextDouble();

if(U < .00001)

{

U = .00001;

}

return -Math.log(U)/rate;

}

}