Network & Mobility — D0021E
Laboration 1

Institutionen för system- och rymdteknik
Luleå tekniska universitet
971 87 Luleå, Sverige

Richard Hultstrand*
Anton Karmeborg**

February 6, 2018

*E-mail: richul-5@student.ltu.se
**E-mail: antkar-5@student.ltu.se

Contents

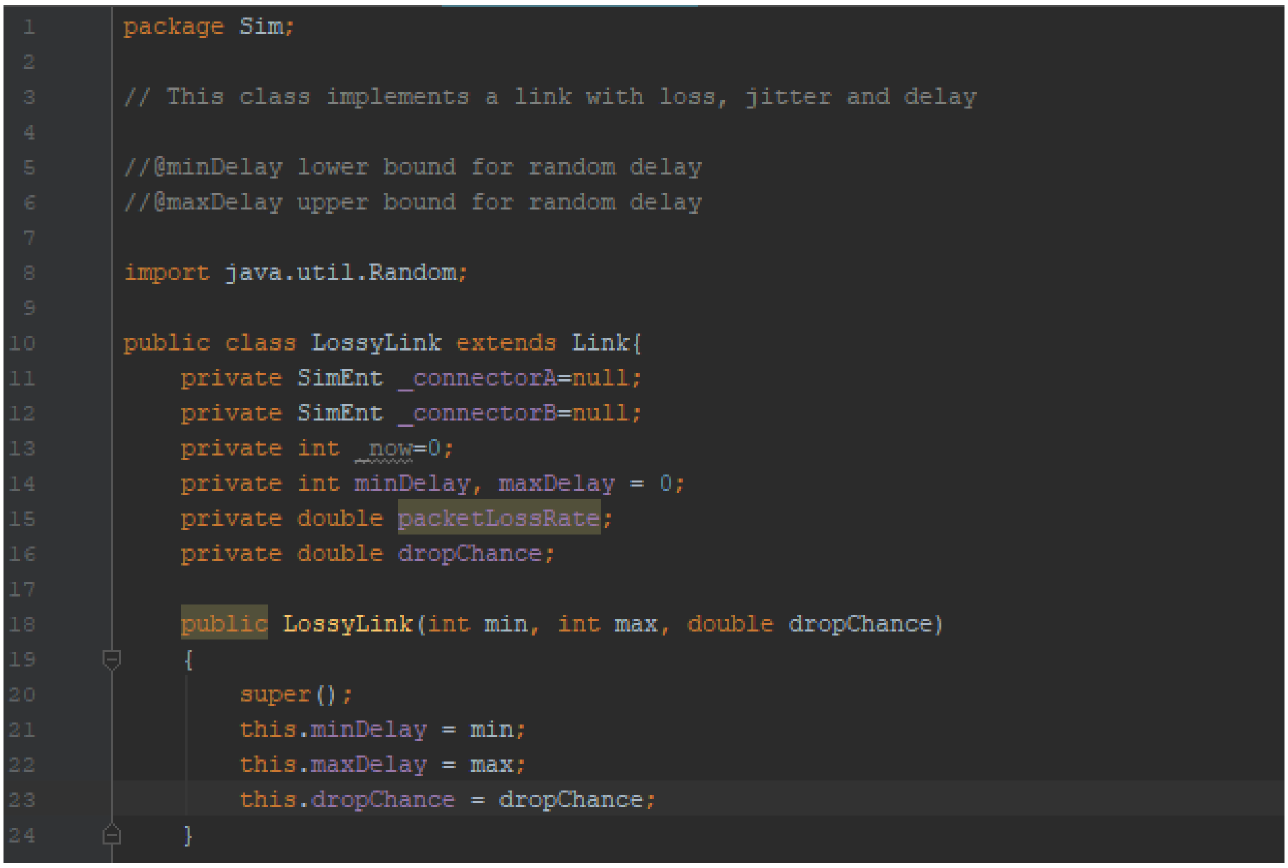
1 Introduction

The objective of this laboration was to get an understanding in how simulators operate and why they are an important tool for modeling complex real world systems. Another goal was also to get an understanding in how normal networks operate by introducing terms such as network delay, jitter and packet loss.

2 Methods

For this laboration a simple simulator with basic yet vital methods was provided. The simulator is written in Java and it is in this environment that the lab was carried out in. The first step was to familiarize with the simulator. This was done by simply reading what each method was used for and running the simulator in order to see its functioning state. Next knowledge was needed in how delay, jitter and packet-loss was calculated and inferred into a network.

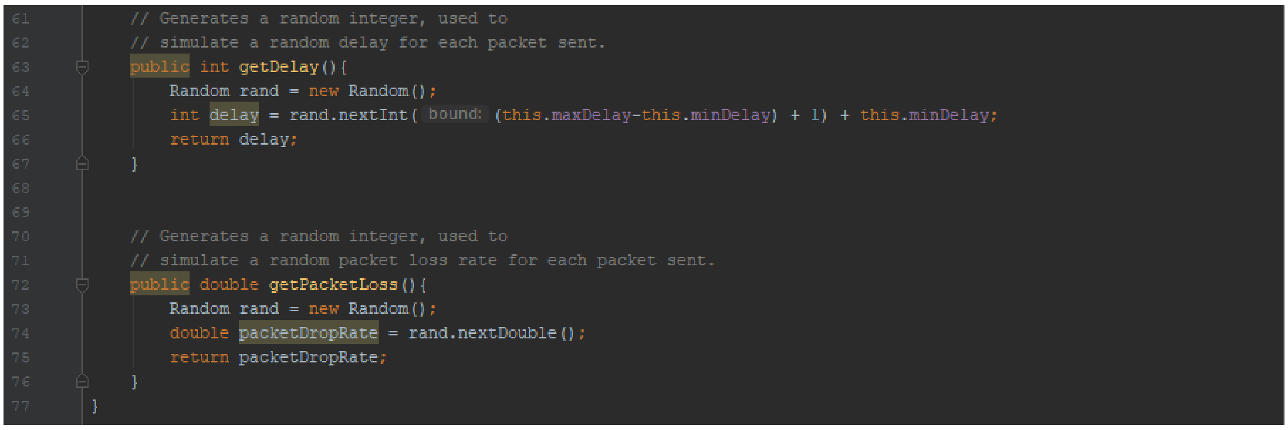
To implement a LossyLink, a class which extends Link was created. The difference being that the LossyLink takes a few more parameters into its constructor, namely a minimum and maximum delay aswell as a packet-drop chance.



```java
package Sim;

// This class implements a link with loss, jitter and delay

//@minDelay lower bound for random delay
//@maxDelay upper bound for random delay

import java.util.Random;

public class LossyLink extends Link{
    private SimEnt _connectorA=null;
    private SimEnt _connectorB=null;
    private int _now=0;
    private int minDelay, maxDelay = 0;
    private double packetLossRate;
    private double dropChance;

    public LossyLink(int min, int max, double dropChance)
    {
        super();
        this.minDelay = min;
        this.maxDelay = max;
        this.dropChance = dropChance;
    }
```

Figure 1: Constructor for LossyLink.java

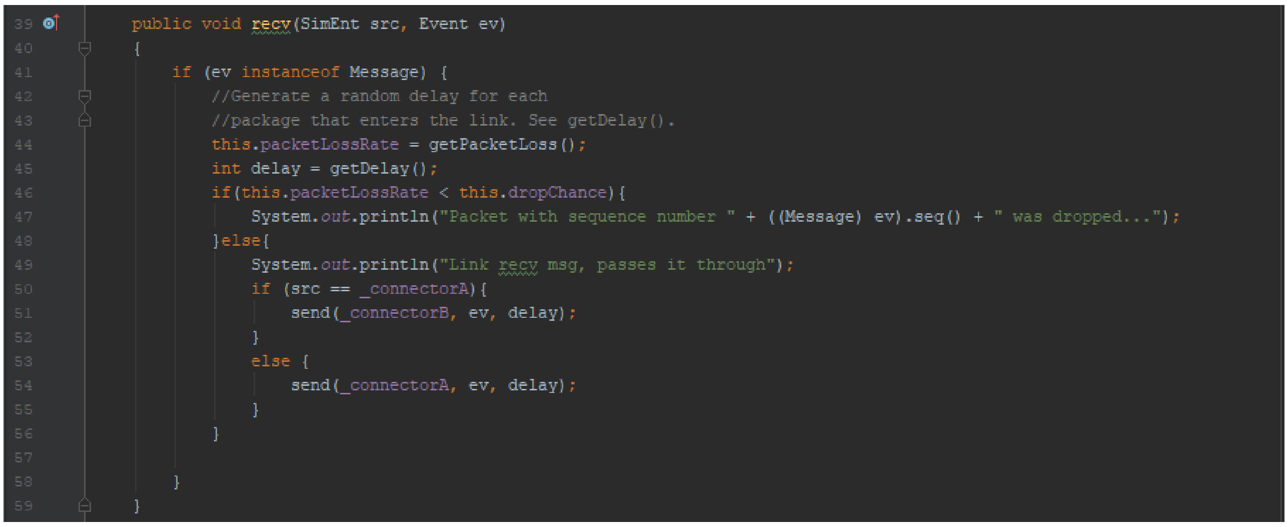
The values for the delay are used in the method getDelay() in order to produce a random integer between minDelay and maxDelay to produce a delay to which the packets are transferred with, see figure 2.



```java
    // Generates a random integer, used to
    // simulate a random delay for each packet sent.
    public int getDelay(){
        Random rand = new Random();
        int delay = rand.nextInt( bound: (this.maxDelay-this.minDelay) + 1) + this.minDelay;
        return delay;
    }


    // Generates a random integer, used to
    // simulate a random packet loss rate for each packet sent.
    public double getPacketLoss(){
        Random rand = new Random();
        double packetDropRate = rand.nextDouble();
        return packetDropRate;
    }
}
```

Figure 2: Delay and Packet-loss functions in LossyLink.java

As seen in figure 2 the method getPacketLoss() generates a random double, which for those who do not know programming is a so called floating-point number. This double is used to determine how high chance the LossyLink has to drop a packet. See figure 3.



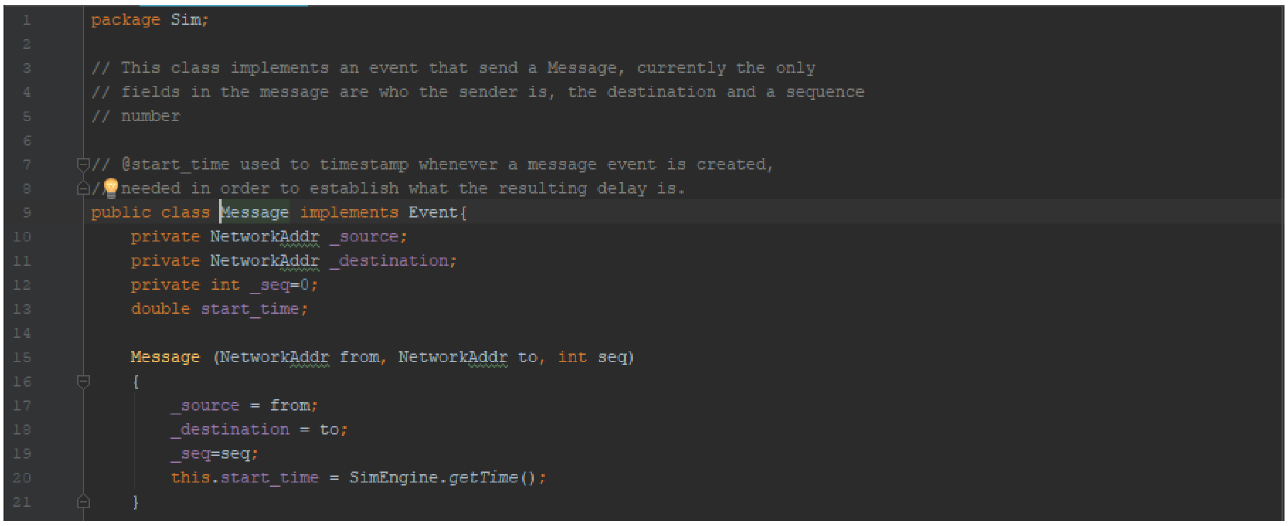
```java
    public void recv(SimEnt src, Event ev)
    {
        if (ev instanceof Message) {
            //Generate a random delay for each
            //package that enters the link. See getDelay().
            this.packetLossRate = getPacketLoss();
            int delay = getDelay();
            if(this.packetLossRate < this.dropChance){
                System.out.println("Packet with sequence number " + ((Message) ev).seq() + " was dropped...");
            }else{
                System.out.println("Link recv msg, passes it through");
                if (src == _connectorA){
                    send(_connectorB, ev, delay);
                }
                else {
                    send(_connectorA, ev, delay);
                }
            }

        }
    }
```

Figure 3: Recv function for LossyLink.java

The recv() function in LossyLink.java is a modified version of the recv() function in Link.java. The changes made are used in order to infer a delay and introduce a chance of packet-loss. The delay is generated for each new packet that needs to be sent, this delay is then placed in the send() function-call.

In order to simulate and to be able to slightly control the drop rate, the double that is sent to the constructor of the LossyLink (see figure 1) is used to compare with a random double generated for each packet. So for example if the double 0.8 is entered into the constructor, theoretically the LossyLink will have an 80% chance to drop a packet. If the random double generated in getPacketLoss() (see figure 2) is higher than the double sent in to the LossyLink through the constructor then a message will be printed that a package was dropped, and the simulation will go on as normal, without sending that packet further.

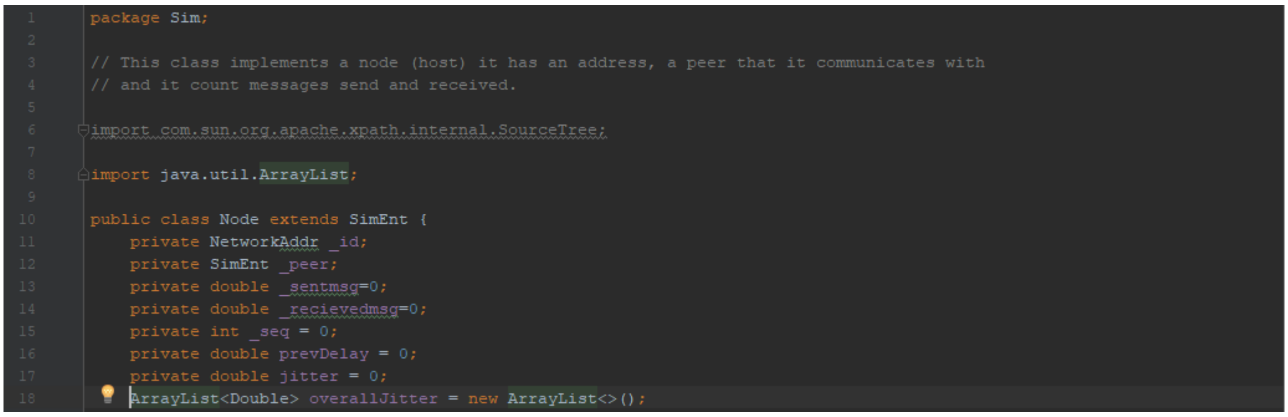
To be able to calculate the delay of packages that are being sent at different times there needs to be a time-stamp set at exactly the time in which this package was created. Since a new Message.java object is created for each individual package, this is a good place to set such a time-stamp. This because each time-stamp will be individual for each package and will be easily recovered.



```java
package Sim;

// This class implements an event that send a Message, currently the only
// fields in the message are who the sender is, the destination and a sequence
// number

// @start_time used to timestamp whenever a message event is created,
// needed in order to establish what the resulting delay is.
public class Message implements Event{
    private NetworkAddr _source;
    private NetworkAddr _destination;
    private int _seq=0;
    double start_time;

    Message (NetworkAddr from, NetworkAddr to, int seq)
    {
        _source = from;
        _destination = to;
        _seq=seq;
        this.start_time = SimEngine.getTime();
    }
```

Figure 4: Message.java time-stamp

As seen in figure 4, the SimEngine.java function getTime() is used to set a specific time for which this object was created. This is done by setting a global variable in the constructor of Message.java to hold the time in the simulation at which this object was created.

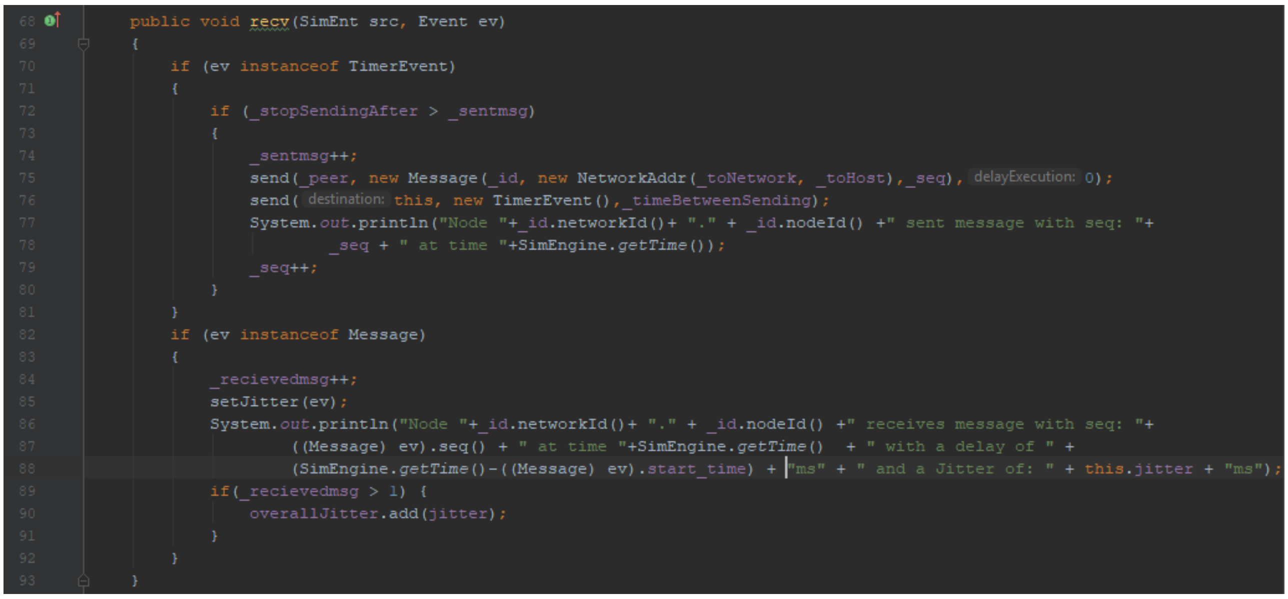


```java
package Sim;

// This class implements a node (host) it has an address, a peer that it communicates with
// and it count messages send and received.

import com.sun.org.apache.xpath.internal.SourceTree;

import java.util.ArrayList;

public class Node extends SimEnt {
    private NetworkAddr _id;
    private SimEnt _peer;
    private double _sentmsg=0;
    private double _recievedmsg=0;
    private int _seq = 0;
    private double prevDelay = 0;
    private double jitter = 0;
    ArrayList<Double> overallJitter = new ArrayList<>();
```

Figure 5: Global variables added to Node.java

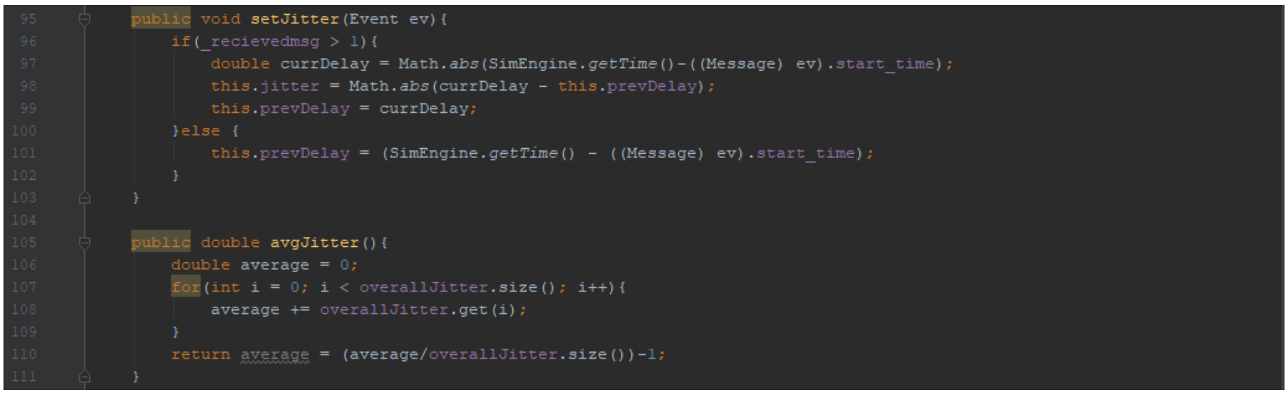
Figure 5 shows a few variables that have been added to the simulation. These variables are _receivedmsg, prevDelay, jitter and overallJitter. These variables will be explained, starting with those pertaining to calculation of jitter between two packets.

As explained in section 1 of this report, jitter is calculated by taking the difference in the delay between the previous and the current package received.



```java
    public void recv(SimEnt src, Event ev)
    {
        if (ev instanceof TimerEvent)
        {
            if (_stopSendingAfter > _sentmsg)
            {
                _sentmsg++;
                send(_peer, new Message(_id, new NetworkAddr(_toNetwork, _toHost),_seq), delayExecution: 0);
                send( destination: this, new TimerEvent(),_timeBetweenSending);
                System.out.println("Node "+_id.networkId()+ "." + _id.nodeId() +" sent message with seq: "+
                        _seq + " at time "+SimEngine.getTime());
                _seq++;
            }
        }
        if (ev instanceof Message)
        {
            _recievedmsg++;
            setJitter(ev);
            System.out.println("Node "+_id.networkId()+ "." + _id.nodeId() +" receives message with seq: "+
                    ((Message) ev).seq() + " at time "+SimEngine.getTime()  + " with a delay of " +
                    (SimEngine.getTime()-((Message) ev).start_time) + "ms" + " and a Jitter of: " + this.jitter + "ms");
            if(_recievedmsg > 1) {
                overallJitter.add(jitter);
            }
        }
    }
```

Figure 6: Modified recv() function in Node.java



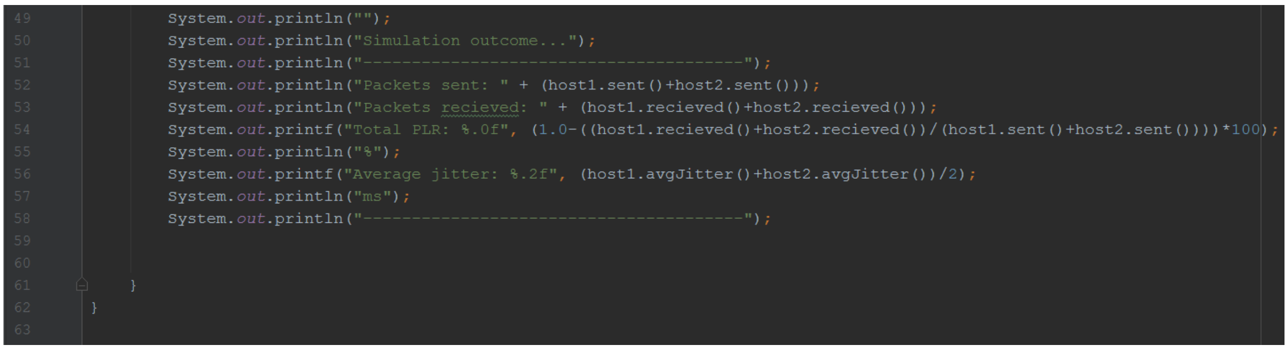
```java
    public void setJitter(Event ev){
        if(_recievedmsg > 1){
            double currDelay = Math.abs(SimEngine.getTime()-((Message) ev).start_time);
            this.jitter = Math.abs(currDelay - this.prevDelay);
            this.prevDelay = currDelay;
        }else {
            this.prevDelay = (SimEngine.getTime() - ((Message) ev).start_time);
        }
    }

    public double avgJitter(){
        double average = 0;
        for(int i = 0; i < overallJitter.size(); i++){
            average += overallJitter.get(i);
        }
        return average = (average/overallJitter.size())-1;
    }
```

Figure 7: Function in Node.java to calculate jitter and average jitter

Changes were made to the recv() function in Node.java (see figure 6) in order to calculate the jitter between two packages and also to add that jitter to the arraylist seen in figure 5. If the recv function gets an event of type message, the _receivedmsg variable is incremented by one and the setJitter() function is called by sending the incoming event to it. The setJitter() function seen in figure 7 then checks if this is the first message it receives. In this case it merely calculates the delay of which this package was delivered and then does nothing else. If it was not the first package received by this node, the setJitter() function first calculates the delay of the package that was just delivered and then calculates the jitter by performing an absolute calculation of the current packages delay minus the previous packages delay. Then finally the function sets the prevDelay variable to the value of the current packages delay.

The recv() function then prints out a message containing information about the received package, such as the delay and jitter it had. Finally this function checks once again if this was the first package that was received at this node, if this was not the first package received, then the jitter is added to an arraylist.

Finally the Run.java class was modified in order to display the final results of the simulation (see figure 8).



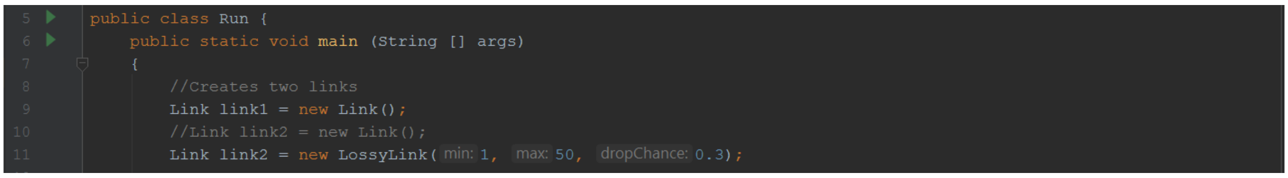
```java
        System.out.println("");
        System.out.println("Simulation outcome...");
        System.out.println("---------------------------------------");
        System.out.println("Packets sent: " + (host1.sent()+host2.sent()));
        System.out.println("Packets recieved: " + (host1.recieved()+host2.recieved()));
        System.out.printf("Total PLR: %.0f", (1.0-((host1.recieved()+host2.recieved())/(host1.sent()+host2.sent())))*100);
        System.out.println("%");
        System.out.printf("Average jitter: %.2f", (host1.avgJitter()+host2.avgJitter())/2);
        System.out.println("ms");
        System.out.println("---------------------------------------");


    }
}
```

Figure 8: Print simulation results.

After the thread running the simulation has finished its computing, a number of print statements will be run. These print statements calculate and display the amount of packets received and sent, the total packet-loss ratio and the average jitter of this simulation.

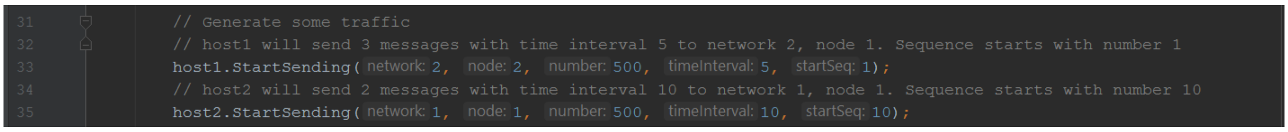
3 Results

The following figures display the results when running the simulation for 1000 packets with one LossyLink and of normal Link. The links send 500 packets each. The delay is set between 1 and 50ms and the LossyLink has a 30% chance to drop a packet (see figures 9 and 10).



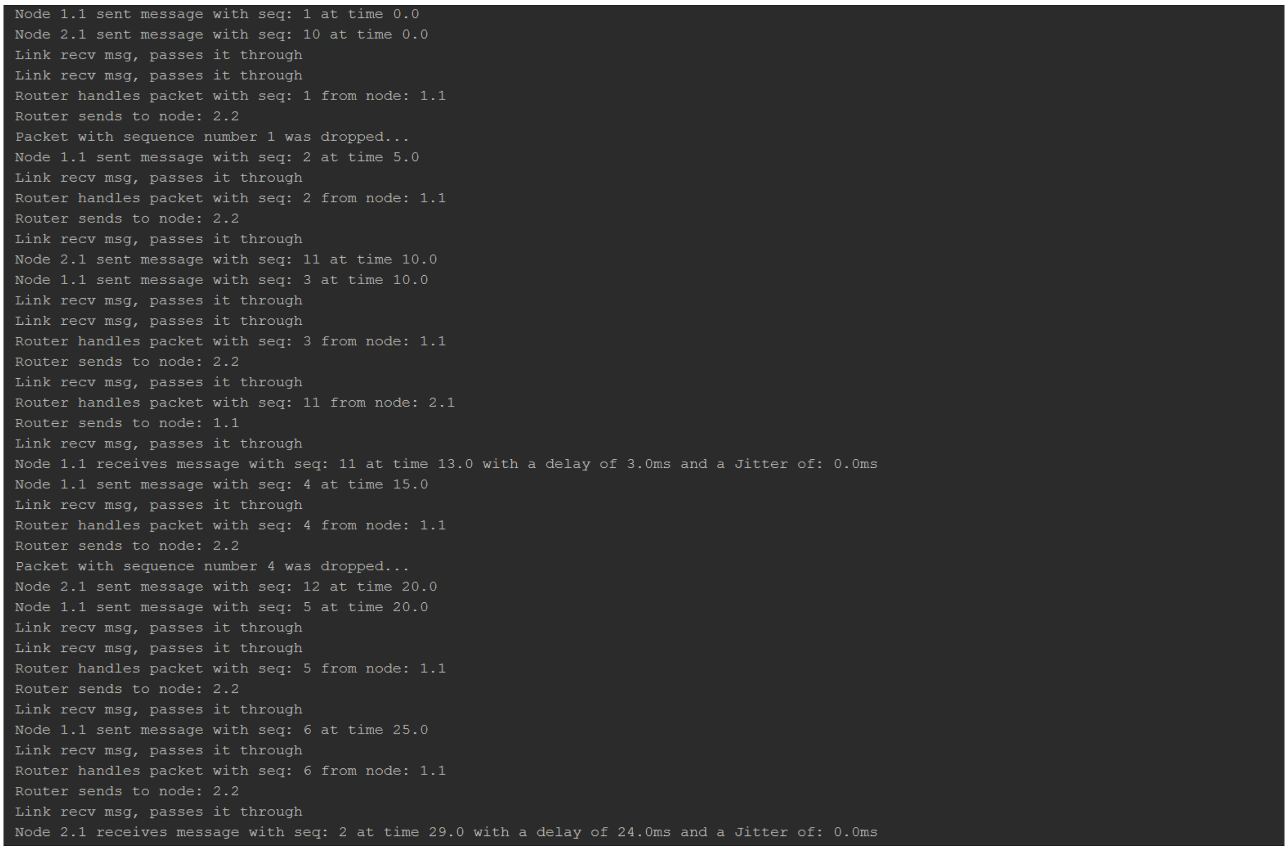
```
public class Run {
    public static void main (String [] args)
    {
        //Creates two links
        Link link1 = new Link();
        //Link link2 = new Link();
        Link link2 = new LossyLink( min: 1,  max: 50,  dropChance: 0.3);
```

Figure 9: Link configurations.



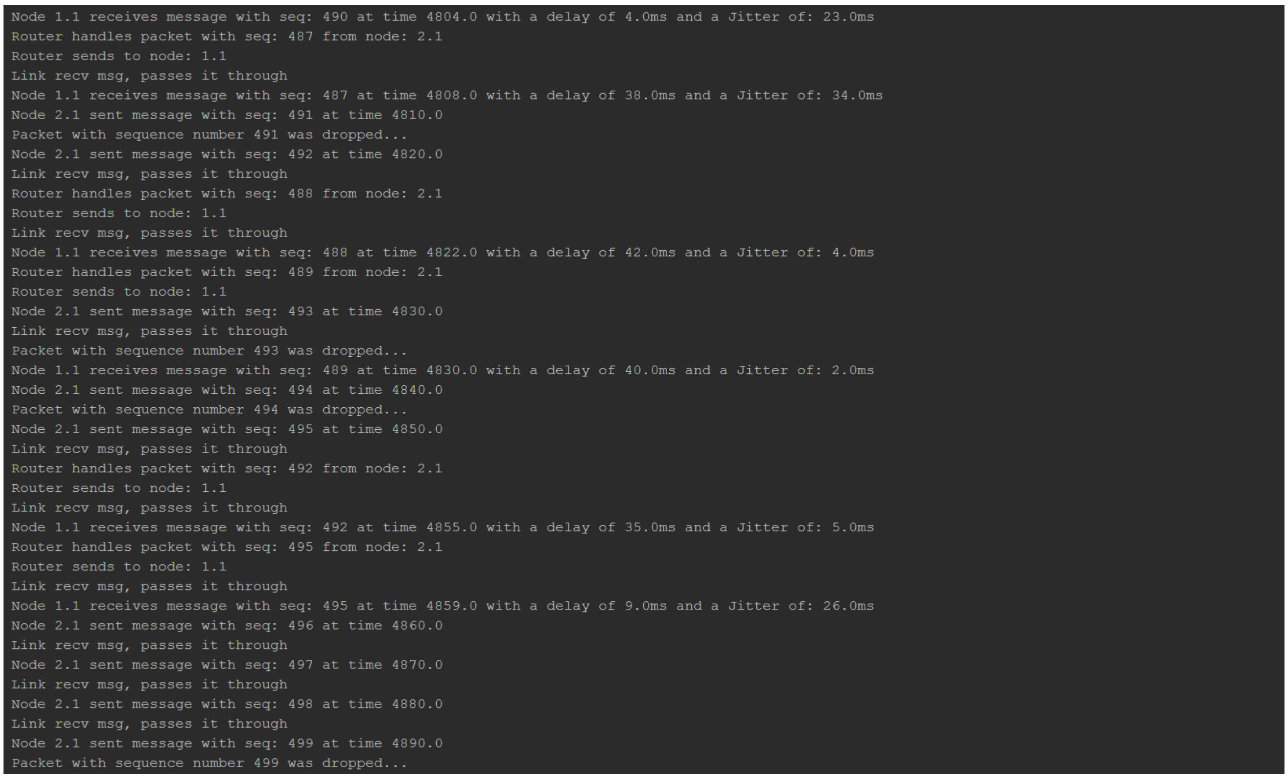
```
        // Generate some traffic
        // host1 will send 3 messages with time interval 5 to network 2, node 1. Sequence starts with number 1
        host1.StartSending( network: 2,  node: 2,  number: 500,  timeInterval: 5,  startSeq: 1);
        // host2 will send 2 messages with time interval 10 to network 1, node 1. Sequence starts with number 10
        host2.StartSending( network: 1,  node: 1,  number: 500,  timeInterval: 10,  startSeq: 10);
```

Figure 10: Node configurations.



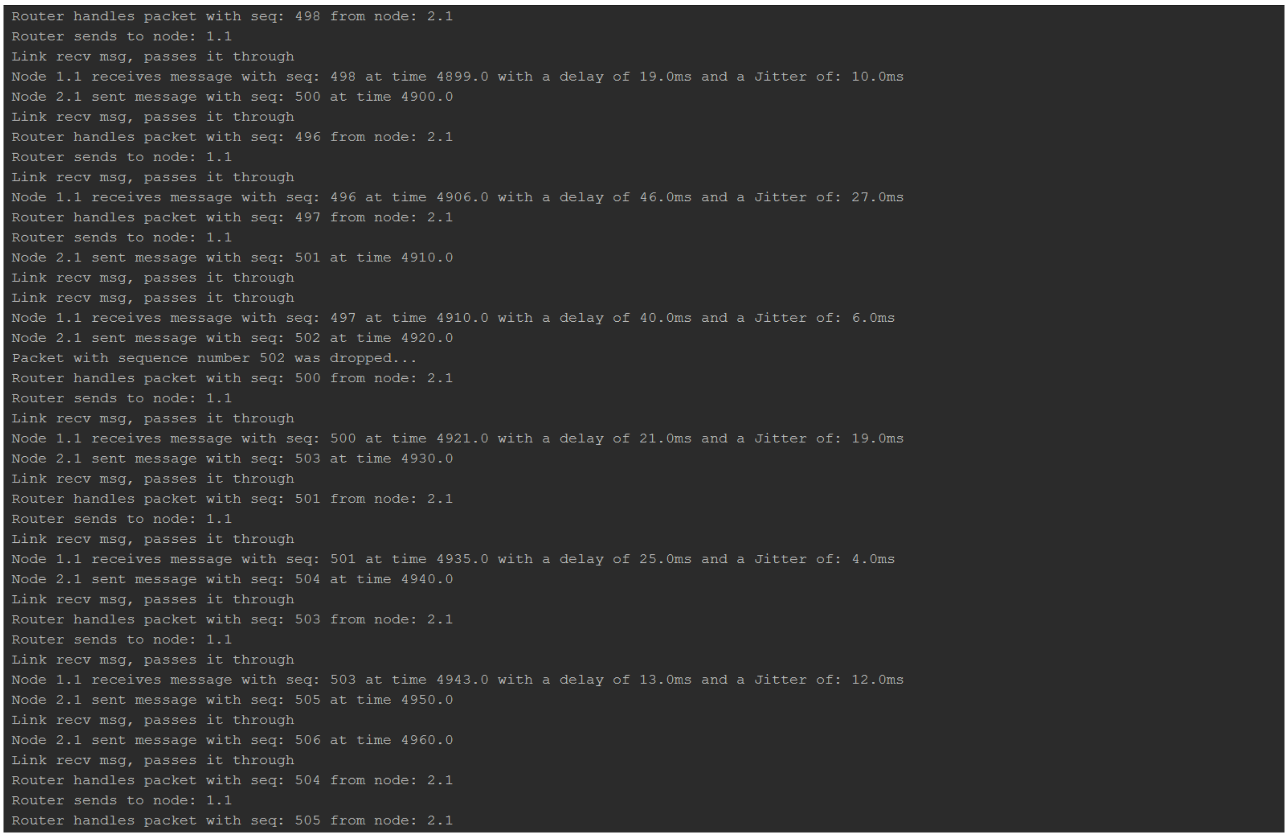
```
Node 1.1 sent message with seq: 1 at time 0.0
Node 2.1 sent message with seq: 10 at time 0.0
Link recv msg, passes it through
Link recv msg, passes it through
Router handles packet with seq: 1 from node: 1.1
Router sends to node: 2.2
Packet with sequence number 1 was dropped...
Node 1.1 sent message with seq: 2 at time 5.0
Link recv msg, passes it through
Router handles packet with seq: 2 from node: 1.1
Router sends to node: 2.2
Link recv msg, passes it through
Node 2.1 sent message with seq: 11 at time 10.0
Node 1.1 sent message with seq: 3 at time 10.0
Link recv msg, passes it through
Link recv msg, passes it through
Router handles packet with seq: 3 from node: 1.1
Router sends to node: 2.2
Link recv msg, passes it through
Router handles packet with seq: 11 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 11 at time 13.0 with a delay of 3.0ms and a Jitter of: 0.0ms
Node 1.1 sent message with seq: 4 at time 15.0
Link recv msg, passes it through
Router handles packet with seq: 4 from node: 1.1
Router sends to node: 2.2
Packet with sequence number 4 was dropped...
Node 2.1 sent message with seq: 12 at time 20.0
Node 1.1 sent message with seq: 5 at time 20.0
Link recv msg, passes it through
Link recv msg, passes it through
Router handles packet with seq: 5 from node: 1.1
Router sends to node: 2.2
Link recv msg, passes it through
Node 1.1 sent message with seq: 6 at time 25.0
Link recv msg, passes it through
Router handles packet with seq: 6 from node: 1.1
Router sends to node: 2.2
Link recv msg, passes it through
Node 2.1 receives message with seq: 2 at time 29.0 with a delay of 24.0ms and a Jitter of: 0.0ms
```

Figure 11: Simulation start.



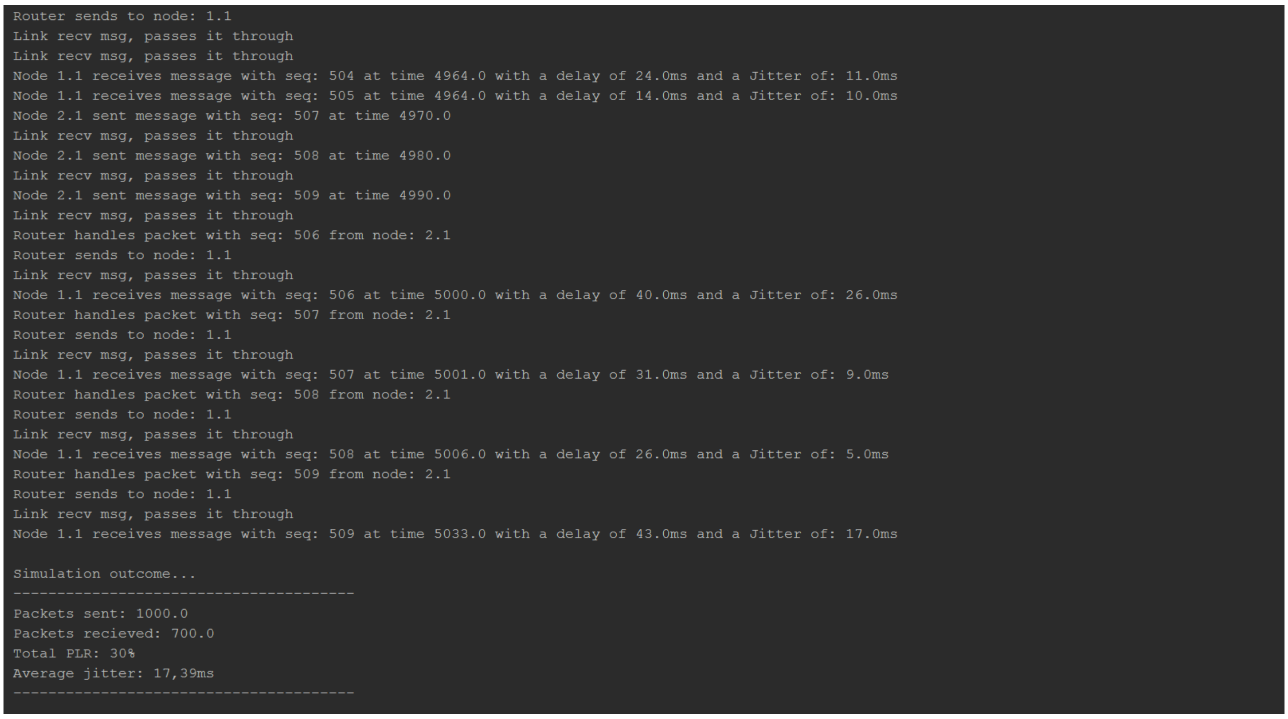
```
Node 1.1 receives message with seq: 490 at time 4804.0 with a delay of 4.0ms and a Jitter of: 23.0ms
Router handles packet with seq: 487 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 487 at time 4808.0 with a delay of 38.0ms and a Jitter of: 34.0ms
Node 2.1 sent message with seq: 491 at time 4810.0
Packet with sequence number 491 was dropped...
Node 2.1 sent message with seq: 492 at time 4820.0
Link recv msg, passes it through
Router handles packet with seq: 488 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 488 at time 4822.0 with a delay of 42.0ms and a Jitter of: 4.0ms
Router handles packet with seq: 489 from node: 2.1
Router sends to node: 1.1
Node 2.1 sent message with seq: 493 at time 4830.0
Link recv msg, passes it through
Packet with sequence number 493 was dropped...
Node 1.1 receives message with seq: 489 at time 4830.0 with a delay of 40.0ms and a Jitter of: 2.0ms
Node 2.1 sent message with seq: 494 at time 4840.0
Packet with sequence number 494 was dropped...
Node 2.1 sent message with seq: 495 at time 4850.0
Link recv msg, passes it through
Router handles packet with seq: 492 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 492 at time 4855.0 with a delay of 35.0ms and a Jitter of: 5.0ms
Router handles packet with seq: 495 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 495 at time 4859.0 with a delay of 9.0ms and a Jitter of: 26.0ms
Node 2.1 sent message with seq: 496 at time 4860.0
Link recv msg, passes it through
Node 2.1 sent message with seq: 497 at time 4870.0
Link recv msg, passes it through
Node 2.1 sent message with seq: 498 at time 4880.0
Link recv msg, passes it through
Node 2.1 sent message with seq: 499 at time 4890.0
Packet with sequence number 499 was dropped...
```

Figure 12: Simulation running.



```
Router handles packet with seq: 498 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 498 at time 4899.0 with a delay of 19.0ms and a Jitter of: 10.0ms
Node 2.1 sent message with seq: 500 at time 4900.0
Link recv msg, passes it through
Router handles packet with seq: 496 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 496 at time 4906.0 with a delay of 46.0ms and a Jitter of: 27.0ms
Router handles packet with seq: 497 from node: 2.1
Router sends to node: 1.1
Node 2.1 sent message with seq: 501 at time 4910.0
Link recv msg, passes it through
Link recv msg, passes it through
Node 1.1 receives message with seq: 497 at time 4910.0 with a delay of 40.0ms and a Jitter of: 6.0ms
Node 2.1 sent message with seq: 502 at time 4920.0
Packet with sequence number 502 was dropped...
Router handles packet with seq: 500 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 500 at time 4921.0 with a delay of 21.0ms and a Jitter of: 19.0ms
Node 2.1 sent message with seq: 503 at time 4930.0
Link recv msg, passes it through
Router handles packet with seq: 501 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 501 at time 4935.0 with a delay of 25.0ms and a Jitter of: 4.0ms
Node 2.1 sent message with seq: 504 at time 4940.0
Link recv msg, passes it through
Router handles packet with seq: 503 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 503 at time 4943.0 with a delay of 13.0ms and a Jitter of: 12.0ms
Node 2.1 sent message with seq: 505 at time 4950.0
Link recv msg, passes it through
Node 2.1 sent message with seq: 506 at time 4960.0
Link recv msg, passes it through
Router handles packet with seq: 504 from node: 2.1
Router sends to node: 1.1
Router handles packet with seq: 505 from node: 2.1
```

Figure 13: Simulation running.



```
Router sends to node: 1.1
Link recv msg, passes it through
Link recv msg, passes it through
Node 1.1 receives message with seq: 504 at time 4964.0 with a delay of 24.0ms and a Jitter of: 11.0ms
Node 1.1 receives message with seq: 505 at time 4964.0 with a delay of 14.0ms and a Jitter of: 10.0ms
Node 2.1 sent message with seq: 507 at time 4970.0
Link recv msg, passes it through
Node 2.1 sent message with seq: 508 at time 4980.0
Link recv msg, passes it through
Node 2.1 sent message with seq: 509 at time 4990.0
Link recv msg, passes it through
Router handles packet with seq: 506 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 506 at time 5000.0 with a delay of 40.0ms and a Jitter of: 26.0ms
Router handles packet with seq: 507 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 507 at time 5001.0 with a delay of 31.0ms and a Jitter of: 9.0ms
Router handles packet with seq: 508 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 508 at time 5006.0 with a delay of 26.0ms and a Jitter of: 5.0ms
Router handles packet with seq: 509 from node: 2.1
Router sends to node: 1.1
Link recv msg, passes it through
Node 1.1 receives message with seq: 509 at time 5033.0 with a delay of 43.0ms and a Jitter of: 17.0ms

Simulation outcome...
---------------------------------------
Packets sent: 1000.0
Packets recieved: 700.0
Total PLR: 30%
Average jitter: 17,39ms
---------------------------------------
```

Figure 14: End of simulation.

Figure 14 illustrates that as expected, the simulation dropped 300 packets, which amounts to a 30% packet-loss ratio The average jitter for the whole simulation was calculated to 17.39ms and the jitter between packets can be seen in figures 11 – 14.

4 Discussion

When creating this simulation we encountered some questions about the jitter calculation, we knew that the jitter is calculated by the difference in delay between a packet sent and the next packet sent. However the lab specification did not resolve it clearly to us whether we should calculate a delay for every hop a packet does, or the total delay directly for when the packet has arrived. After consulting with our teachers we were able to resolve this problem. We think that the lab specification was a bit short and not fully explained, but this also meant that we had to understand the concepts of what we were implementing better, thus giving us a more comprehensive knowledge of this. We think that the lab overall was a good introduction on how packets work their way through a network and it gave a good understanding of the delay, jitter and probability.

References

[1] The PHP Group, PHP Manual.
The PHP Group. http://php.net/manual/en/index.php 2017-12-26
Accessed: 2017-12-29