LULEÅ UNIVERITY OF TECHNOLOGY

COMPUTER SCIENCE

D0021E

INTRODUCTION TO SIMULATOR AND MODELING NETWORK IMPAIRMENTS

Author
S. JONSSON
setjon-7@student.ltu.se
M. LARSSON ANDERSSON
magany-7@student.ltu.se
M. JONSSON
micjon-5@student.ltu.se

Supervisor
A. Prof. KARAN MITRA
SAGUNA SAGUNA

March 20, 2020

1 Introduction

The objective of the first laboration in this course was to understand a simulator given to us and perform simple simulations.

We also learned to implement jitter, delay and packet loss to get a deeper understanding of the concepts.

2 Methods

The laboration was done in a simulator written in java provided to us by the course administrators. The simulator in question provides basic simulation of end-to-end host communication.

```java
public class LossyLink  extends Link{
  public static int dropPackets = 0;
  private double dropChance;
  private int delay;
  private int jitter;
  private Random random;
  /**
   * Constructor of a link that have a chance to drop packets and induces a delay with variation
       (jitter)
   * @param delay static value a packet sent on link is delayed by
   * @param jitter a dynamic value the packet is additionally delayed by, 0 to given value
   * @param dropChance chance the link has to drop a packet
   */
  public LossyLink(int delay, int jitter, double dropChance) {
    this.delay = delay;
    this.dropChance = dropChance;
    this.jitter = jitter;
    this.random = new Random();

  }
  public ArrayList<Integer> jitterAList = new ArrayList();
  public ArrayList<Integer> jitterBList = new ArrayList();
  public int numberOfPacketsSentByA = 0;
  public int numberOfPacketsSentByB = 0;
  public int jitterA;
  public int lastDelayA;
  public int jitterB;
  private int lastDelayB;
  @Override
  public void recv(SimEnt src, Event ev)
  {
    if (ev instanceof Message) {
      if(random.nextDouble() > dropChance) {/// checks if link should drop packet
        int nextDelay = random.nextInt(jitter+1) + delay; /// calculating how long the packet
            should delay by
        System.out.println("Link recv msg, passes it through, delayed " + nextDelay);
        if (src == _connectorA) {
          ///for analysis
          numberOfPacketsSentByA++;
          int jitter = Math.abs(nextDelay-lastDelayA);///actual jitter
          jitterAList.add(jitter);
          jitterA += jitter;
          lastDelayA = nextDelay;
          ///
          send(_connectorB, ev, nextDelay);
        }
        else
        {
          ///for analysis
          numberOfPacketsSentByB++;
          int jitter = Math.abs(nextDelay-lastDelayB);///actual jitter
          jitterBList.add(jitter);
          jitterB += jitter;
          lastDelayB = nextDelay;
          ///
          send(_connectorA, ev, nextDelay);
        }
        }
      else {
        System.out.println("Bad link dropped packet with seq " + ((Message) ev).seq());
        dropPackets++;
      }

    }

  }
}
```

Figure 1: lossylink

3 Results

In the laboration we used a large packet loss rate at 25% and chose a small jitter of 10ms. Fig. 2 input data 25% packet loss. Fig. 3 input data 10ms jitter.

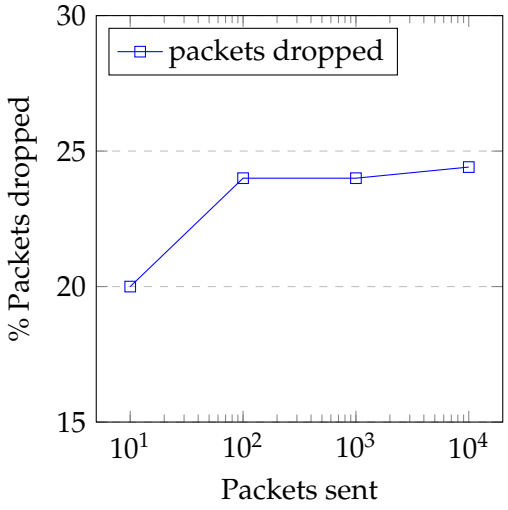

Figure 2: Packet Loss

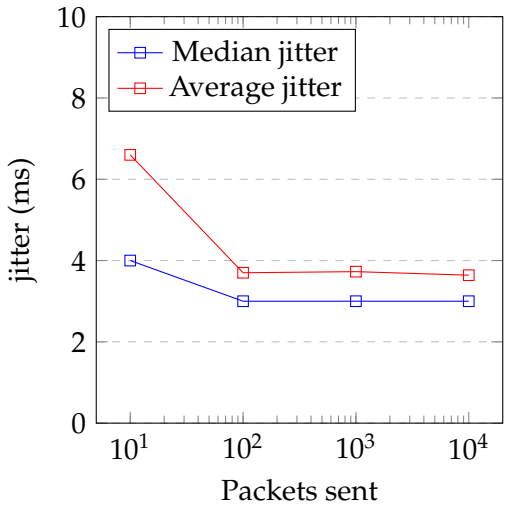

Figure 3: Jitter

4 Discussion

In graph ploting jitter with our given value of jitter 10ms it seems to stablize around 3.5ms with our testing method. Note that when testing with our given value of 10ms it actually adds between 0-10ms to delay and not +-10ms to the delay. So our value seems to be where it should considering how much the difference in delay could be between two delays. However using +-10ms would have given a more easy to understand result.

When drawing conclusions from the graphs (Fig. 2 & Fig. 3) You can see that using a small sample size can lead to inaccurate readings.