LULEÅ UNIVERITY OF TECHNOLOGY

COMPUTER SCIENCE

D0021E

IMPLEMENTING TRAFFIC GENERATORS

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

In the second lab of this course we learned about three different traffic models, constant bit rate (CBR), Gaussian traffic distribution and Poisson distribution which included learning about calculating the possibilities to send a package for Normal and Poisson.

2 Methods

The simulator given to us was implemented with three new classes for CBR, Gaussian traffic, and Poisson for generating the traffic. We also implemented a Sink class to receive the generated inputs from the different classes.

```
1  public class CBR extends Node {
2
3    //redacted variable declarations for shorter code block
4
5    public CBR (int network, int node)
6    {
7      super(network, node);
8      _id = new NetworkAddr(network, node);
9    }
10
11   public void StartSending(int network, int node, int pkgPerSecond, int timeToSend)
12   {
13     _toNetwork = network;
14     _toHost = node;
15     _pkgPerSecond = pkgPerSecond;
16     _timeToSend = timeToSend;
17     send(this, new TimerEvent(), 0);
18   }
19
20   public void recv(SimEnt src, Event ev)
21   {
22     if (ev instanceof TimerEvent)
23     {
24       if (SimEngine.getTime() < _timeToSend)
25       {
26
27         for (int i = 0; i < _pkgPerSecond; i++)
28         {
29           send(_peer, new Message(_id, new NetworkAddr(_toNetwork, _toHost),_seq),0);
30
31           _sentmsg++;
32           _seq++;
33         }
34       }
35       send(this, new TimerEvent(),1);
36     }
37   }
38 }
```

Figure 1: CBR

```java
public class Gaussian extends Node {

  //redacted variable declarations for shorter code block

    public Gaussian (int network, int node) {
        super(node, node);
        _id = new NetworkAddr(network, node);
    }

    // Modified to only take a time limit, deviation and mean.
    public void StartSending(int network, int node, int stdDeviation, int mean, int packages)
    {
        _toNetwork = network;
        _stdDeviation = stdDeviation;
        _mean = mean;
        _toHost = node;
        _seq = 1;

        _packages = packages;
        send(this, new TimerEvent(), 0);
        System.out.println("Sending signal to start sending...");

    }

    // Override: Modified to send packages as a normal distribution with a random gaussian number.
    public void recv(SimEnt src, Event ev)
    {
        if (ev instanceof TimerEvent)
        {
          if(_sentmsg > _packages)
            return;

          double x = random.nextGaussian() * _stdDeviation + _mean;

          send(_peer, new Message(_id, new NetworkAddr(_toNetwork, _toHost), _seq), x);

          _sentmsg++;
            _seq++;

            send(this, new TimerEvent(), x);
        }
    }
}
```

Figure 2: Gaussian

```
public class Poisson extends Node {

    //redacted variable declarations for shorter code block

    public Poisson (int network, int node) {
        super(node, node);
        _id = new NetworkAddr(network, node);
    }

    public void StartSending(int network, int node, int mean, int packages)
    {
        _toNetwork = network;
        _lambda = Math.exp(-mean);
        _toHost = node;
        _seq = 1;
        _packages = packages;

        send(this, new TimerEvent(), 0);
        System.out.println("Sending signal to start sending...");
    }

    //returns next time in milliseconds to send a message
    public int nextTime()
    {
      int k = 0;

      for(double p = 1.0; p > _lambda; k++)
      {
        p *= random.nextDouble();
      }

      return k - 1;
    }

    // Override: Modified to send packages with a poisson distributed delay
    public void recv(SimEnt src, Event ev)
    {
        if (ev instanceof TimerEvent)
        {
          if(_sentmsg > _packages)
            return;

          double next = nextTime();
          double nextSeconds = (next / 100.0);

          _sentmsg++;
          send(_peer, new Message(_id, new NetworkAddr(_toNetwork, _toHost), _seq), nextSeconds);
            _seq++;

            send(this, new TimerEvent(), nextSeconds);
        }
    }
}
```

Figure 3: Poisson

3 Results

In all our tests we chose to send 1000 packets per distribution model where the graphs show Δtime for Gaussian and Poisson and normal time for CBR.

A graph depicting the total number of packages sent over a set time with constant bit rate.

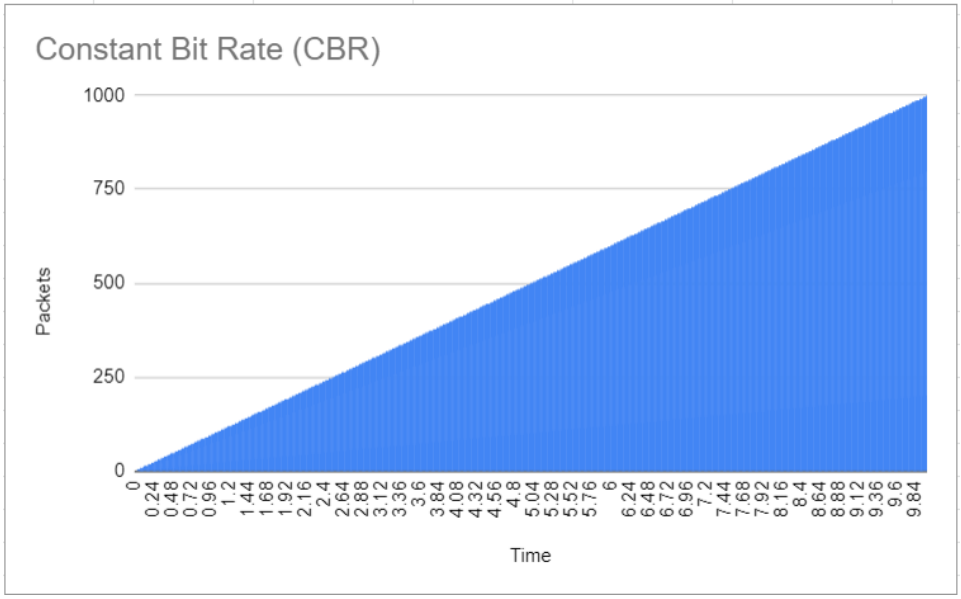

Figure 4: *CBR distribution sending 100 packets per second for 10 seconds*

A histogram showing how the delay in milliseconds between messages follows a gaussian distribution with varying deviations ranging from 3 up to 10.

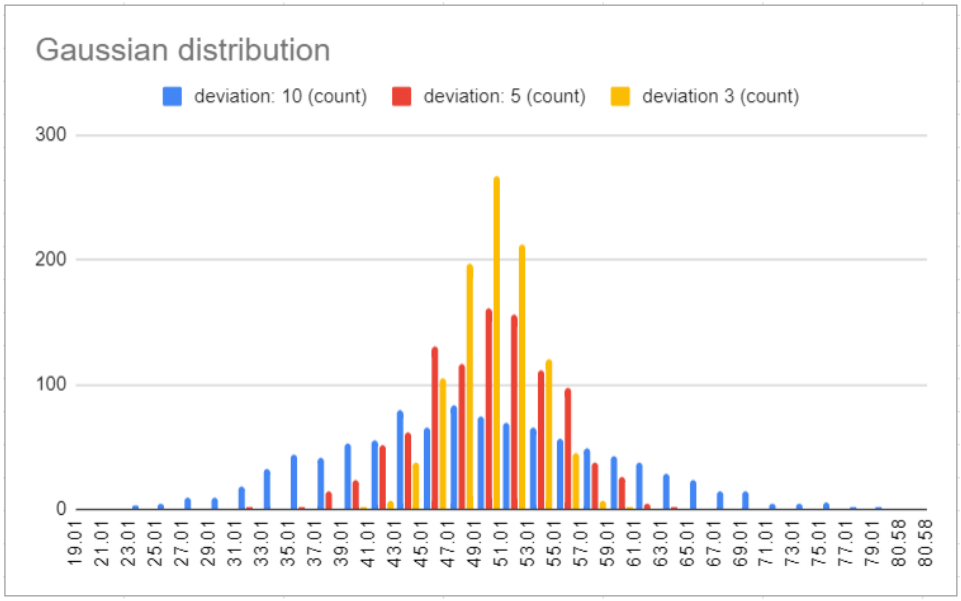

Figure 5: *Gaussian distribution with variance of the deviation*

A histogram showing how the delay in milliseconds between messages follows a poisson distribution with varying mean ranging from 1 up to 10.

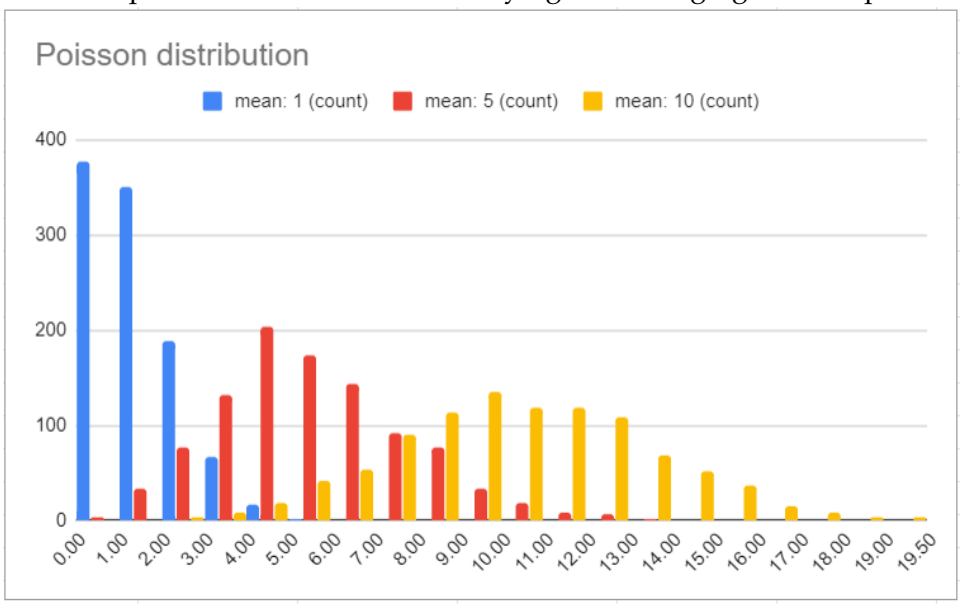

Figure 6: *Poisson distribution with variance of the mean*

4 Discussion

Our results show good examples of how traffic of the different types are handled. From the data we can easily see why for example why CBR is the go to traffic type for chat services such as Skype that require that there is no variation in delay. With a large number of independent traffic streams in a traffic application the process may look like a Poisson process. The higher mean you have the process will look more and more like normal distribution as shown in the graphs fig. 5 and fig. 6.

During our implementation of the nodes we faced some issues that stemed from understanding the meaning and intentions of how the different methods where supposed to function, especially the poisson distrubution. The main issue we faced where that there where hard to find resources defining how the delay should be distrubuted compared to the other which was were easy to find how the method should distribute the traffic once implemented.