SIMULATION PROJECT REPORT 2

Implementation of Token Bucket Algorithm

Under the guidance of:

Professor Dr. Ing Faqir Zarrar Yousaf

Presented by:

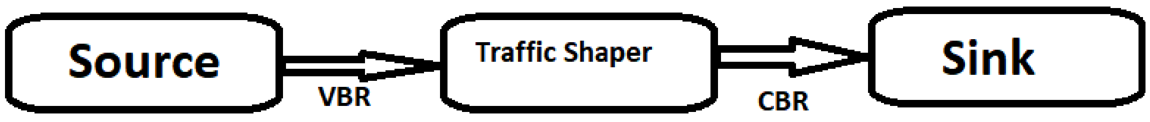
Mohammad Taha Siddiqui
11012463

Abstract:

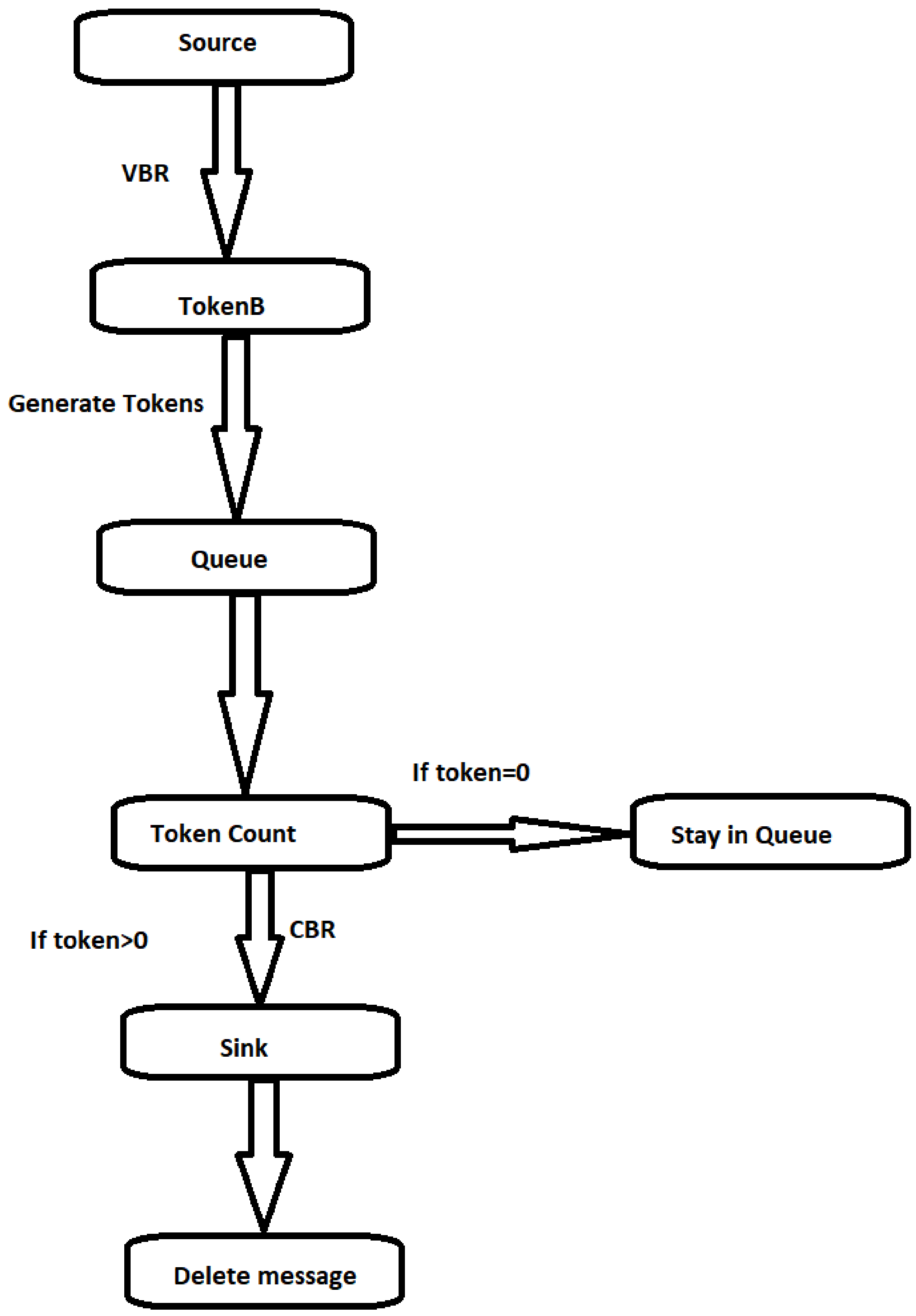
In this project, we are trying to create a network topology with two stations, one being a packet source and the other being a sink via a Traffic Shaper. The traffic from source is randomly generated following Poisson distribution for variable packet rate. The traffic shaper will queue the incoming packets from the source and then schedule them at a constant bit rate towards the sink. Thus, the traffic between source and the Traffic Shaper is VBR, while between the Traffic Shaper and sink is CBR.

Description:

Our project consists of a Source (Taha), Sink (Receiver) and a Traffic Shaper (Queue) in between.
Source generates packets at variable bit rate (VBR) and sends it to the Traffic shaper.
Traffic shaper consists of a queue and a token bucket which holds tokens.
The task of the traffic shaper is to convert the variable bit rate into constant bit rate.
Output from the traffic shaper is first in first out.
When the tokens are available for each packet from the source is then sent to the sink.



Flow Chart:


Source
VBR
TokenB
Generate Tokens
Queue
If token=0
Token Count
Stay in Queue
If token>0
CBR
Sink
Delete message

Running code:

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

class taha: public cSimpleModule
{
private:
    simtime_t timer;
    cMessage *timeoutMsg;
    int k;
    cOutVector tahavector;
public:
  taha();
  virtual ~taha();

protected:
  virtual void initialize() override;
  virtual void handleMessage(cMessage *msg) override;
};
Define_Module(taha);

taha::taha()
{
    timeoutMsg = nullptr;
}

taha::~taha()
{
    cancelAndDelete(timeoutMsg);
}

void taha::initialize(){
    timer = uniform(0,10);
    timeoutMsg = new cMessage(" start ");
    k = uniform(1,5);
    for(int i = 0; i< k; i++){
        EV<< " sending message " << endl;
        cMessage *msg = new cMessage( " Packet ");
        send(msg,"out");

    }
    scheduleAt(simTime()+timer, timeoutMsg);
}

void taha:: handleMessage(cMessage *msg){
    k = uniform(1,5);
    for(int i = 0; i< k; i++){
        EV<< " sending message " << endl;
        cMessage *msg = new cMessage( " Bucketlist ");
        send(msg,"out");

    }
    timer = uniform(0,10);
    tahavector.record(timer);
    scheduleAt(simTime()+timer, timeoutMsg);

}

class tokenB: public cSimpleModule
{
private:
    simtime_t timer;
    cMessage *timeoutMsg;
    int k;
    int tokencounter = 1;
    cQueue queue;
public:
  tokenB();
  virtual ~tokenB();

protected:
  virtual void initialize() override;
  virtual void handleMessage(cMessage *msg) override;
};
Define_Module(tokenB);

tokenB::tokenB()
{
    timeoutMsg  = nullptr;
}

tokenB::~tokenB()
{
    cancelAndDelete(timeoutMsg);
}

void tokenB::initialize(){
    timer = 9;
    timeoutMsg = new cMessage( " generate ");
    scheduleAt(simTime()+timer, timeoutMsg);
}
void tokenB:: handleMessage(cMessage *msg){
    if( msg == timeoutMsg){
        if(tokencounter<5){
            tokencounter++;
            EV<<" tokencounter value is "<< tokencounter << endl;
        }
        scheduleAt(simTime()+timer, timeoutMsg);
    }
    else if(tokencounter > 0){
        EV<< " queueing messages "<< endl;
        queue.insert(msg);
        cMessage *tmsg = (cMessage *)queue.pop();
        send(tmsg,"out");
        tokencounter--;
        EV<< " sent message to receiver, remaining tokens are: " << tokencounter
<< endl;

    }
    else{
        EV<< " received from Taha "<< endl;
        queue.insert(msg);

    }
}

class receiver: public cSimpleModule
{
private:
    cOutVector recvector;
    simsignal_t recsignal;

protected:
    void initialize() override;
    void handleMessage(cMessage *msg)override;
};

Define_Module(receiver);

void receiver:: initialize(){
    recsignal = registerSignal("rec");

}

void receiver:: handleMessage(cMessage *msg){
    EV<< " RECEIVED MESSAGE FROM tokenB "<< endl;
    simtime_t rec = simTime() - msg->getCreationTime();
    recvector.record(rec);
    delete msg;
}
```

NED File:

```
simple taha
{
    parameters:
        @display("i=block/routing");
    gates:
        output out;
}

simple tokenB
{
    parameters:
        @display("i=block/routing");
    gates:
        input in;
        output out;
}
simple receiver
{
    parameters:
        @display("i=block/routing");
    gates:
        input in;
}


network bs13
{
    submodules:
        TAHA: taha {
            parameters:
                @display("i=,cyan");
        }
        QUEUE: tokenB {
            parameters:
                @display("i=,gold");
        }
        RECEIVER: receiver{
            parameters:
                @display("i=,blue");
        }

    connections:
        TAHA.out --> {  delay = 100ms; } --> QUEUE.in;
        RECEIVER.in <-- {  delay = 100ms; } <-- QUEUE.out;
}
```

.ini File:

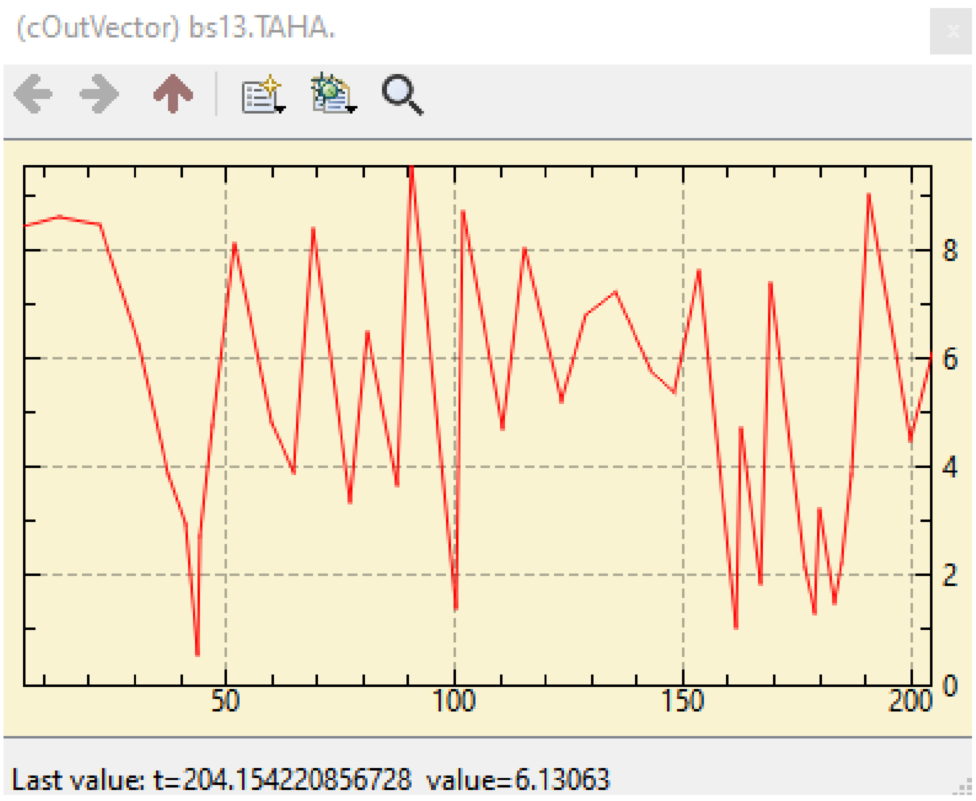
```
[General]
network = bs13
```

Output Plots:

At the source:

The packets are sent out using the uniform function. A self-timer is called to dispatch a random number of messages at irregular intervals.

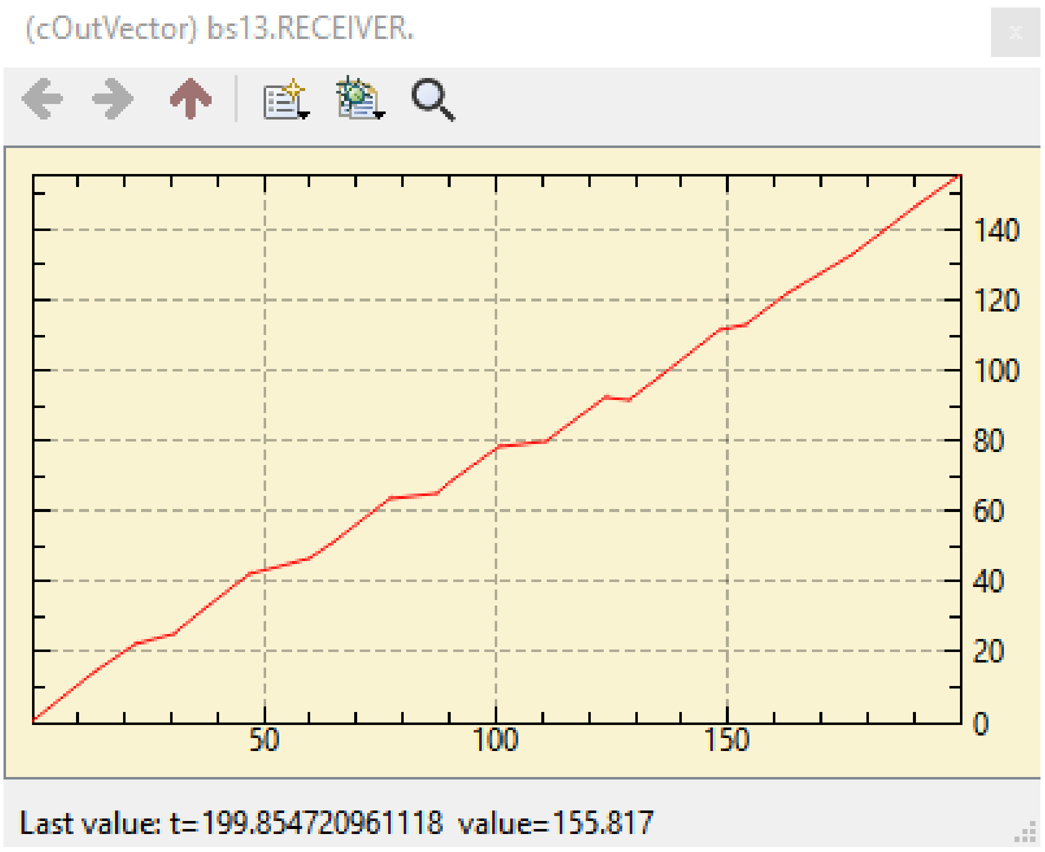
This method has been adopted to compensate for the Poisson's Distribution function.



Last value: t=204.154220856728 value=6.13063

At the receiver:

A general trend is observed at the receiver where the packets are carried to the sink/receiver at regular intervals of time.



Last value: t=199.854720961118 value=155.817

Conclusion

- Packets are sent to the traffic shaper at variable intervals.
- The tokens are generated at fixed intervals which carry the packets to the sink.
- Packets are received at a constant rate at the sink.
- The values are noted and graphs are plotted accordingly.