Communication Network

Project – GO BACK-N PROTCOL

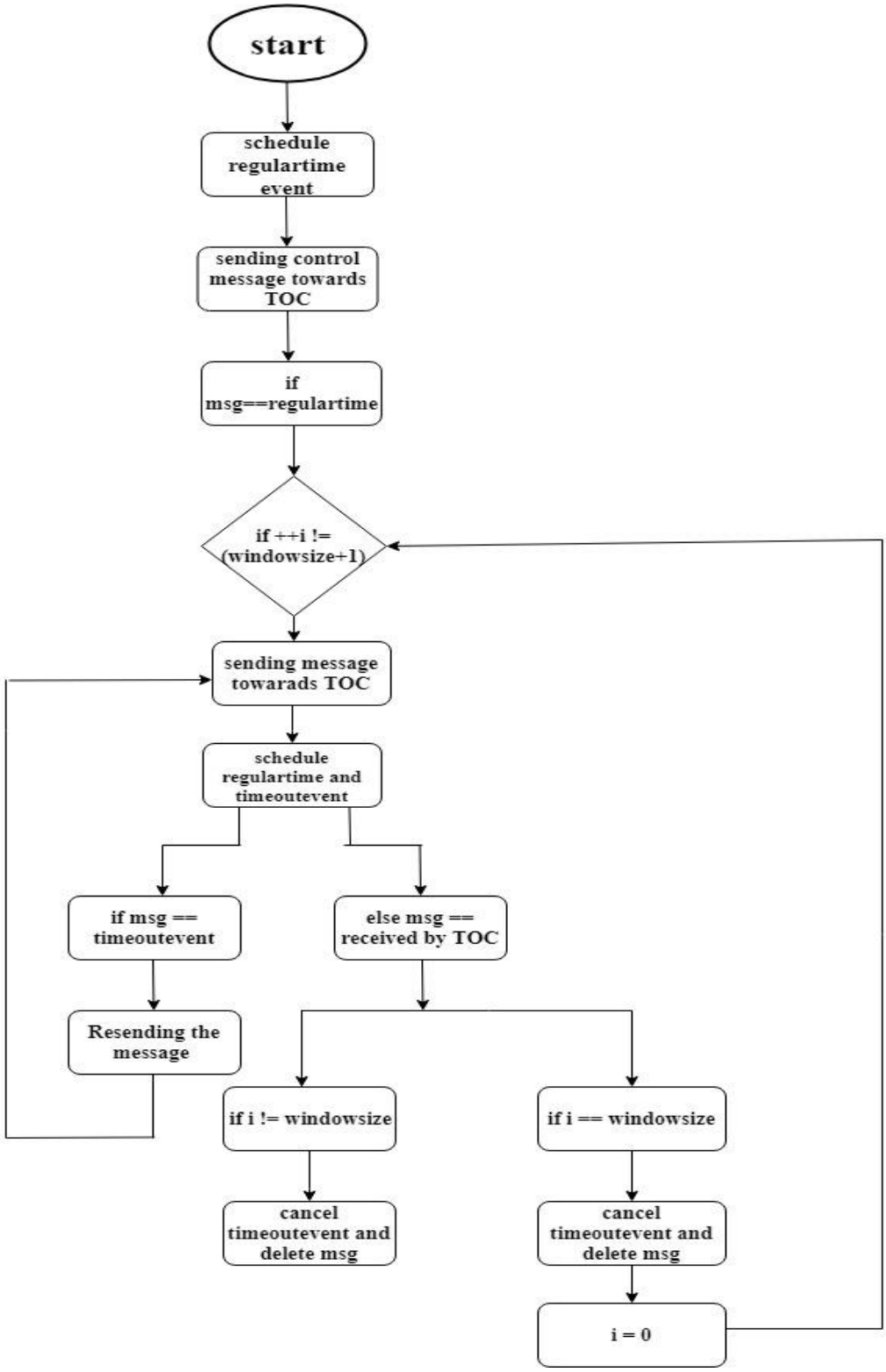
Prepared by:- Aditya Shashikant Nandgaokar (11012434)

Mohammad Taha Siddiqui (11012463)

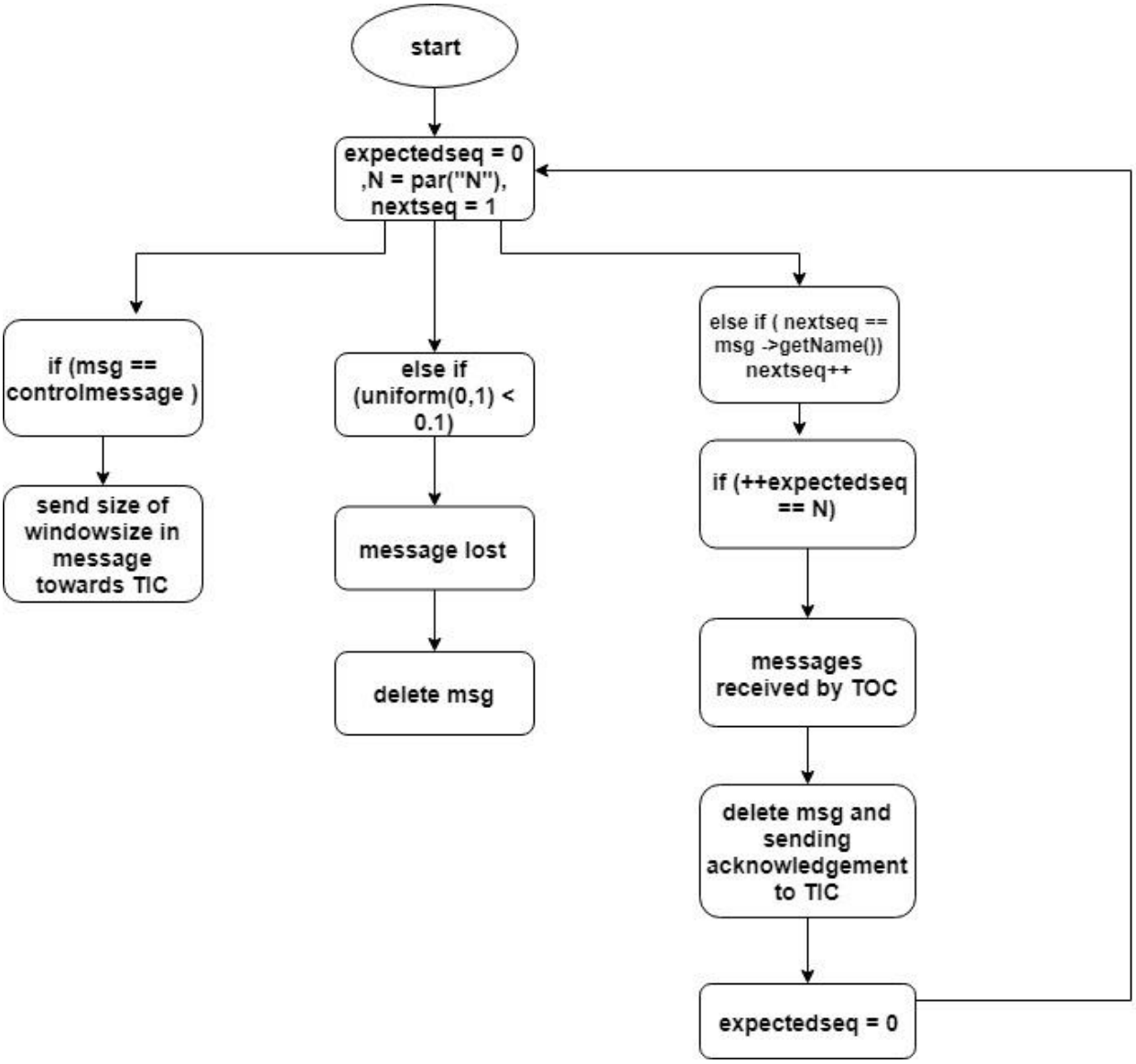
Objective:-

Using GO BACK-N PROTOCOL, we can vary the window size of our system and can also vary parameter N (i.e. the number of messages we received from TIC before sending an acknowledgement from TOC to TIC.) In this if we lose any packet we can get back that message and process will continue by sequence of messages.

Flow Chart of TIC:-



Flow chart of **TOC**:-



Source Code:-

TIC –

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

using namespace omnetpp;

class Tic2: public cSimpleModule {
private:
    simtime_t timeout;
    cMessage *timeoutevent;
    int seq;
    cMessage *message;
    simtime_t intime;
    cMessage *regulartime;
    int i;
    cMessage *count[255];
    cMessage  *temp[255];
    char timeeventseq[3];
    int windowsize;
    int j;
    int N;
    int k;
    cMessage *controlmessage;

    bool windowSizeReceived = false;
public:
    Tic2();
    virtual ~Tic2();

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

Define_Module(Tic2);

Tic2::Tic2() {
    timeoutevent = message = regulartime = controlmessage =  nullptr;

}
Tic2::~Tic2() {
    cancelAndDelete(timeoutevent);
    delete message;
    cancelAndDelete(regulartime);
}

void Tic2::initialize()
```

```
{
    seq = 0;

    timeoutevent = new cMessage("timeoutevent");
    regulartime = new cMessage("regulartime");

    windowsize = par("windowsize");
    i = 0;
    j = 0;
    k = 0;
    N = par("N");
    timeout = 0.5*N ;
    intime = 0.5;

    scheduleAt(simTime() + intime, regulartime);

    for(i=0;i<256;i++)
        count[i]=nullptr;

    cMessage *msg = new cMessage("controlmessage");
        send(msg, "out");


}

void Tic2::handleMessage(cMessage *msg) {


    if (windowSizeReceived) {
        if (msg == regulartime) {

          if(++i != (windowsize + 1)){


                EV << "sending packet towards TOC.\n";
                message = generateNewMessage();
                sendCopyOf(message);

                sprintf(timeeventseq,"%d",seq);
                timeoutevent = new cMessage(timeeventseq);
                count[seq] = timeoutevent;

                scheduleAt(simTime() + intime, regulartime);
                scheduleAt(simTime() + timeout, timeoutevent);



            }

        }

        else if (msg->isSelfMessage()) {
            EV << "Timer has expired, Resending the message "<<msg<<endl;
```

```
        seq=atoi(msg->getName())-1;

    }
    else {
       if(i != windowsize ){

        EV << "Received:" << msg->getName() << "\n";

        EV << "Timer cancelled.\n";
        j = seq - N;

        while(j <= seq){

            timeoutevent=count[j];

            if(timeoutevent)
            if(timeoutevent->isSelfMessage())
            if(timeoutevent->isScheduled())
                cancelEvent(timeoutevent);
           j++;
        }
        delete msg;
        delete message;

       }
       else {

           EV << "Received:" << msg->getName() << "\n";
           EV << "Timer cancelled.\n";
           j = seq - N;

           while(j <= seq){


            timeoutevent=count[j] ;

            if(timeoutevent)
            if(timeoutevent->isSelfMessage())
            if(timeoutevent->isScheduled())
                cancelEvent(timeoutevent);
           j++;
           }

               delete msg;
               delete message;
               i = 0;

              }



    }
}
else{
```

```
        windowSizeReceived=true;
EV<<"Receved window size."<<endl;
    }

}

cMessage *Tic2::generateNewMessage()
{
    if(seq <= 254){
    char msgname[3];
    sprintf(msgname, "%d", ++seq);
    cMessage *msg = new cMessage(msgname);
    return msg;
    }

    if(seq == 255){
    char msgname[30];
    seq = 0;
    sprintf(msgname, "packet-%d", ++seq);
    cMessage *msg = new cMessage(msgname);

    return msg;
    }
}

void Tic2::sendCopyOf(cMessage *msg) {
    cMessage *copy = (cMessage *) msg->dup();
    send(copy, "out");
}
```

TOC-

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

using namespace omnetpp;


class Toc2 : public cSimpleModule
{
  private:
    int expectedseq;
    int N, temp, a;
    cMessage *controlmessage;
    int windowsize;
    int nextSeq;
  protected:
    virtual void handleMessage(cMessage *msg) override;
```

```
    virtual void initialize() override;

};

Define_Module(Toc2);

void Toc2::initialize() {
   expectedseq = 0;
   N = par("N");
   windowsize = par("windowsize");

   nextSeq=1;

   WATCH(expectedseq);
   WATCH(nextSeq);
}


void Toc2::handleMessage(cMessage *msg)
{
    EV << "Received "<<msg << endl;
    if(strcmp("controlmessage",msg->getName())==0){
        char buf[20];
        sprintf(buf,"windowsize %d", windowsize);
        cMessage *msg = new cMessage(buf);
        send(msg, "out");
        EV << "Sending Receiver window size" << endl;
    }
    else if (uniform(0, 1) < 0.1 ) {
        EV << "\"Losing\" message " << msg << endl;
        bubble("message lost");
        delete msg;
    }
    else if(nextSeq==atoi(msg->getName())){
        nextSeq++;
        if(++expectedseq == N){

            EV << msg << " received" << endl;
            delete msg;
            send(new cMessage("ack"), "out");
            expectedseq = 0;
            }
    }
}
```

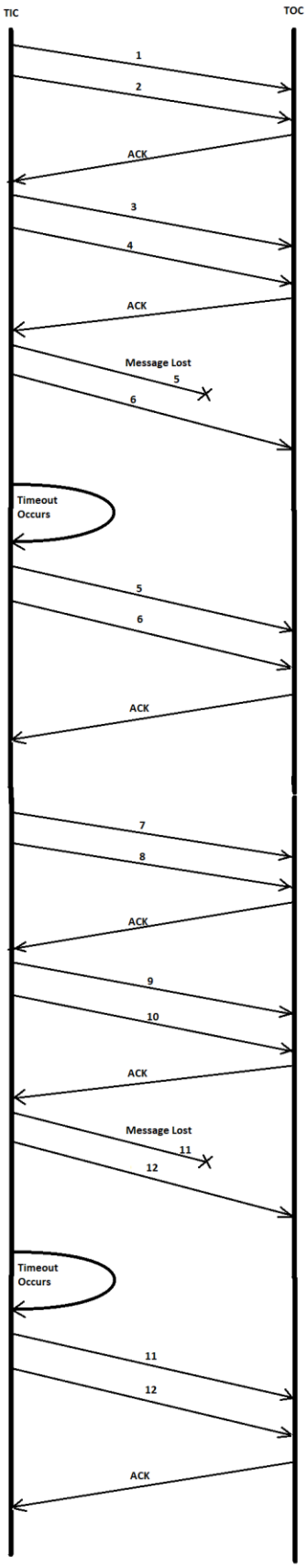
TICTOC.NED-

```
simple Tic2
{
    parameters:
        //double datarate = default(2);
       int windowsize = default(2);
       int N = default(2);
        @display("i=block/routing");

        //double datarate;
    gates:
        input in;
        output out;
}


simple Toc2
{
    parameters:
        int N = default(2);
        int windowsize = default(2);
        @display("i=block/routing");
    gates:
        input in;
        output out;

}
//
// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.
//
channel DatarateChannel
{
@class(cDatarateChannel);
@signal[channelBusy](type=long);
@signal[messageSent](type=cMessage);
@signal[messageDiscarded](type=cMessage);
@statistic[busy](...);
@statistic[utilization](...);
@statistic[packets](...);
@statistic[packetBytes](...);
@statistic[packetsDiscarded](...);
@statistic[throughput](...);
bool disabled = default(false);
double delay = default(0s) @unit(s); // propagation delay
double datarate = default(10bps) @unit(bps); // bits per second; 0=infinite
double ber = default(0); // bit error rate (BER)
double per = default(0); // packet error rate (PER)
}


network Tictoc2
{
```

```
    @display("bgb=193,190");
    submodules:
        tic: Tic2 {
            parameters:

                @display("i=,cyan;p=133.96,43.996666");
        }
        toc: Toc2 {
            parameters:

                @display("i=,gold;p=45.966667,84.71");
        }
    connections:
        tic.out --> DatarateChannel --> toc.in;

        tic.in <--  DatarateChannel <-- toc.out;
}
```

OMNET.INI-

```
network = Tictoc2
eventlog-file = ${resultdir}/${configname}-${iterationvarsf}#${repetition}.elog
eventlog-message-detail-pattern =
eventlog-recording-intervals =
record-eventlog = false
simtime-resolution = ps
**.module-eventlog-recording = false
**.tic.windowsize = 5
**.toc.N = 2
**.tic.N = 2
**.toc.windowsize = 5
```



Packet Sequence Chart

Window Size: 4
N= Ack= 2