# Code

## NED File

```
simple Tic
{
        parameters:
                @display("i=block/routing");
        gates:
                input in;
                output out;
}
simple Toc
{
        parameters:
                @display("i=block/process");
        gates:
                input in;
                output out;
}
network project1
{
        @display("bgb=214,303");
        submodules:
                tic: Tic {
                parameters:
                        @display("i=,cyan;is=s;p=98.82667,160.59334");
                }
                toc: Toc {
                parameters:
                        @display("i=,gold;is=s;p=98.82667,54.5");
                }
        connections:
                 tic.out --> {  delay = 1000ms; } --> toc.in;
                 tic.in <-- {  delay = 1000ms; } <-- toc.out;
}
```

## Source File

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

#define ASK_WINDOW_SIZE "Ask Window Size"
#define WINDOW_SIZE_REPLY "Window Size Reply"
#define DATA_MESSAGE "Data Sent"
#define TIMEOUT_MESSAGE "Timeout"
#define RECEIVED_ACK "Received"
#define NOT_RECEIVED_ACK "Not Received"
#define TIMEOUT_TIME 3.0
#define WINDOW_SIZE 5
#define ERROR_RATE 10

using namespace omnetpp;

class Tic: public cSimpleModule{
   private:
        cMessage *message;   //holds message to send data
        int counter = 0;
        int window_size = 0;    //hold Toc window size during initialization
        bool dropRandomPacket();
        void sendMessage(const char* message_string, int value);
   protected:
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
};

inline bool Tic::dropRandomPacket(){
        return ((rand() * 1.0) / RAND_MAX) * 100 > ERROR_RATE;
}

//Setup the message to send to Toc class
```

```
inline void Tic::sendMessage(const char* message_string, int value =
0){
        message = new cMessage(message_string, 0);
        //Set the name of the message
        message->setName(message_string);
        //Set the value of the message (Useful for some messages)
        message->setKind(value);
        //Send the message to Toc
        send(message,"out");
}

void Tic::initialize(){
        //Initialize random value to new seed every iteration
        srand(time(NULL));
        //Ask Toc for window size
        sendMessage(ASK_WINDOW_SIZE);
}

void Tic::handleMessage(cMessage *msg){
        //Handle Toc reply of window size
        if(!strcmp(msg->getName(), WINDOW_SIZE_REPLY)){
                //Get value of message for window size and store in class
        variable
                window_size = msg->getKind();
        }
        //Handle Toc reply for dropped packet acknowledgement
        else if(!strcmp(msg->getName(), NOT_RECEIVED_ACK)){
                //Get value of dropped packet to start sending data from this
        packet
                //Reset counter to dropped packet value
                counter = msg->getKind();
        }
        //Send n number of messages where windows size is w
        // n <= w (w - n = dropped packets)
        for (int i = 0; i < window_size; i++){
                //drop packets randomly using new random value
                //Higher the error rate, lower chance of dropping
                if(dropRandomPacket()){
                        sendMessage(DATA_MESSAGE, counter);
                        EV<< "Sending : " << counter << std::endl;
```

```
            }
            //Increment packet value to send next iteration
            counter++;
        }
}

Define_Module(Tic);


class Toc: public cSimpleModule{
   private:
        int latest_value = 0; //hold last valid value captured by Toc
        int loop_counter = 0; //temporary counter
        simtime_t timeout = TIMEOUT_TIME;
        int window_size = WINDOW_SIZE;
        cMessage *message; //message object passed to Toc
        cMessage *timeoutEvent; //timeout message object passed to Toc
        void sendMessage(const char* message_string, int value = 0);
   protected:
        virtual void initialize() override;
        virtual void handleMessage(cMessage *msg) override;
};

//Create and send message to Tic
inline void Toc::sendMessage(const char* message_string, int value){
        message = new cMessage(message_string, 0);
        message->setName(message_string);
        message->setKind(value);
        send(message,"out");
}

void Toc::initialize(){
        timeoutEvent = new cMessage(TIMEOUT_MESSAGE);
        //Schedule timeout event to execute if no message is received by Toc
        scheduleAt(simTime() + timeout, timeoutEvent);
}

void Toc::handleMessage(cMessage *msg){
        //cancel previous timeout event as message was received
        cancelEvent(timeoutEvent);
```

```
        //if message is for window size, send window size
        if(!strcmp(msg->getName(), ASK_WINDOW_SIZE)){
                sendMessage(WINDOW_SIZE_REPLY, window_size);
        }
        //if message received was normal data message from Tic
        else if(!strcmp(msg->getName(), DATA_MESSAGE)){
                //get value of message
                int current_value = msg->getKind();
                loop_counter++;
                //if new value received is higher than in order
                //this means value was skipped or dropped from Tic
                //don't update latest value as its required in RN message
                if(current_value == latest_value + 1){
                        latest_value = current_value;
                        EV << "Last: " << latest_value << std::endl;
                }
                EV << "Received: " << current_value << std::endl;
                if(loop_counter == window_size){
                        sendMessage(RECEIVED_ACK, latest_value + 1);
                        EV << "Successfully received all packets! \n";
                        loop_counter = 0;
                }
        }
        //if message is timeout, no message was received, so determine if
RR or RN to be sent
        else if(!strcmp(msg->getName(), TIMEOUT_MESSAGE)){
                EV << "Received " << loop_counter << " of " << window_size
        << " values\n";
                //if number of packets sent == size of window, send RR, else
        send RN
                sendMessage(NOT_RECEIVED_ACK, latest_value + 1);
                EV << "Missing packet " << latest_value + 1 << "\n";
                //reset loop counter to count window size messages next
        iteration
                loop_counter = 0;
        }
        //schedule new timeout event for if no messages received within
timeout
        scheduleAt(simTime() + timeout, timeoutEvent);
}
```

```
Define_Module(Toc);
```