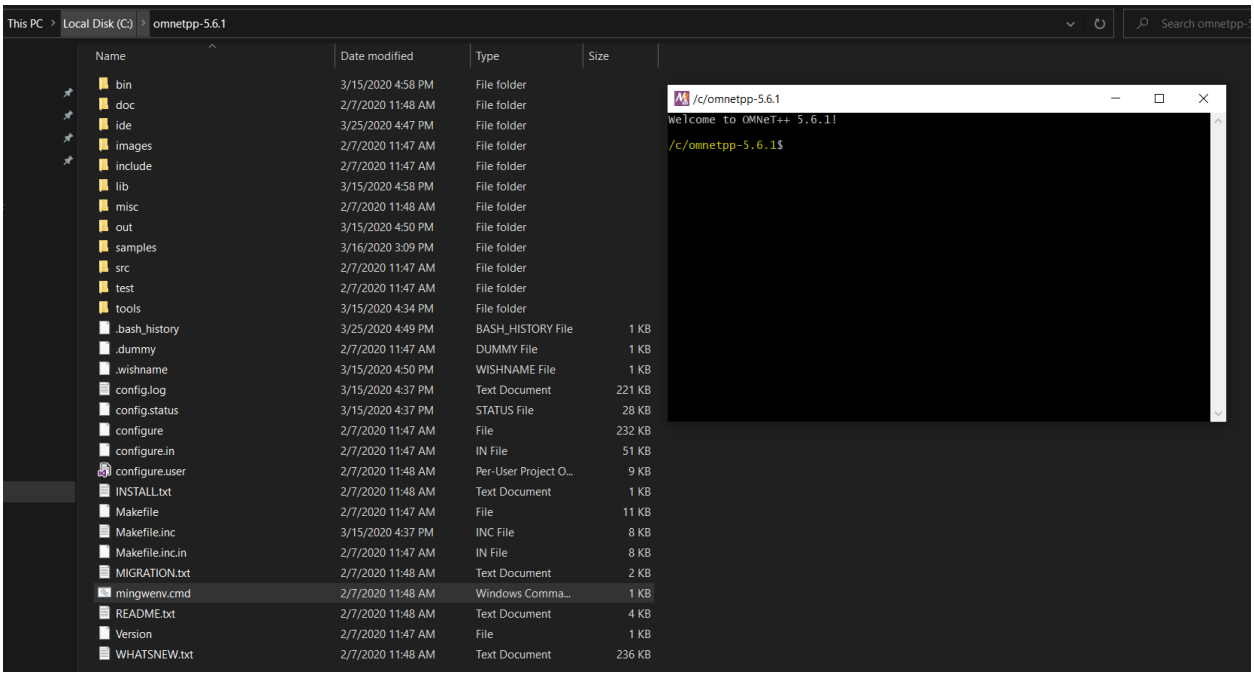
Lab 1:

1. Setup OMNET++

result:



2. Tictoc tutorial

Part1: Getting Started

new file: tictoc1.ned

```
simple Txc1
{
    gates:
        input in;
        output out;
}

//
// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.
//
network Tictoc1
{
    @display("bgb=276,174");
    submodules:
        tic: Txc1 {
            @display("p=39,29");
        }
        toc: Txc1 {
            @display("p=198,134");
```

```
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

new file: txc1.cc

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

using namespace omnetpp;

/**
 * Derive the Txc1 class from cSimpleModule. In the Tictoc1 network,
 * both the `tic' and `toc' modules are Txc1 objects, created by OMNeT++
 * at the beginning of the simulation.
 */
class Txc1 : public cSimpleModule
{
  protected:
    // The following redefined virtual function holds the algorithm.
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

// The module class needs to be registered with OMNeT++
Define_Module(Txc1);

void Txc1::initialize()
{
    // Initialize is called at the beginning of the simulation.
    // To bootstrap the tic-toc-tic-toc process, one of the modules needs
    // to send the first message. Let this be `tic'.

    // Am I Tic or Toc?
    if (strcmp("tic", getName()) == 0) {
        // create and send first message on gate "out". "tictocMsg" is an
        // arbitrary string which will be the name of the message object.
        cMessage *msg = new cMessage("tictocMessage");
        send(msg, "out");
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    // The handleMessage() method is called whenever a message arrives
    // at the module. Here, we just send it to the other module, through
    // gate `out'. Because both `tic' and `toc' does the same, the message
    // will bounce between the two.
    send(msg, "out"); // send out the message
}
```

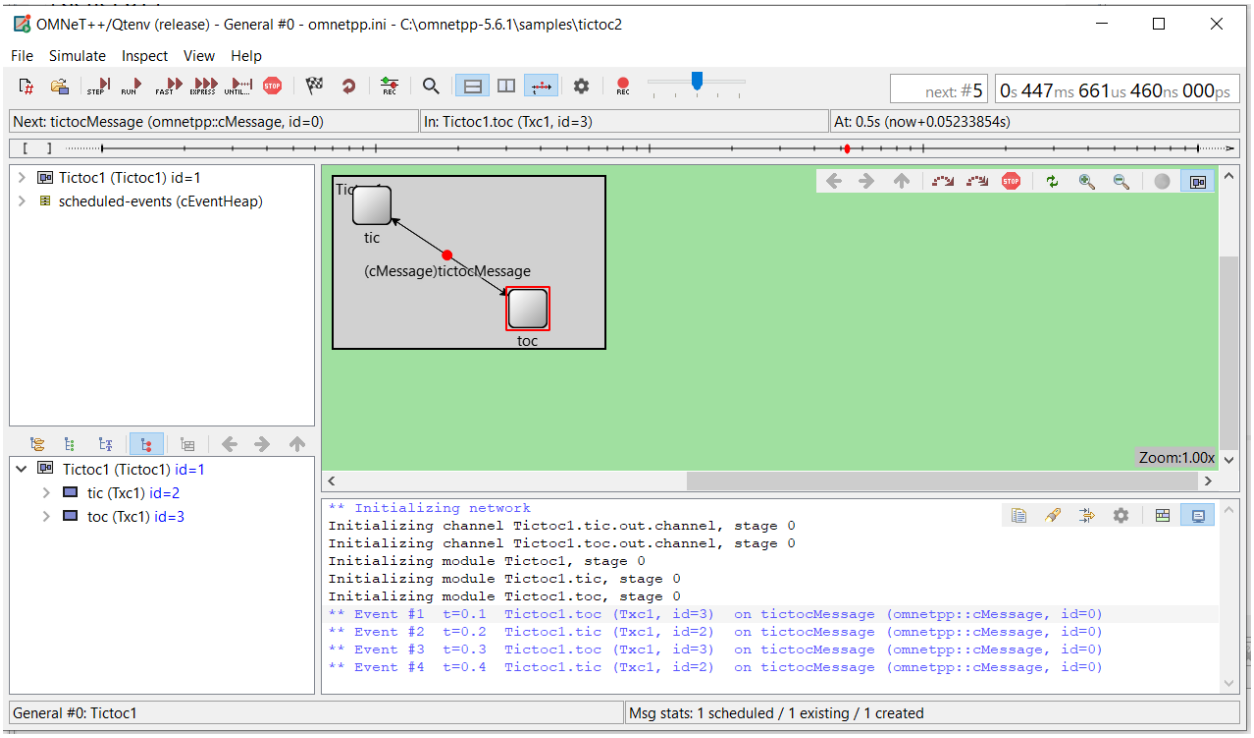
new file: omnetpp.ini

```
[General]
```

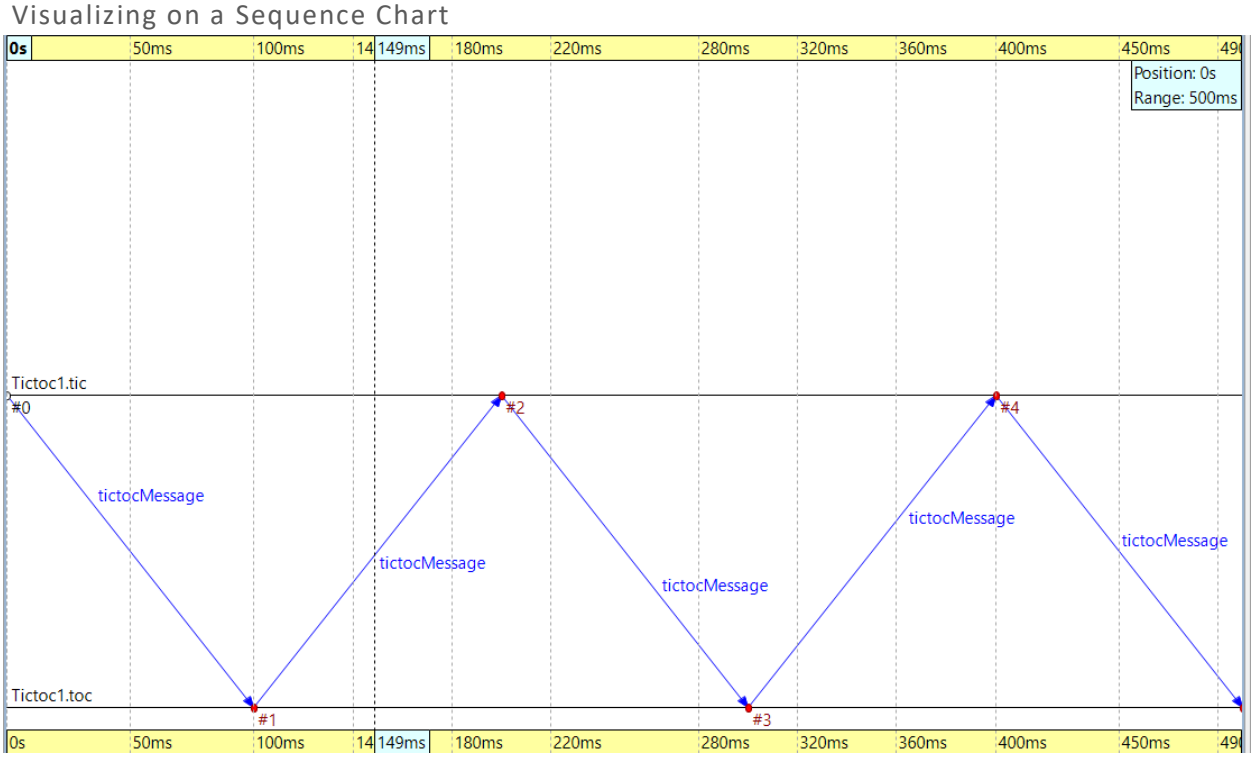
```
network = Tictoc1
```

Part2: Running the Simulation

Running the Simulation



Visualizing on a Sequence Chart



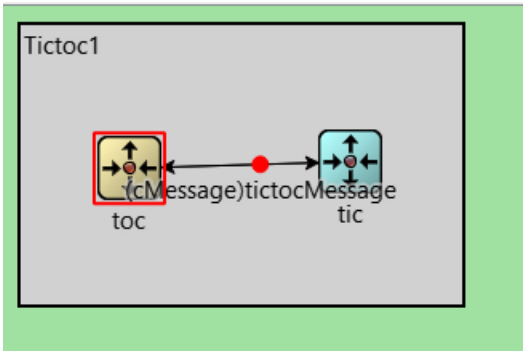
Part 3 - Enhancing the 2-node TicToc

a) Adding icons

edit file: tictoc1.ned

```
simple Txc1
{
    parameters:
        @display("i=block/routing"); // add a default icon
    gates:
        input in;
        output out;
}

network Tictoc1
{
    @display("bgb=276,174");
    submodules:
        tic: Txc1 {
            parameters:
                @display("i=,cyan");
        }
        toc: Txc1 {
            parameters:
               @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

Result:



b) Adding logging

edit file: txc1.cc

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

using namespace omnetpp;


class Txc1 : public cSimpleModule
{
```

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

Define_Module(Txc1);

void Txc1::initialize()
{
    if (strcmp("tic", getName()) == 0) {
        cMessage *msg = new cMessage("tictocMessage");
        send(msg, "out");
        EV << "Sending initial message\n"; //Log
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    send(msg, "out");
    EV << "Received message `" << msg->getName() << "', sending it out
again\n"; //Log
}
```

Result:

```
Initializing module Tictoc1, stage 0
Initializing module Tictoc1.tic, stage 0
INFO (Txc1)Tictoc1.tic:Sending initial message
Initializing module Tictoc1.toc, stage 0
** Event #1  t=0.1  Tictoc1.toc (Txc1, id=3)  on tictocMessage (omnetpp::cMessage, id=0)
INFO:Received message `tictocMessage', sending it out again
** Event #2  t=0.2  Tictoc1.tic (Txc1, id=2)  on tictocMessage (omnetpp::cMessage, id=0)
INFO:Received message `tictocMessage', sending it out again
** Event #3  t=0.3  Tictoc1.toc (Txc1, id=3)  on tictocMessage (omnetpp::cMessage, id=0)
INFO:Received message `tictocMessage', sending it out again
```

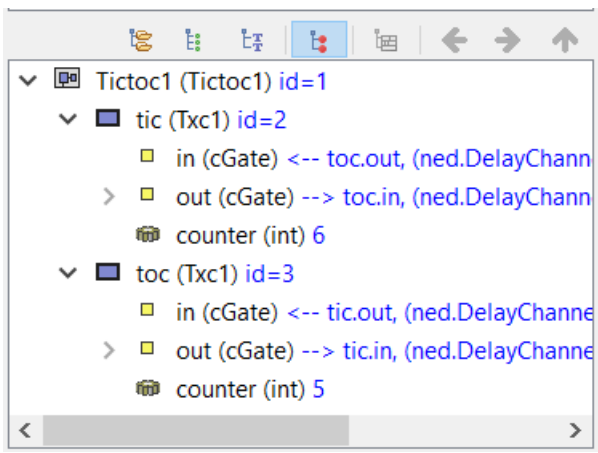
c) Adding state variables

edit file: txc1.cc

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

using namespace omnetpp;

class Txc1: public cSimpleModule {
private:
    int counter;  // Note the counter here

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

Define_Module(Txc1);

void Txc1::initialize() {
    counter = 10;
```

```
    WATCH(counter);
    if (strcmp("tic", getName()) == 0) {
        cMessage *msg = new cMessage("tictocMessage");
        send(msg, "out");
        EV << "Sending initial message\n"; //Log
    }
}

void Txc1::handleMessage(cMessage *msg) {
    counter--;
    if (counter == 0) {
        EV << getName() << "'s counter reached zero, deleting message\n";
        delete msg;
    }
    else {
        EV << getName() << "'s counter is " << counter << ", sending back
message\n";
        send(msg, "out");
    }
}
```

Result:



d) Adding parameters

edit file: tictoc1.ned

```
simple Txc1
{
    parameters:
        bool sendMsgOnInit = default(false); //new
        int limit = default(2); //new
        @display("i=block/routing");
    gates:
        input in;
        output out;
}

network Tictoc1
{
    @display("bgb=276,174");
    submodules:
```

```
        tic: Txc1 {
            parameters:
                sendMsgOnInit = true; //new
                @display("i=,cyan");
        }
        toc: Txc1 {
            parameters:
                sendMsgOnInit = false; //new
                                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

edit file: omnetpp.ini

```
[General]
network = Tictoc1
Tictoc1.toc.limit = 5
# or Tictoc4.t*c.limit=5
# or Tictoc4.*.limit=5
# or **.limit=5
# make same effect
```

edit file: txc1.cc

```
...
void Txc1::initialize() {
    counter = par("limit"); //new
    WATCH(counter);
    if (strcmp("tic", getName()) == 0) {
        cMessage *msg = new cMessage("tictocMessage");
        send(msg, "out");
        EV << "Sending initial message\n";
    }
}
...
```

e) Using NED inheritance

edit file: tictoc1.ned

```
simple Txc1
{
    parameters:
        bool sendMsgOnInit = default(false);
        int limit = default(2);
        @display("i=block/routing");
    gates:
        input in;
        output out;
}

simple Tic1 extends Txc1
{
```

```
    parameters:
        @display("i=,cyan");
        sendMsgOnInit = true;
}

simple Toc1 extends Txc1
{
    parameters:
        @display("i=,gold");
        sendMsgOnInit = false;
}

network Tictoc1
{
    @display("bgb=276,174");
    submodules:
        tic: Tic1;
        toc: Toc1;
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

f) Modeling processing delay

edit file: txc1.cc

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

using namespace omnetpp;

class Txc1: public cSimpleModule {
private:
    cMessage *event;  // pointer to the event object which we'll use for
timing
    cMessage *tictocMsg; // variable to remember the message until we send it
back

public:
    Txc1();
    virtual ~Txc1();

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

Define_Module(Txc1);

Txc1::Txc1()
{
    // Set the pointer to nullptr, so that the destructor won't crash
    // even if initialize() doesn't get called because of a runtime
    // error or user cancellation during the startup process.
    event = tictocMsg = nullptr;
```

```
}

Txc1::~Txc1()
{
    // Dispose of dynamically allocated the objects
    cancelAndDelete(event);
    delete tictocMsg;
}

void Txc1::initialize()
{
    // Create the event object we'll use for timing -- just any ordinary
message.
    event = new cMessage("event");

    // No tictoc message yet.
    tictocMsg = nullptr;

    if (strcmp("tic", getName()) == 0) {
        // We don't start right away, but instead send an message to
ourselves
        // (a "self-message") -- we'll do the first sending when it arrives
        // back to us, at t=5.0s simulated time.
        EV << "Scheduling first send to t=5.0s\n";
        tictocMsg = new cMessage("tictocMsg");
        scheduleAt(5.0, event);
    }
}

void Txc1::handleMessage(cMessage *msg)
{
    // There are several ways of distinguishing messages, for example by
message
    // kind (an int attribute of cMessage) or by class using dynamic_cast
    // (provided you subclass from cMessage). In this code we just check if
we
    // recognize the pointer, which (if feasible) is the easiest and fastest
    // method.
    if (msg == event) {
        // The self-message arrived, so we can send out tictocMsg and nullptr
out
        // its pointer so that it doesn't confuse us later.
        EV << "Wait period is over, sending back message\n";
        send(tictocMsg, "out");
        tictocMsg = nullptr;
    }
    else {
        // If the message we received is not our self-message, then it must
        // be the tic-toc message arriving from our partner. We remember its
        // pointer in the tictocMsg variable, then schedule our self-message
        // to come back to us in 1s simulated time.
        EV << "Message arrived, starting to wait 1 sec...\n";
        tictocMsg = msg;
        scheduleAt(simTime()+ 1.0, event);
    }
}
```

result:

```
Initializing module Tictoc1.tic, stage 0
INFO (Tic1)Tictoc1.tic:Scheduling first send to t=5.0s
Initializing module Tictoc1.toc, stage 0
** Event #1  t=5  Tictoc1.tic (Tic1, id=2)  on selfmsg event (omnetpp::cMessage, id=0)
INFO:Wait period is over, sending back message
** Event #2  t=5.1  Tictoc1.toc (Toc1, id=3)  on tictocMsg (omnetpp::cMessage, id=1)
INFO:Message arrived, starting to wait 1 sec...
** Event #3  t=6.1  Tictoc1.toc (Toc1, id=3)  on selfmsg event (omnetpp::cMessage, id=2)
INFO:Wait period is over, sending back message
** Event #4  t=6.2  Tictoc1.tic (Tic1, id=2)  on tictocMsg (omnetpp::cMessage, id=1)
INFO:Message arrived, starting to wait 1 sec...
** Event #5  t=7.2  Tictoc1.tic (Tic1, id=2)  on selfmsg event (omnetpp::cMessage, id=0)
INFO:Wait period is over, sending back message
```

g) Random numbers and parameters

edit file txc1.cc

```
void Txc1::handleMessage(cMessage *msg) {
    if (msg == event) {
        EV << "Wait period is over, sending back message\n";
        send(tictocMsg, "out");
        tictocMsg = nullptr;
    } else {
        // "Lose" the message with 0.1 probability:
        if (uniform(0, 1) < 0.1) {
            EV << "\"Losing\" message\n";
            delete msg;
        } else {
            // The "delayTime" module parameter can be set to values like
            // "exponential(5)" (tictoc7.ned, omnetpp.ini), and then here
            // we'll get a different delay every time.
            simtime_t delay = par("delayTime");

            EV << "Message arrived, starting to wait " << delay << "
secs...\n";
            tictocMsg = msg;
            scheduleAt(simTime() + delay, event);
        }
    }
}
```

edit file: omnetpp.ini

[General]

```
network = Tictoc1
# argument to exponential() is the mean; truncnormal() returns values from
# the normal distribution truncated to nonnegative values
Tictoc7.tic.delayTime = exponential(3s)
Tictoc7.toc.delayTime = truncnormal(3s,1s)
```

edit file: tictoc1.ned

```
simple Txc1
{
    parameters:
```

```
        volatile double delayTime @unit(s);    // delay before sending back
message
        @display("i=block/routing");
    gates:
        input in;
        output out;
}

network Tictoc1
{
    submodules:
        tic: Txc1 {
            parameters:
                @display("i=,cyan");
        }
        toc: Txc1 {
            parameters:
                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

result:

```
** Event #9  t=17.529696499957  Tictoc1.tic (Txc1, id=2)  on selfmsg event (omnetpp
INFO:Wait period is over, sending back message
** Event #10  t=17.629696499957  Tictoc1.toc (Txc1, id=3)  on tictocMsg (omnetpp::cMessage, id=1)
INFO:Message arrived, starting to wait 2.089757553678 secs...
** Event #11  t=19.719454053635  Tictoc1.toc (Txc1, id=3)  on selfmsg event (omnetpp::cMessage, id=2)
INFO:Wait period is over, sending back message
** Event #12  t=19.819454053635  Tictoc1.tic (Txc1, id=2)  on tictocMsg (omnetpp::cMessage, id=1)
INFO:Message arrived, starting to wait 6.670573266109 secs...
** Event #13  t=26.490027319744  Tictoc1.tic (Txc1, id=2)  on selfmsg event (omnetpp::cMessage, id=0)
INFO:Wait period is over, sending back message
** Event #14  t=26.590027319744  Tictoc1.toc (Txc1, id=3)  on tictocMsg (omnetpp::cMessage, id=1)
INFO:"Losing" message
<!> No more events, simulation completed -- at t=26.590027319744s, event #14
```

h) Timeout, cancelling timers

edit file: tictoc1.ned

```
simple Tic1
{
    parameters:
        @display("i=block/routing");
    gates:
        input in;
        output out;
}

simple Toc1
{
    parameters:
        @display("i=block/process");
    gates:
        input in;
```

```
        output out;
}

network Tictoc1
{
    submodules:
        tic: Tic1 {
            parameters:
                @display("i=,cyan");
        }
        toc: Toc1 {
            parameters:
                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

edit file: txc1.cc

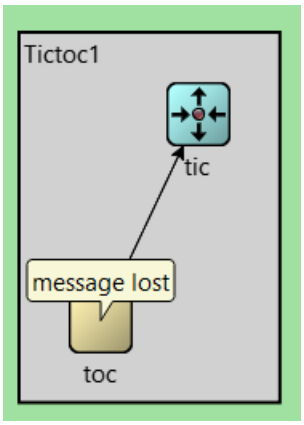
```
#include <string.h>
#include <omnetpp.h>

using namespace omnetpp;

class Tic1: public cSimpleModule {
private:
    simtime_t timeout;  // timeout
    cMessage *timeoutEvent;  // holds pointer to the timeout self-message

public:
    Tic1();
    virtual ~Tic1();

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

Define_Module(Tic1);

Tic1::Tic1() {
    timeoutEvent = nullptr;
}

Tic1::~Tic1() {
    cancelAndDelete(timeoutEvent);
}

void Tic1::initialize() {
    // Initialize variables.
    timeout = 1.0;
    timeoutEvent = new cMessage("timeoutEvent");

    // Generate and send initial message.
```

```
    EV << "Sending initial message\n";
    cMessage *msg = new cMessage("tictocMsg");
    send(msg, "out");
    scheduleAt(simTime() + timeout, timeoutEvent);
}

void Tic1::handleMessage(cMessage *msg) {
    if (msg == timeoutEvent) {
        // If we receive the timeout event, that means the packet hasn't
        // arrived in time and we have to re-send it.
        EV << "Timeout expired, resending message and restarting timer\n";
        cMessage *newMsg = new cMessage("tictocMsg");
        send(newMsg, "out");
        scheduleAt(simTime() + timeout, timeoutEvent);
    } else {  // message arrived
              // Acknowledgement received -- delete the received message and
cancel
              // the timeout event.
        EV << "Timer cancelled.\n";
        cancelEvent(timeoutEvent);
        delete msg;

        // Ready to send another one.
        cMessage *newMsg = new cMessage("tictocMsg");
        send(newMsg, "out");
        scheduleAt(simTime() + timeout, timeoutEvent);
    }
}

/**
 * Sends back an acknowledgement -- or not.
 */
class Toc1: public cSimpleModule {
protected:
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Toc1);

void Toc1::handleMessage(cMessage *msg) {
    if (uniform(0, 1) < 0.1) {
        EV << "\"Losing\" message.\n";
        bubble("message lost");  // making animation more informative...
        delete msg;
    } else {
        EV << "Sending back same message as acknowledgement.\n";
        send(msg, "out");
    }
}
```

edit file: omnetpp.ini

```
[General]
network = Tictoc1
```

result:



i) Retransmitting the same message

edit file: txc1.cc

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

using namespace omnetpp;

class Tic1: public cSimpleModule {
private:
    simtime_t timeout;  // timeout
    cMessage *timeoutEvent;  // holds pointer to the timeout self-message
    int seq; // message sequence number
    cMessage *message; // message that has to be re-sent on timeout
public:
    Tic1();
    virtual ~Tic1();

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

Define_Module(Tic1);

Tic1::Tic1() {
    timeoutEvent = nullptr;
}

Tic1::~Tic1() {
    cancelAndDelete(timeoutEvent);
    delete message;
}

void Tic1::initialize() {
// Initialize variables.
    seq = 0;
    timeout = 1.0;
```

```
    timeoutEvent = new cMessage("timeoutEvent");

// Generate and send initial message.
    EV << "Sending initial message\n";
    message = generateNewMessage();
    sendCopyOf (message);
    scheduleAt(simTime() + timeout, timeoutEvent);
}

void Tic1::handleMessage(cMessage *msg) {
    if (msg == timeoutEvent) {
        // If we receive the timeout event, that means the packet hasn't
        // arrived in time and we have to re-send it.
        EV << "Timeout expired, resending message and restarting timer\n";
        cMessage *newMsg = new cMessage("tictocMsg");
        sendCopyOf(message);
        scheduleAt(simTime() + timeout, timeoutEvent);
    } else {  // message arrived
        // Acknowledgement received!
        EV << "Received: " << msg->getName() << "\n";
        delete msg;

        // Also delete the stored message and cancel the timeout event.
        EV << "Timer cancelled.\n";
        cancelEvent(timeoutEvent);
        delete message;

        // Ready to send another one.
        message = generateNewMessage();
        sendCopyOf(message);
        scheduleAt(simTime() + timeout, timeoutEvent);
    }
}

cMessage* Tic1::generateNewMessage() {
    // Generate a message with a different name every time.
    char msgname[20];
    sprintf(msgname, "tic-%d", ++seq);
    cMessage *msg = new cMessage(msgname);
    return msg;
}

void Tic1::sendCopyOf(cMessage *msg) {
    // Duplicate message and send the copy.
    cMessage *copy = (cMessage*) msg->dup();
    send(copy, "out");
}

/**
 * Sends back an acknowledgement -- or not.
 */
class Toc1: public cSimpleModule {
protected:
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Toc1);
```

```
void Toc1::handleMessage(cMessage *msg) {
    if (uniform(0, 1) < 0.1) {
        EV << "\"Losing\" message.\n";
        bubble("message lost");
        delete msg;
    } else {
        EV << "Sending back same message as acknowledgement.\n";
        send(msg, "out");
    }
}
```

result:

```
INFO:Received: tic-30340
INFO:Timer cancelled.
** Event #67561  t=9508.1  Tictoc1.toc (Toc1, id=3)  on tic-30341 (omnetpp::cMessage, id=67562)
INFO:Sending back same message as acknowledgement.
** Event #67562  t=9508.2  Tictoc1.tic (Tic1, id=2)  on tic-30341 (omnetpp::cMessage, id=67562)
INFO:Received: tic-30341
INFO:Timer cancelled.
** Event #67563  t=9508.3  Tictoc1.toc (Toc1, id=3)  on tic-30342 (omnetpp::cMessage, id=67564)
INFO:Sending back same message as acknowledgement.
** Event #67564  t=9508.4  Tictoc1.tic (Tic1, id=2)  on tic-30342 (omnetpp::cMessage, id=67564)
INFO:Received: tic-30342
INFO:Timer cancelled.
** Event #67565  t=9508.5  Tictoc1.toc (Toc1, id=3)  on tic-30343 (omnetpp::cMessage, id=67566)
INFO:Sending back same message as acknowledgement.
** Event #67566  t=9508.6  Tictoc1.tic (Tic1, id=2)  on tic-30343 (omnetpp::cMessage, id=67566)
INFO:Received: tic-30343
INFO:Timer cancelled.
** Event #67567  t=9508.7  Tictoc1.toc (Toc1, id=3)  on tic-30344 (omnetpp::cMessage, id=67568)
INFO:"Losing" message.
** Event #67568  t=9509.6  Tictoc1.tic (Tic1, id=2)  on selfmsg timeoutEvent (omnetpp::cMessage, id=0)
INFO:Timeout expired, resending message and restarting timer
** Event #67569  t=9509.7  Tictoc1.toc (Toc1, id=3)  on tic-30344 (omnetpp::cMessage, id=67570)
INFO:Sending back same message as acknowledgement.
```