```
//tictoc1.ned
// module 정의
simple Txc1
{
    gates:
        input in;
        output out;
}
//
// Txc1이라는 module에 in, out gates 설정
// Two instances (tic and toc) of Txc1 connected both ways.
// Tic and toc will pass messages to one another.

// Tictioc1 이라는 network 정의
network Tictoc1
{
    submodules: // network에서 submodule 정의
        tic: Txc1; // tic은 모듈 Txc1이다.
        toc: Txc1; // toc 또한 "
    connections: // network에서 connection 관계 정의
        tic.out --> {  delay = 100ms; } --> toc.in;
        // tic에서 out gate를 통해 100ms delay를 가지고 toc의 in gate로 간다.
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

```
// txc1.cc
// ned 파일의 실질적인 움직임을 정의
#include <string.h>
#include <omnetpp.h>
/**
 * 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.
 */
// cSimpleModule에 정의되어있는(OMNeT++에 정의) 모듈 Txc1의 동작을 정의한다.
class Txc1 : public cSimpleModule
{
  protected:
    // The following redefined virtual function holds the algorithm.
    // initialize와 handleMessage가 모듈 동작의 대부분이다.
    virtual void initialize();
    // msg를 받아 처리한다.
    virtual void handleMessage(cMessage *msg);
};
// The module class needs to be registered with OMNeT++
// 모듈 등록
Define_Module(Txc1);
```

```
// Functions definition
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?
    // getName()은 모듈의 이름을 가져온다. -> tic에서 시작.
    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.
        // "tictocMsg"라는 메세지를 만들어서 msg에 저장.
        cMessage *msg = new cMessage("tictocMsg");
        // send(msg, gateName, gateIndex), return 값은 딜레이되서 나온 결과값(ned에 정의)
        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");
}
```

```
// tictoc2.ned
// Here we make the model look a bit prettier in the GUI. We assign the
// "block/routing" icon to the simple module. All submodules of type
// Txc2 will use this icon by default
//
simple Txc2
{
    parameters:
        @display("i=block/routing");
        // add a default icon
    gates:
        input in;
        output out;
}

//
// Make the two module look a bit different with colorization effect.
// Use cyan for `tic', and yellow for `toc'.
//
network Tictoc2
{
    submodules:
        tic: Txc2 {
            parameters:
                @display("i=,cyan"); // do not change the icon (first arg of i=) just colorize it
        }
        toc: Txc2 {
            parameters:
                @display("i=,gold"); // here too
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

```
// txc2.cc
#include <string.h>
#include <omnetpp.h>
/**
 * In this class we add some debug messages to Txc1. When you run the
 * simulation in the OMNeT++ GUI Tkenv, the output will appear in
 * the main text window, and you can also open separate output windows
 * for `tic' and `toc'.
 */
class Txc2 : public cSimpleModule
{
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

Define_Module(Txc2);

void Txc2::initialize()
{
    if (strcmp("tic", getName()) == 0)
    {
        // The `ev' object works like `cout' in C++.
        EV << "Sending initial message\n";
        cMessage *msg = new cMessage("tictocMsg");
        send(msg, "out");
    }
}

void Txc2::handleMessage(cMessage *msg)
{
    // msg->getName() is name of the msg object, here it will be "tictocMsg".
    EV << "Received message `" << msg->getName() << "', sending it out again\n";
    send(msg, "out");
}
```

```
** Initializing network
Initializing channel Tictoc2.tic.out.channel, stage 0
Initializing channel Tictoc2.toc.out.channel, stage 0
Initializing module Tictoc2, stage 0
Tictoc2.tic: Initializing module Tictoc2.tic, stage 0
Tictoc2.tic: Sending initial message
Tictoc2.toc: Initializing module Tictoc2.toc, stage 0
** Event #1  t=0.1  Tictoc2.toc (Txc2, id=3), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #2  t=0.2  Tictoc2.tic (Txc2, id=2), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #3  t=0.3  Tictoc2.toc (Txc2, id=3), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #4  t=0.4  Tictoc2.tic (Txc2, id=2), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #5  t=0.5  Tictoc2.toc (Txc2, id=3), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #6  t=0.6  Tictoc2.tic (Txc2, id=2), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #7  t=0.7  Tictoc2.toc (Txc2, id=3), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #8  t=0.8  Tictoc2.tic (Txc2, id=2), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #9  t=0.9  Tictoc2.toc (Txc2, id=3), on `tictocMsg' (cMessage, id=0)
Received message `tictocMsg', sending it out again
** Event #10  t=1  Tictoc2.tic (Txc2, id=2), on `tictocMsg' (cMessage, id=0)
```

```
// tictoc3.ned
simple Txc3
{
    parameters:
        @display("i=block/routing");
    gates:
        input in;
        output out;
}

//
// Same as Tictoc2.
//
network Tictoc3
{
    submodules:
        tic: Txc3 {
            parameters:
                @display("i=,cyan");
        }
        toc: Txc3 {
            parameters:
                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

```
// txc3.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * In this class we add a counter, and delete the message after ten exchanges.
 */
class Txc3 : public cSimpleModule
{
  private:
    int counter;  // Note the counter here
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc3);
void Txc3::initialize()
{
    // Initialize counter to ten. We'll decrement it every time and delete
    // the message when it reaches zero.
    counter = 10; // 카운터 기능을 넣었다.
    // The WATCH() statement below will let you examine the variable under
    // Tkenv. After doing a few steps in the simulation, double-click either
    // `tic' or `toc', select the Contents tab in the dialog that pops up,
    // and you'll find "counter" in the list.
    WATCH(counter);

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

```
void Txc3::handleMessage(cMessage *msg)
{
    // Increment counter and check value.
    counter--;
    if (counter==0)
    {
        // If counter is zero, delete message. If you run the model, you'll
        // find that the simulation will stop at this point with the message
        // "no more events".
        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");
    }
}
```

```
// Txc4.ned
simple Txc4
{
    parameters:
        // whether the module should send out a message on initialization
        // 이렇게 함으로서 초기에 누구를 먼저 보낼 지 정할 수 있다.
        bool sendMsgOnInit = default(false); // sendMsgOnInit을 false로 지정.
        int limit = default(2);   // another parameter with a default value
        @display("i=block/routing");
    gates:
        input in;
        output out;
}
network Tictoc4
{
    submodules:
        tic: Txc4 {
            parameters:
                // tic의 sendMsgOnInit을 true로 지정함으로서 순서를 지정하는 효과.
                sendMsgOnInit = true;
                @display("i=,cyan");
        }
        toc: Txc4 {
            parameters:
                sendMsgOnInit = false;
                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

```cpp
// txc4.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * In this step you'll learn how to add input parameters to the simulation:
 * we'll turn the "magic number" 10 into a parameter.
 */
class Txc4 : public cSimpleModule
{
  private:
    int counter;
   // 각 모듈에 private member변수로 counter를 지정함으로서,
   // 각 모듈이 개별적인 counter를 갖게된다.
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc4);
void Txc4::initialize()
{
    // Initialize the counter with the "limit" module parameter, declared
    // in the NED file (tictoc4.ned).
    counter = par("limit");
    // we no longer depend on the name of the module to decide
    // whether to send an initial message
    if (par("sendMsgOnInit").boolValue() == true)
    {
        EV << "Sending initial message\n";
        cMessage *msg = new cMessage("tictocMsg");
        send(msg, "out");
    }
}
void Txc4::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");
    }
}
```

```
// tictoc5.ned
// Same as Txc4. This module will be the base of the Tic and Toc types.
//
simple Txc5
{
    parameters:
        bool sendMsgOnInit = default(false);
        int limit = default(2);
        @display("i=block/routing");
    gates:
        input in;
        output out;
}
//
// Specialize the module by defining parameters. We could have left the whole body
// empty, because the default value of the sendMsgOnInit parameter is false anyway.
// Note that the limit parameter is still unbound here.
// 각 모듈을 또 만들어서 넣는다.
simple Tic5 extends Txc5
{
    parameters:
        @display("i=,cyan");
        sendMsgOnInit = true;   // Tic modules should send a message on init
}

//
// Specialize the module by defining parameters. We could have left the whole body
// empty, because the default value of the sendMsgOnInit parameter is false anyway.
// Note that the limit parameter is still unbound here.
//
```

```
simple Toc5 extends Txc5
{
    parameters:
        @display("i=,gold");
        sendMsgOnInit = false;  // Toc modules should NOT send a message on init
}

//
// Adding module parameters.
//
network Tictoc5
{
    submodules:
        tic: Tic5;  // the limit parameter is still unbound here. We will get it from the ini file
        toc: Toc5;
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

```
// txc6.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * In the previous models, `tic' and `toc' immediately sent back the
 * received message. Here we'll add some timing: tic and toc will hold the
 * message for 1 simulated second before sending it back. In OMNeT++
 * such timing is achieved by the module sending a message to itself.
 * Such messages are called self-messages (but only because of the way they
 * are used, otherwise they are completely ordinary messages) or events.
 * Self-messages can be "sent" with the scheduleAt() function, and you can
 * specify when they should arrive back at the module.
 * We leave out the counter, to keep the source code small.
 */
// 메세지 홀드 구현. OMNeT에서는 자기 자신에게 보내는 것으로 구현한다. scheduleAt()함수로 구현.
class Txc6 : 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:
    Txc6(); // 생성자.
    virtual ~Txc6(); // 소멸자.
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc6);
```

```cpp
Txc6::Txc6()
{
    // Set the pointer to NULL, 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 = NULL;
}
Txc6::~Txc6()
{
    // Dispose of dynamically allocated the objects
    cancelAndDelete(event); // void cancelAndDelete(cMessage *msg);
    delete tictocMsg;
}
void Txc6::initialize()
{
    // Create the event object we'll use for timing -- just any ordinary message.
    event = new cMessage("event");
    // No tictoc message yet.
    tictocMsg = NULL;
    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); // 자신에게 재전송의 의미.
    }
}
```

```
void Txc6::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 NULL out
        // its pointer so that it doesn't confuse us later.
        EV << "Wait period is over, sending back message\n";
        send(tictocMsg, "out");
        tictocMsg = NULL;
    }
    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);
    }
}
```

```
// Txc7.ned
simple Txc7
{
    parameters:
       // 딜레이 변수, 난수값이다.
        volatile double delayTime @unit(s);   // delay before sending back message
        @display("i=block/routing");
    gates:
        input in;
        output out;
}

network Tictoc7
{
    submodules:
        tic: Txc7 {
            parameters:
                @display("i=,cyan");
        }
        toc: Txc7 {
            parameters:
                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

```
// txc7.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * In this step we'll introduce random numbers. We change the delay from 1s
 * to a random value which can be set from the NED file or from omnetpp.ini.
 * In addition, we'll "lose" (delete) the packet with a small probability.
 */
// 난수 생성, 확률로 패킷손실까지.
class Txc7 : public cSimpleModule
{
  private:
    cMessage *event;
    cMessage *tictocMsg;
  public:
    Txc7();
    virtual ~Txc7();
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc7);
Txc7::Txc7()
{
    event = tictocMsg = NULL;
}
Txc7::~Txc7()
{
    cancelAndDelete(event);
    delete tictocMsg;
}
```

```
void Txc7::initialize()
{
    event = new cMessage("event");
    tictocMsg = NULL;

    if (strcmp("tic", getName()) == 0)
    {
        EV << "Scheduling first send to t=5.0s\n";
        scheduleAt(5.0, event);
        tictocMsg = new cMessage("tictocMsg");
    }
}
void Txc7::handleMessage(cMessage *msg)
{
    if (msg == event)
    {
        EV << "Wait period is over, sending back message\n";
        send(tictocMsg, "out");
        tictocMsg = NULL;
    }
    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);
        }
    }
}
```

```
// Tictoc8.ned
simple Tic8
{
    parameters:
        @display("i=block/routing");
    gates:
        input in;
        output out;
}

simple Toc8
{
    parameters:
        @display("i=block/process");
    gates:
        input in;
        output out;
}

network Tictoc8
{
    submodules:
        tic: Tic8 {
            parameters:
                @display("i=,cyan");
        }
        toc: Toc8 {
            parameters:
                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

```
// txc8.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * Let us take a step back, and remove random delaying from the code.
 * We'll leave in, however, losing the packet with a small probability.
 * And, we'll we do something very common in telecommunication networks:
 * if the packet doesn't arrive within a certain period, we'll assume it
 * was lost and create another one. The timeout will be handled using
 * (what else?) a self-message.
 */
// 메세지가 도착하지 않았을 때, 오지 않은 것으로 간주, 다시 생성한다.
class Tic8 : public cSimpleModule
{
  private:
    simtime_t timeout;  // timeout
    cMessage *timeoutEvent;  // holds pointer to the timeout self-message
  public:
    Tic8();
    virtual ~Tic8();
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Tic8);
Tic8::Tic8()
{
    timeoutEvent = NULL;
}
Tic8::~Tic8()
{
    cancelAndDelete(timeoutEvent);
}
```

```cpp
void Tic8::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 Tic8::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 Toc8 : public cSimpleModule
{
  protected:
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Toc8);
void Toc8::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");
    }
}
```

```
// ticctoc9.ned
simple Tic9
{
    parameters:
        @display("i=block/routing");
    gates:
        input in;
        output out;
}

simple Toc9
{
    parameters:
        @display("i=block/process");
    gates:
        input in;
        output out;
}

//
// Same as Tictoc8.
//
network Tictoc9
{
    submodules:
        tic: Tic9 {
            parameters:
                @display("i=,cyan");
        }
        toc: Toc9 {
            parameters:
                @display("i=,gold");
        }
    connections:
        tic.out --> {  delay = 100ms; } --> toc.in;
        tic.in <-- {  delay = 100ms; } <-- toc.out;
}
```

```cpp
// txc9.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * In the previous model we just created another packet if we needed to
 * retransmit. This is OK because the packet didn't contain much, but
 * in real life it's usually more practical to keep a copy of the original
 * packet so that we can re-send it without the need to build it again.
 */
// 메시지의 카피를 저장했다가 다시 보내는게 재생성해서 다시보내는 것 보다 효율.
class Tic9 : 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:
    Tic9();
    virtual ~Tic9();
  protected:
    virtual cMessage *generateNewMessage();
    virtual void sendCopyOf(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Tic9);
Tic9::Tic9()
{
    timeoutEvent = message = NULL;
}
Tic9::~Tic9()
{
    cancelAndDelete(timeoutEvent);
    delete message;
}
```

```cpp
void Tic9::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 Tic9::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";
        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 *Tic9::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 Tic9::sendCopyOf(cMessage *msg)
{
    // Duplicate message and send the copy.
    cMessage *copy = (cMessage *) msg->dup();
    send(copy, "out");
}
/**
 * Sends back an acknowledgement -- or not.
 */
class Toc9 : public cSimpleModule
{
  protected:
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Toc9);
void Toc9::handleMessage(cMessage *msg)
{
    if (uniform(0,1) < 0.1)
    {
        EV << "\"Losing\" message " << msg << endl;
        bubble("message lost");
        delete msg;
    }
    else
    {
        EV << msg << " received, sending back an acknowledgement.\n";
        delete msg;
        send(new cMessage("ack"), "out");
    }
}
```

```
// tictoc10.ned
simple Txc10
{
    parameters:
        @display("i=block/routing");
    gates:
       // 벡터를 이용한 게이트 정
        input in[];  // declare in[] and out[] to be vector gates
        output out[];
}
network Tictoc10
{
    submodules:
        tic[6]: Txc10;
    connections:
        tic[0].out++ --> {  delay = 100ms; } --> tic[1].in++;
        tic[0].in++ <-- {  delay = 100ms; } <-- tic[1].out++;
        tic[1].out++ --> {  delay = 100ms; } --> tic[2].in++;
        tic[1].in++ <-- {  delay = 100ms; } <-- tic[2].out++;
        tic[1].out++ --> {  delay = 100ms; } --> tic[4].in++;
        tic[1].in++ <-- {  delay = 100ms; } <-- tic[4].out++;
        tic[3].out++ --> {  delay = 100ms; } --> tic[4].in++;
        tic[3].in++ <-- {  delay = 100ms; } <-- tic[4].out++;
        tic[4].out++ --> {  delay = 100ms; } --> tic[5].in++;
        tic[4].in++ <-- {  delay = 100ms; } <-- tic[5].out++;
}
```

```
// txc10.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * Let's make it more interesting by using several (n) `tic' modules,
 * and connecting every module to every other. For now, let's keep it
 * simple what they do: module 0 generates a message, and the others
 * keep tossing it around in random directions until it arrives at
 * module 2.
 */
class Txc10 : public cSimpleModule
{
  protected:
    virtual void forwardMessage(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc10);
void Txc10::initialize()
{
    if (getIndex()==0) // 인덱스가 있으면.
    {
        // Boot the process scheduling the initial message as a self-message.
        char msgname[20];
        sprintf(msgname, "tic-%d", getIndex());
        cMessage *msg = new cMessage(msgname);
        scheduleAt(0.0, msg);
    }
}
```

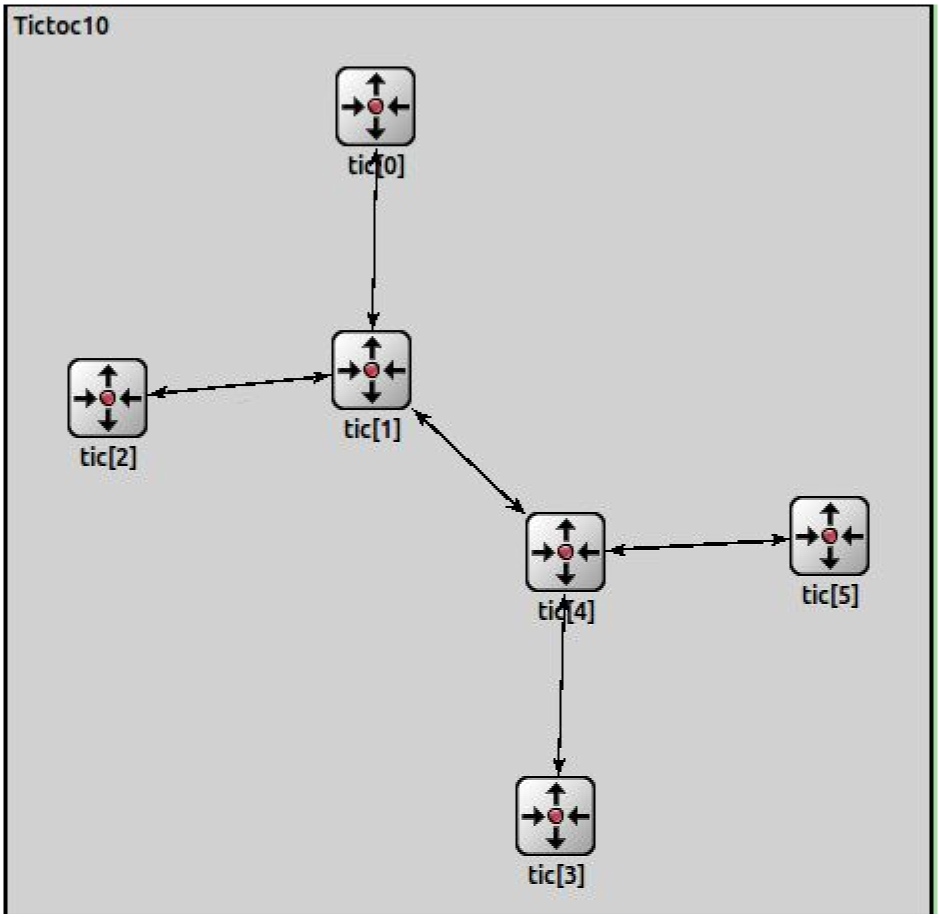
```
// tictoc11.ned
simple Txc11
{
    parameters:
        @display("i=block/routing");
    gates:
        input in[];  // declare in[] and out[] to be vector gates
        output out[];
}
//
// Using local channel type definition to reduce the redundancy
// of connection definitions.
//
network Tictoc11
{
    types:
        channel Channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
        tic[6]: Txc11;
    connections:
        tic[0].out++ --> Channel --> tic[1].in++;
        tic[0].in++ <-- Channel <-- tic[1].out++;
        tic[1].out++ --> Channel --> tic[2].in++;
        tic[1].in++ <-- Channel <-- tic[2].out++;
        tic[1].out++ --> Channel --> tic[4].in++;
        tic[1].in++ <-- Channel <-- tic[4].out++;
        tic[3].out++ --> Channel --> tic[4].in++;
        tic[3].in++ <-- Channel <-- tic[4].out++;
        tic[4].out++ --> Channel --> tic[5].in++;
        tic[4].in++ <-- Channel <-- tic[5].out++;
}
```

```
// txc11.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * Let's make it more interesting by using several (n) `tic' modules,
 * and connecting every module to every other. For now, let's keep it
 * simple what they do: module 0 generates a message, and the others
 * keep tossing it around in random directions until it arrives at
 * module 2.
 */
class Txc11 : public cSimpleModule
{
  protected:
    virtual void forwardMessage(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc11);
void Txc11::initialize()
{
    if (getIndex()==0)
    {
        // Boot the process scheduling the initial message as a self-message.
        char msgname[20];
        sprintf(msgname, "tic-%d", getIndex());
        cMessage *msg = new cMessage(msgname);
        scheduleAt(0.0, msg);
    }
}
```

```
void Txc11::handleMessage(cMessage *msg)
{
    if (getIndex()==3)
    {
        // Message arrived.
        EV << "Message " << msg << " arrived.\n";
        delete msg;
    }
    else
    {
        // We need to forward the message.
        forwardMessage(msg);
    }
}
void Txc11::forwardMessage(cMessage *msg)
{
    // In this example, we just pick a random gate to send it on.
    // We draw a random number between 0 and the size of gate `out[]'.
    int n = gateSize("out"); // gate의 갯수를 받는다.
    int k = intuniform(0,n-1); // random 으로 보내
    EV << "Forwarding message " << msg << " on port out[" << k << "]\n";
    send(msg, "out", k);
}
```

```cpp
void Txc10::handleMessage(cMessage *msg)
{
    if (getIndex()==3)
    {
        // Message arrived.
        EV << "Message " << msg << " arrived.\n";
        delete msg;
    }
    else
    {
        // We need to forward the message.
        forwardMessage(msg);
    }
}
void Txc10::forwardMessage(cMessage *msg)
{
    // In this example, we just pick a random gate to send it on.
    // We draw a random number between 0 and the size of gate `out[]'.
    int n = gateSize("out");
    int k = intuniform(0,n-1);

    EV << "Forwarding message " << msg << " on port out[" << k << "]\n";
    send(msg, "out", k);
}
```


Tictoc10
tic[0]
tic[1]
tic[2]
tic[3]
tic[4]
tic[5]

```
// tictoc12.ned
// channel 개념의 사용.
simple Txc12
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];  // declare two way connections
}
// using two way connections to further simplify the network definition
network Tictoc12
{
    types:
        channel Channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
        tic[6]: Txc12;
    connections:
        tic[0].gate++ <--> Channel <--> tic[1].gate++;
        tic[1].gate++ <--> Channel <--> tic[2].gate++;
        tic[1].gate++ <--> Channel <--> tic[4].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
}
```

```
// txc.12.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
/**
 * Let's make it more interesting by using several (n) `tic' modules,
 * and connecting every module to every other. For now, let's keep it
 * simple what they do: module 0 generates a message, and the others
 * keep tossing it around in random directions until it arrives at
 * module 2.
 */
class Txc12 : public cSimpleModule
{
  protected:
    virtual void forwardMessage(cMessage *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc12);
void Txc12::initialize()
{
    if (getIndex()==0)
    {
        // Boot the process scheduling the initial message as a self-message.
        char msgname[20];
        sprintf(msgname, "tic-%d", getIndex());
        cMessage *msg = new cMessage(msgname);
        scheduleAt(0.0, msg);
    }
}
```

```
void Txc12::handleMessage(cMessage *msg)
{
    if (getIndex()==3)
    {
        // Message arrived.
        EV << "Message " << msg << " arrived.\n";
        delete msg;
    }
    else
    {
        // We need to forward the message.
        forwardMessage(msg);
    }
}
void Txc12::forwardMessage(cMessage *msg)
{
    // In this example, we just pick a random gate to send it on.
    // We draw a random number between 0 and the size of gate `gate[]'.
    int n = gateSize("gate");
    int k = intuniform(0,n-1);
    EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
    // $o and $i suffix is used to identify the input/output part of a two way gate
    send(msg, "gate$o", k); // gateName의 변화.
}
```

```
// tictoc13.ned
simple Txc13
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

//
// Same as Tictoc12
//
network Tictoc13
{
    types:
        channel Channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
        tic[6]: Txc13;
    connections:
        tic[0].gate++ <--> Channel <--> tic[1].gate++;
        tic[1].gate++ <--> Channel <--> tic[2].gate++;
        tic[1].gate++ <--> Channel <--> tic[4].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
}
```

```
/*
 *tictoc13.msg
 *message TicTocMsg13
{
    int source;
    int destination;
    int hopCount = 0;
}
 * */
// txc13.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
// Include a generated file: the header file created from tictoc13.msg.
// It contains the definition of the TictocMsg10 class, derived from
// cMessage.
#include "tictoc13_m.h"
/**
 * In this step the destination address is no longer node 2 -- we draw a
 * random destination, and we'll add the destination address to the message.
 *
 * The best way is to subclass cMessage and add destination as a data member.
 * Hand-coding the message class is usually tiresome because it contains
 * a lot of boilerplate code, so we let OMNeT++ generate the class for us.
 * The message class specification is in tictoc13.msg -- tictoc13_m.h
 * and .cc will be generated from this file automatically.
 *
 * To make the model execute longer, after a message arrives to its destination
 * the destination node will generate another message with a random destination
 * address, and so forth.
 */
// 메세지에 관련된 클래스들은 tictoc13.msg/tictoc13_m.h에 정의되어있다.
// 그리고 메세지.cc는 이것들로 부터 메세지를 생성할 것 이다.
```

```
class Txc13 : public cSimpleModule
{
  protected:
    virtual TicTocMsg13 *generateMessage(); // tictoc12_m.h에 정의되어있음.
    // TicTocMsg13형의 메세지를 생성.
    virtual void forwardMessage(TicTocMsg13 *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc13);
void Txc13::initialize()
{
    // Module 0 sends the first message
    if (getIndex() == 0)
    {
        // Boot the process scheduling the initial message as a self-message.
        TicTocMsg13 *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}
void Txc13::handleMessage(cMessage *msg)
{
    // dynamic assign정도로만 알아두자.
    TicTocMsg13 *ttmsg = check_and_cast<TicTocMsg13 *>(msg);
    if (ttmsg->getDestination()==getIndex()) // return: destination var
    {
        // Message arrived.
        EV << "Message " << ttmsg << " arrived after " << ttmsg->getHopCount() << " hops.\n";
        bubble("ARRIVED, starting new one!");
        delete ttmsg;
        // Generate another one.
        EV << "Generating another message: ";
        TicTocMsg13 *newmsg = generateMessage();
        EV << newmsg << endl;
        forwardMessage(newmsg);
    }
    else
    {
        // We need to forward the message.
        forwardMessage(ttmsg);
    }
}
```

```
TicTocMsg13 *Txc13::generateMessage()
{
    // Produce source and destination addresses.
    int src = getIndex();   // our module index
    int n = size();      // module vector size
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;
    char msgname[20];
    sprintf(msgname, "tic-%d-to-%d", src, dest);
    // Create message object and set source and destination field.
    TicTocMsg13 *msg = new TicTocMsg13(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}
void Txc13::forwardMessage(TicTocMsg13 *msg)
{
    // Increment hop count.
    msg->setHopCount(msg->getHopCount()+1);
    // Same routing as before: random gate.
    int n = gateSize("gate");
    int k = intuniform(0,n-1);
    EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
    send(msg, "gate$o", k);
}
```

```
// tictoc14.ned
simple Txc14
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

//
// Same as Tictoc12
//
network Tictoc14
{
    types:
        channel Channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
        tic[6]: Txc14;
    connections:
        tic[0].gate++ <--> Channel <--> tic[1].gate++;
        tic[1].gate++ <--> Channel <--> tic[2].gate++;
        tic[1].gate++ <--> Channel <--> tic[4].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
}
```

```cpp
// txc14.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "tictoc14_m.h"
/**
 * In this step we keep track of how many messages we send and received,
 * and display it above the icon.
 */
// 메세지 트래킹.
class Txc14 : public cSimpleModule
{
  private:
    long numSent;
    long numReceived;
  protected:
    virtual TicTocMsg14 *generateMessage();
    virtual void forwardMessage(TicTocMsg14 *msg);
    virtual void updateDisplay();
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc14);
void Txc14::initialize()
{
    // Initialize variables
    numSent = 0;
    numReceived = 0;
    WATCH(numSent);
    WATCH(numReceived);
    // Module 0 sends the first message
    if (getIndex()==0)
    {
        // Boot the process scheduling the initial message as a self-message.
        TicTocMsg14 *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}
```

```cpp
void Txc14::handleMessage(cMessage *msg)
{
    TicTocMsg14 *ttmsg = check_and_cast<TicTocMsg14 *>(msg);
    if (ttmsg->getDestination()==getIndex())
    {
        // Message arrived
        int hopcount = ttmsg->getHopCount();
        EV << "Message " << ttmsg << " arrived after " << hopcount << " hops.\n";
        numReceived++;
        delete ttmsg;
        bubble("ARRIVED, starting new one!");
        // Generate another one.
        EV << "Generating another message: ";
        TicTocMsg14 *newmsg = generateMessage();
        EV << newmsg << endl;
        forwardMessage(newmsg);
        numSent++;
        if (ev.isGUI()) // gui설정이면 1 return.
            updateDisplay();
    }
    else
    {
        // We need to forward the message.
        forwardMessage(ttmsg);
    }
}
TicTocMsg14 *Txc14::generateMessage()
{
    // Produce source and destination addresses.
    int src = getIndex();   // our module index
    int n = size();      // module vector size
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;
    char msgname[20];
    sprintf(msgname, "tic-%d-to-%d", src, dest);
    // Create message object and set source and destination field.
    TicTocMsg14 *msg = new TicTocMsg14(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}
```

```
void Txc14::forwardMessage(TicTocMsg14 *msg)
{
    // Increment hop count.
    msg->setHopCount(msg->getHopCount()+1);
    // Same routing as before: random gate.
    int n = gateSize("gate");
    int k = intuniform(0,n-1);
    EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
    send(msg, "gate$o", k);
}
void Txc14::updateDisplay()
{
    char buf[40];
    sprintf(buf, "rcvd: %ld sent: %ld", numReceived, numSent);
    getDisplayString().setTagArg("t",0,buf);
}
```

```
// tictoc15.ned
simple Txc15
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

//
// Same as Tictoc12
//
network Tictoc15
{
    types:
        channel Channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
        tic[6]: Txc15;
    connections:
        tic[0].gate++ <--> Channel <--> tic[1].gate++;
        tic[1].gate++ <--> Channel <--> tic[2].gate++;
        tic[1].gate++ <--> Channel <--> tic[4].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
}
```

```cpp
// txc15.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "tictoc15_m.h"
/**
 * This model is exciting enough so that we can collect some statistics.
 * We'll record in output vectors the hop count of every message upon arrival.
 * Output vectors are written into the omnetpp.vec file and can be visualized
 * with the Plove program.
 *
 * We also collect basic statistics (min, max, mean, std.dev.) and histogram
 * about the hop count which we'll print out at the end of the simulation.
 */
// 통계를 추출하여 다른 output vector에 저장. (result)
class Txc15 : public cSimpleModule
{
  private:
    long numSent;
    long numReceived;
    cLongHistogram hopCountStats;
    cOutVector hopCountVector;
  protected:
    virtual TicTocMsg15 *generateMessage();
    virtual void forwardMessage(TicTocMsg15 *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    // The finish() function is called by OMNeT++ at the end of the simulation:
    virtual void finish();
};
Define_Module(Txc15);
```

```
void Txc15::initialize()
{
    // Initialize variables
    numSent = 0;
    numReceived = 0;
    WATCH(numSent);
    WATCH(numReceived);
    hopCountStats.setName("hopCountStats");
    hopCountStats.setRangeAutoUpper(0, 10, 1.5);
    hopCountVector.setName("HopCount");
    // Module 0 sends the first message
    if (getIndex()==0)
    {
        // Boot the process scheduling the initial message as a self-message.
        TicTocMsg15 *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}
void Txc15::handleMessage(cMessage *msg)
{
    TicTocMsg15 *ttmsg = check_and_cast<TicTocMsg15 *>(msg);

    if (ttmsg->getDestination()==getIndex())
    {
        // Message arrived
        int hopcount = ttmsg->getHopCount();
        EV << "Message " << ttmsg << " arrived after " << hopcount << " hops.\n";
        bubble("ARRIVED, starting new one!");
        // update statistics.
        numReceived++;
        hopCountVector.record(hopcount);
        hopCountStats.collect(hopcount);
        delete ttmsg;
        // Generate another one.
        EV << "Generating another message: ";
        TicTocMsg15 *newmsg = generateMessage();
        EV << newmsg << endl;
        forwardMessage(newmsg);
        numSent++;
    }
```

```
    else
    {
        // We need to forward the message.
        forwardMessage(ttmsg);
    }
}
TicTocMsg15 *Txc15::generateMessage()
{
    // Produce source and destination addresses.
    int src = getIndex();
    int n = size();
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;
    char msgname[20];
    sprintf(msgname, "tic-%d-to-%d", src, dest);
    // Create message object and set source and destination field.
    TicTocMsg15 *msg = new TicTocMsg15(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}
void Txc15::forwardMessage(TicTocMsg15 *msg)
{
    // Increment hop count.
    msg->setHopCount(msg->getHopCount()+1);
    // Same routing as before: random gate.
    int n = gateSize("gate");
    int k = intuniform(0,n-1);
    EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
    send(msg, "gate$o", k);
}
```

```
void Txc15::finish()
{
    // This function is called by OMNeT++ at the end of the simulation.
    EV << "Sent:     " << numSent << endl;
    EV << "Received: " << numReceived << endl;
    EV << "Hop count, min:    " << hopCountStats.getMin() << endl;
    EV << "Hop count, max:    " << hopCountStats.getMax() << endl;
    EV << "Hop count, mean:   " << hopCountStats.getMean() << endl;
    EV << "Hop count, stddev: " << hopCountStats.getStddev() << endl;
    recordScalar("#sent", numSent);
    recordScalar("#received", numReceived);
    hopCountStats.recordAs("hop count");
}
```

```
// tictioc16.ned
simple Txc16
{
    parameters:
        @signal[arrival](type="long");
        @statistic[hopCount](title="hop count"; source="arrival"; record=vector,stats; interpolationmode=none);
        @display("i=block/routing");
    gates:
        inout gate[];
}

//
// Same as Tictoc12
//
network Tictoc16
{
    types:
        channel Channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
        tic[6]: Txc16;
    connections:
        tic[0].gate++ <--> Channel <--> tic[1].gate++;
        tic[1].gate++ <--> Channel <--> tic[2].gate++;
        tic[1].gate++ <--> Channel <--> tic[4].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
}
```

```
// txc16.cc
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "tictoc16_m.h"
/**
 * The main problem with the previous step is that we must modify the model's
 * code if we want to change what statistics are gathered. Statistic calculation
 * is woven deeply into the model code which is hard to modify and understand.
 *
 * OMNeT++ 4.1 provides a different mechanism called 'signals' that we can use
 * to gather statistics. First we have to identify the events where the state
 * of the model changes. We can emit signals at these points that carry the value
 * of chosen state variables. This way the C++ code only emits signals, but how those
 * signals are processed are determined only by the listeners that are attached to them.
 *
 * The signals the model emits and the listeners that process them can be defined in
 * the NED file using the 'signal' and 'statistic' property.
 *
 * We will gather the same statistics as in the previous step, but notice that we will not need
 * any private member variables to calculate these values. We will use only a single signal that
 * is emitted when a message arrives and carries the hopcount in the message.
 */
class Txc16 : public cSimpleModule
{
  private:
    simsignal_t arrivalSignal;

  protected:
    virtual TicTocMsg16 *generateMessage();
    virtual void forwardMessage(TicTocMsg16 *msg);
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};
Define_Module(Txc16);
void Txc16::initialize()
{
    arrivalSignal = registerSignal("arrival");
    // Module 0 sends the first message
    if (getIndex()==0)
    {
        // Boot the process scheduling the initial message as a self-message.
        TicTocMsg16 *msg = generateMessage();
        scheduleAt(0.0, msg);
    }
}
```

```cpp
void Txc16::handleMessage(cMessage *msg)
{
    TicTocMsg16 *ttmsg = check_and_cast<TicTocMsg16 *>(msg);

    if (ttmsg->getDestination()==getIndex())
    {
        // Message arrived
        int hopcount = ttmsg->getHopCount();
        // send a signal
        emit(arrivalSignal, hopcount);
        EV << "Message " << ttmsg << " arrived after " << hopcount << " hops.\n";
        bubble("ARRIVED, starting new one!");
        delete ttmsg;
        // Generate another one.
        EV << "Generating another message: ";
        TicTocMsg16 *newmsg = generateMessage();
        EV << newmsg << endl;
        forwardMessage(newmsg);
    }
    else
    {
        // We need to forward the message.
        forwardMessage(ttmsg);
    }
}
TicTocMsg16 *Txc16::generateMessage()
{
    // Produce source and destination addresses.
    int src = getIndex();
    int n = size();
    int dest = intuniform(0,n-2);
    if (dest>=src) dest++;
    char msgname[20];
    sprintf(msgname, "tic-%d-to-%d", src, dest);
    // Create message object and set source and destination field.
    TicTocMsg16 *msg = new TicTocMsg16(msgname);
    msg->setSource(src);
    msg->setDestination(dest);
    return msg;
}
void Txc16::forwardMessage(TicTocMsg16 *msg)
{
    // Increment hop count.
    msg->setHopCount(msg->getHopCount()+1);
    // Same routing as before: random gate.
    int n = gateSize("gate");
    int k = intuniform(0,n-1);
    EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
    send(msg, "gate$o", k);
}
```