# LAB 03

## Nội dung

# Part 4,5,6 Tictoc tutorial

## Part 4 - Turning it Into a Real Network

### 4.1 More than two nodes

file tictoc10.ned

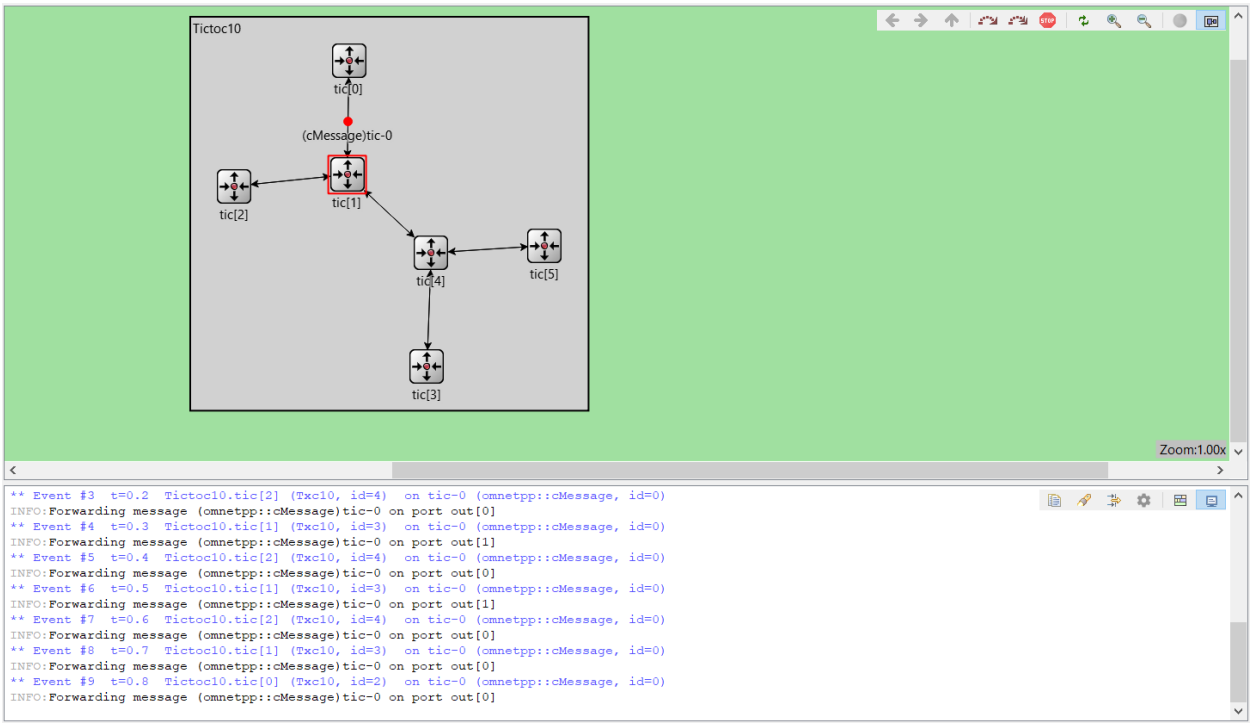
```
simple Txc10
{
    parameters:
        @display("i=block/routing");
    gates:
        input in[];  // declare in[] and out[] to be vector gates
        output out[];
}

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++;
}
```

file txc10.cc

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

using namespace omnetpp;


class Txc10 : public cSimpleModule
{
  protected:
    virtual void forwardMessage(cMessage *msg);
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Txc10);
```

```
void Txc10::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 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);
}
```

result




## 4.2 Channels and inner type definitions

file 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++;
}
```

### 4.3 Using two-way connections

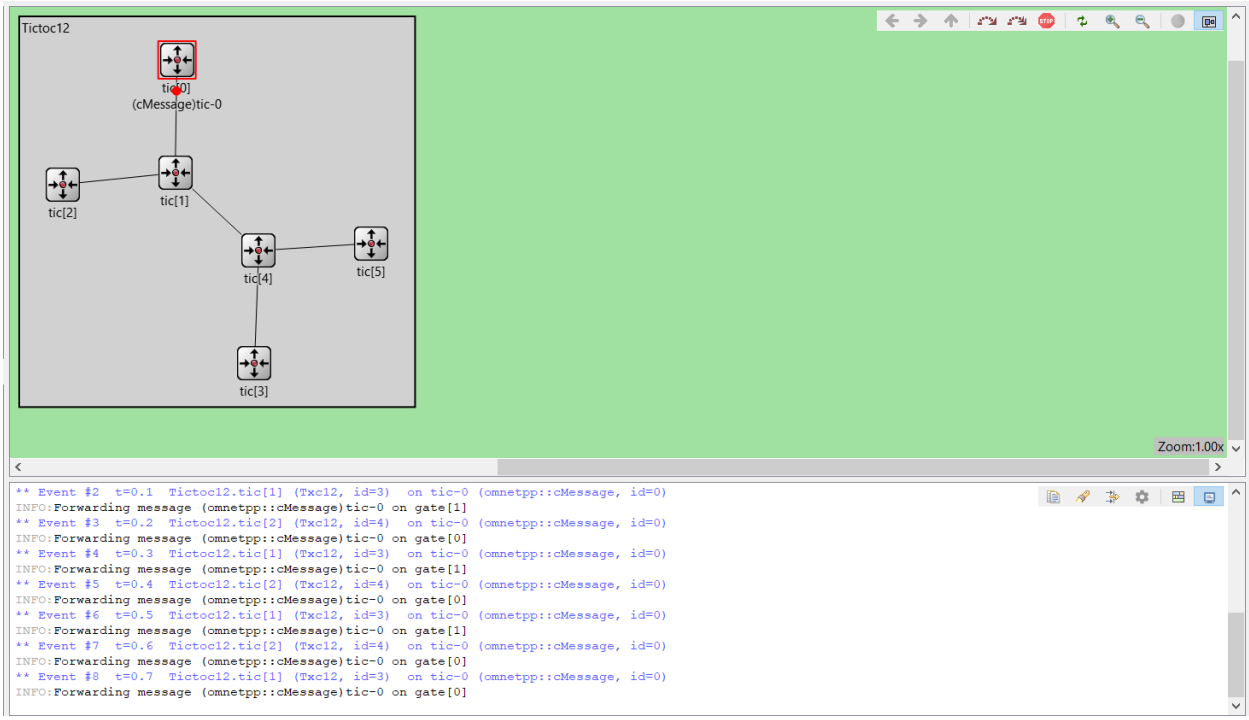
tictoc12.ned

```
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++;
}
```

txc12.cc

```
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);
}
```

result




## 4.4 Defining our message class

### tictoc13.ned

giống tictoc12.ned

### txc13.cc

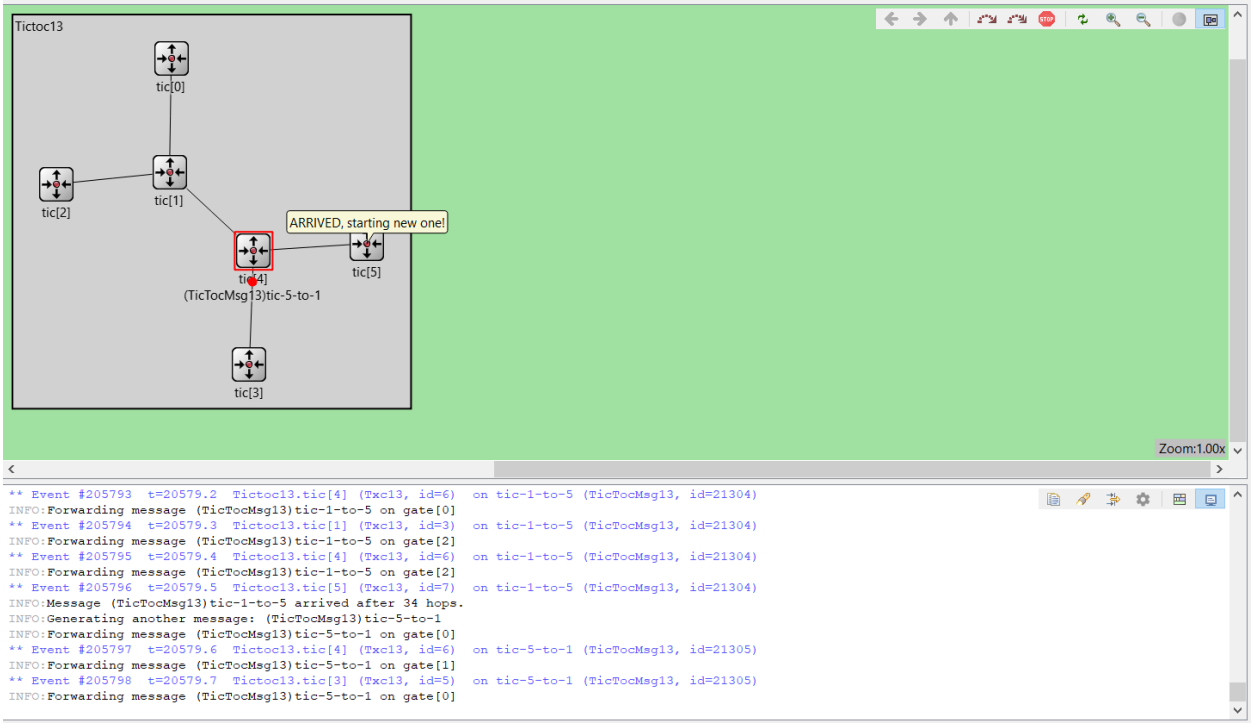
```
class Txc13 : public cSimpleModule
{
  protected:
    virtual TicTocMsg13 *generateMessage();
    virtual void forwardMessage(TicTocMsg13 *msg);
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

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)
{
    TicTocMsg13 *ttmsg = check_and_cast<TicTocMsg13 *>(msg);

    if (ttmsg->getDestination() == getIndex()) {
        // 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 = getVectorSize();  // 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);
}
```

tictoc13.msg

```
message TicTocMsg13
{
    int source;
    int destination;
    int hopCount = 0;
}
```

result




# Part 5 - Adding Statistics Collection

## 5.1 Displaying the number of packets sent/received

### tictoc14.ned

giống tictoc13.ned

### txc14.cc

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "tictoc14_m.h"

using namespace omnetpp;

/**
 * In this step we keep track of how many messages we send and received,
 * and display it above the icon.
 */
class Txc14 : public cSimpleModule
{
  private:
    long numSent;
    long numReceived;

  protected:
    virtual TicTocMsg14 *generateMessage();
    virtual void forwardMessage(TicTocMsg14 *msg);
    virtual void refreshDisplay() const override;

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

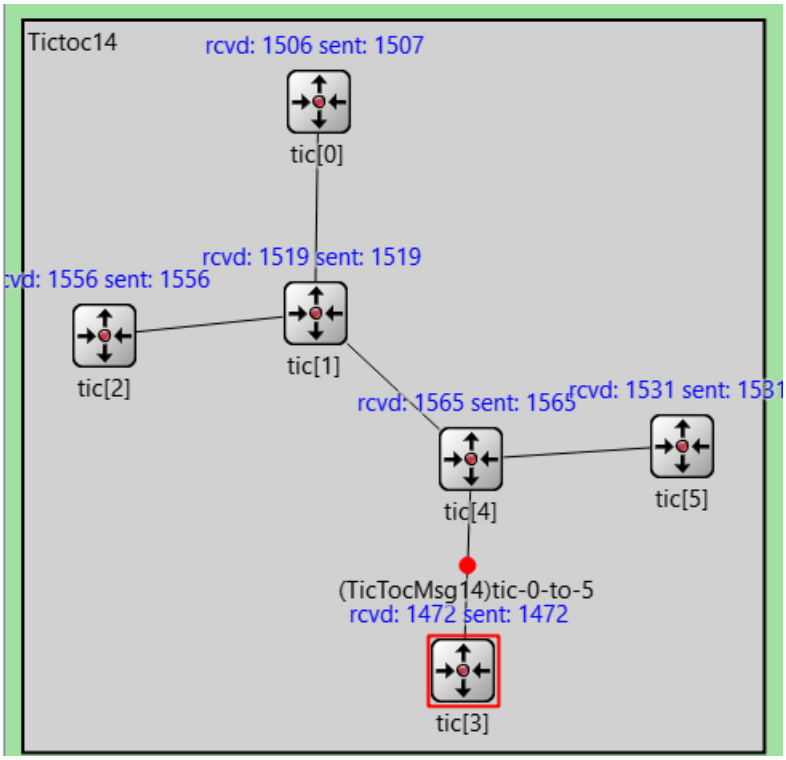
```
Define_Module(Txc14);

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();
        numSent++;
        scheduleAt(0.0, msg);
    }
}

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++;
    }
    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 = getVectorSize();  // 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::refreshDisplay() const
{
    char buf[40];
    sprintf(buf, "rcvd: %ld sent: %ld", numReceived, numSent);
    getDisplayString().setTagArg("t", 0, buf);
}
```

tictoc14.msg

giống tictoc13.msg

result:




## 5.2 Adding statistics collection

tictoc15.ned

không đổi

tictoc15.msg

không đổi

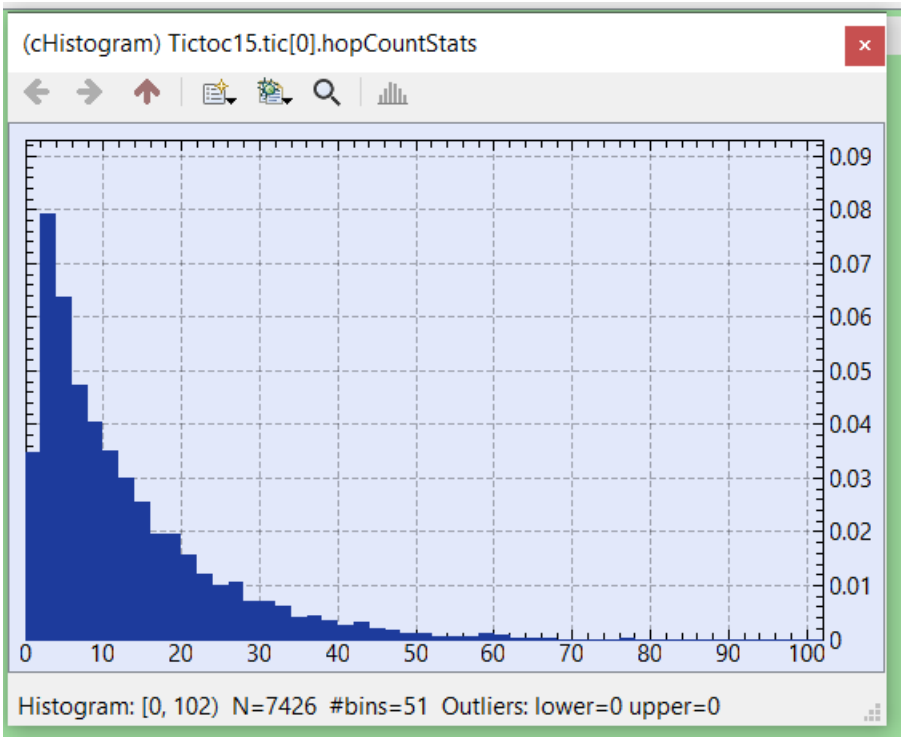
txc15.cc

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "tictoc15_m.h"

using namespace omnetpp;

class Txc15 : public cSimpleModule
{
  private:
    long numSent;
    long numReceived;
    cHistogram hopCountStats;
    cOutVector hopCountVector;

  protected:
    virtual TicTocMsg15 *generateMessage();
    virtual void forwardMessage(TicTocMsg15 *msg);
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;

    // The finish() function is called by OMNeT++ at the end of the simulation:
    virtual void finish() override;
};

Define_Module(Txc15);

void Txc15::initialize()
{
    // Initialize variables
    numSent = 0;
    numReceived = 0;
    WATCH(numSent);
    WATCH(numReceived);

    hopCountStats.setName("hopCountStats");
    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 = getVectorSize();
    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");
}
```

result:




## 5.3 Statistic collection without modifying your model

tictoc16.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[];
}

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++;
}
```

tictoc16.msg

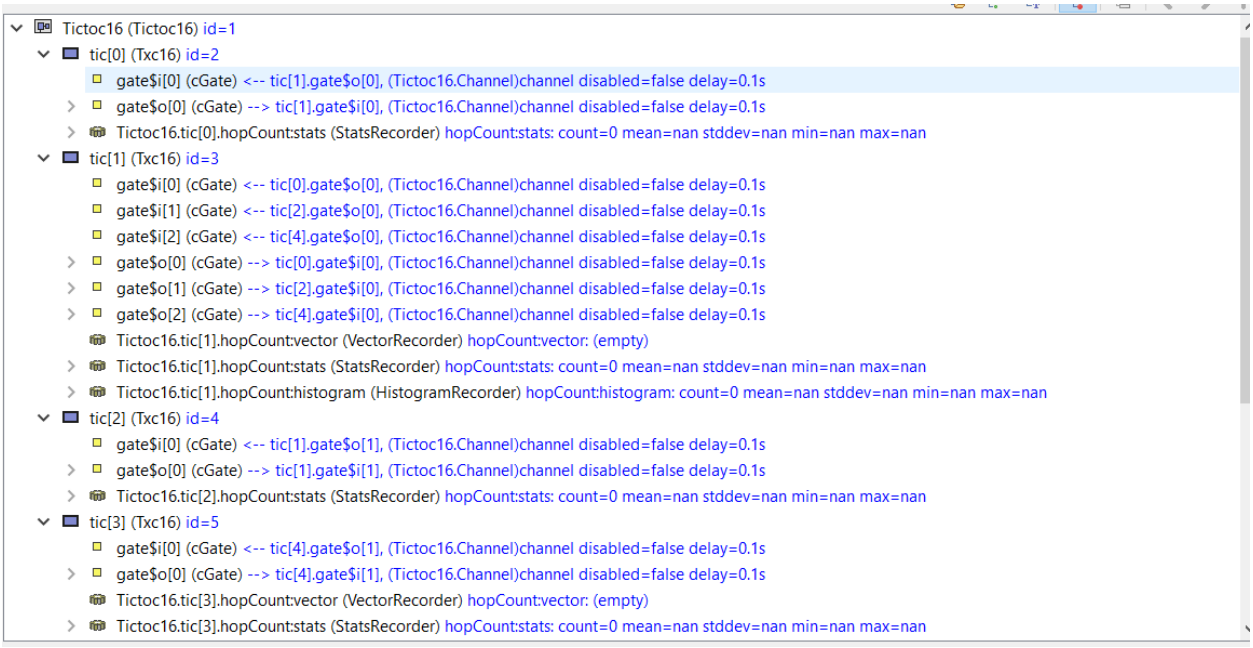
không đổi

txc16.cc

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "tictoc16_m.h"

using namespace omnetpp;

class Txc16 : public cSimpleModule
{
  private:
    simsignal_t arrivalSignal;

  protected:
    virtual TicTocMsg16 *generateMessage();
    virtual void forwardMessage(TicTocMsg16 *msg);
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

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);
    }
}

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 = getVectorSize();
    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);
}
```

result




## 5.4 Adding figures

tictoc17.ned

```
simple Txc17
{
    parameters:
        @signal[arrival](type="long");
        @statistic[hopCount](title="hop count"; source="arrival"; record=vector,stats;
interpolationmode=none);

        @display("i=block/routing");
    gates:
        inout gate[];
}

network Tictoc17
{
    parameters:
        @figure[description](type=text; pos=5,20; font=,,bold;
                text="Random routing example - displaying last hop count");
        @figure[lasthopcount](type=text; pos=5,35; text="last hopCount: N/A");
    types:
        channel Channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
        tic[6]: Txc17;
    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++;
}
```

txc17.cc

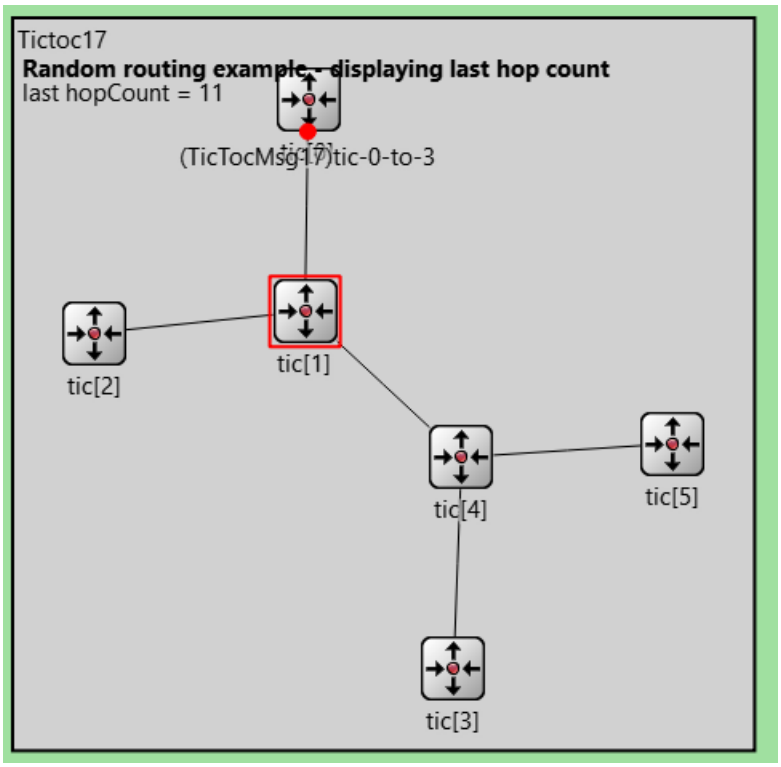
thêm một đoạn code vào hàm:

```
void Txc17::handleMessage(cMessage *msg)
{
    TicTocMsg17 *ttmsg = check_and_cast<TicTocMsg17 *>(msg);

    if (ttmsg->getDestination() == getIndex()) {
        // Message arrived
        int hopcount = ttmsg->getHopCount();
        // send a signal
        emit(arrivalSignal, hopcount);

        if (hasGUI()) {
            char label[50];
            // Write last hop count to string
            sprintf(label, "last hopCount = %d", hopcount);
            // Get pointer to figure
            cCanvas *canvas = getParentModule()->getCanvas();
            cTextFigure *textFigure = check_and_cast<cTextFigure*>(canvas-
>getFigure("lasthopcount"));
            // Update figure text
            textFigure->setText(label);
        }

        EV << "Message " << ttmsg << " arrived after " << hopcount << " hops.\n";
        bubble("ARRIVED, starting new one!");

        delete ttmsg;

        // Generate another one.
        EV << "Generating another message: ";
        TicTocMsg17 *newmsg = generateMessage();
        EV << newmsg << endl;
        forwardMessage(newmsg);
    }
    else {
        // We need to forward the message.
        forwardMessage(ttmsg);
    }
}
```

tictoc17.msg

không đổi

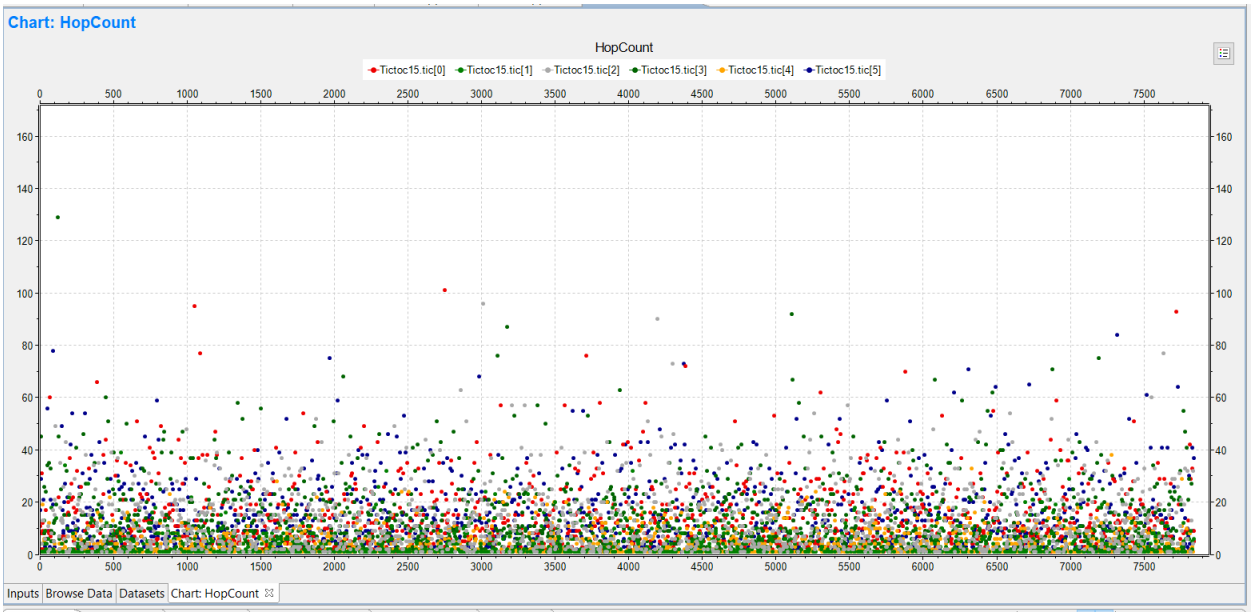
result:




# Part 6 - Visualizing the Results With the IDE

## 6.1 Visualizing output scalars and vectors

result:



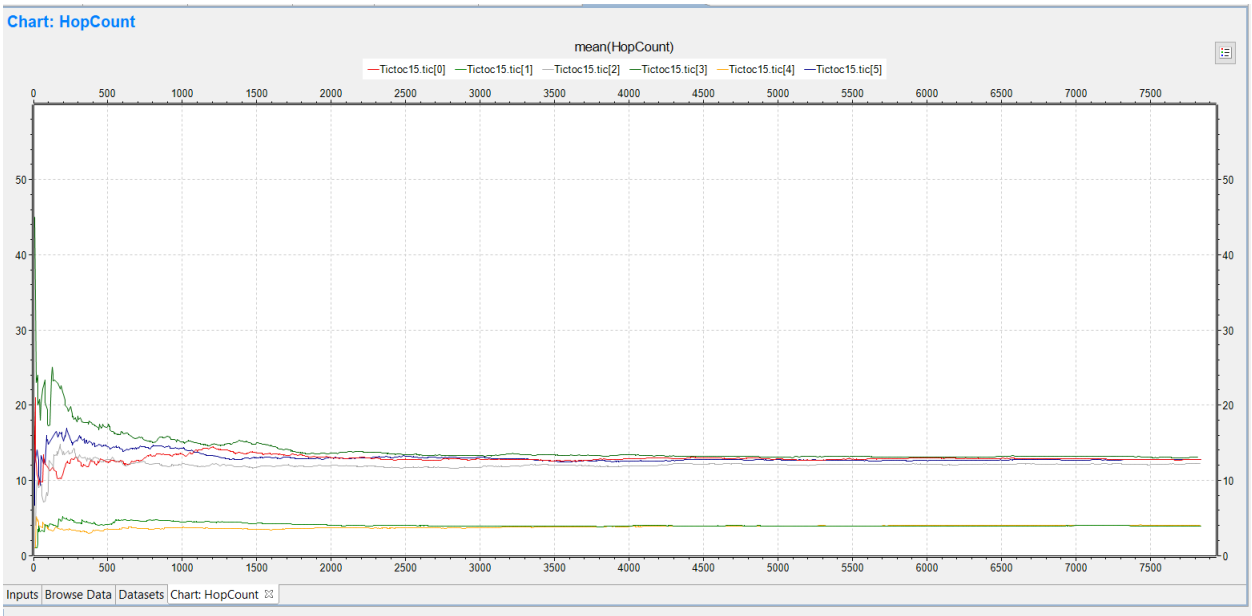


Chart: HopCount
mean(HopCount)
Tictoc15.tic[0]
Tictoc15.tic[1]
Tictoc15.tic[2]
Tictoc15.tic[3]
Tictoc15.tic[4]
Tictoc15.tic[5]
Inputs
Browse Data
Datasets
Chart: HopCount

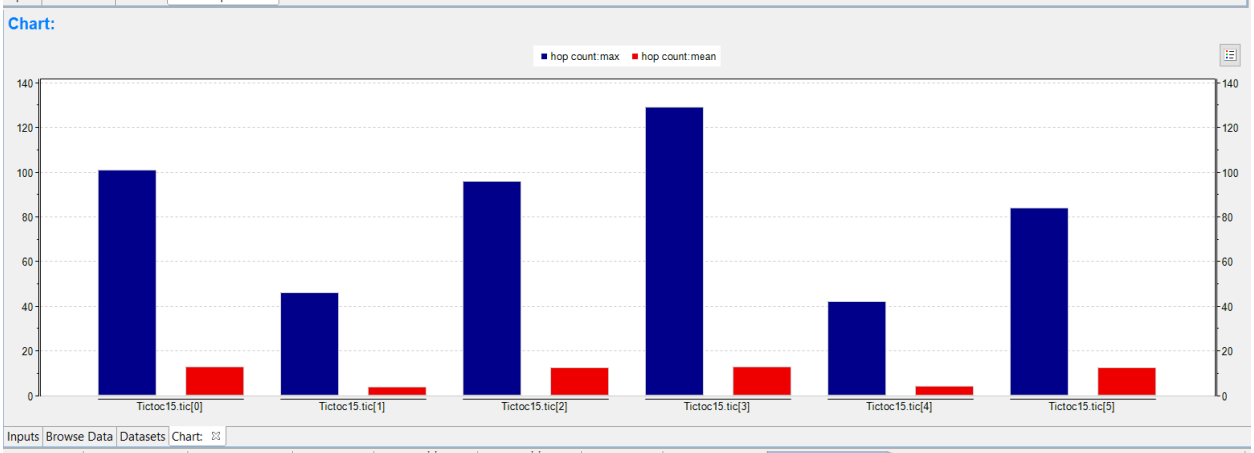


Chart:
hop count:max
hop count:mean
Tictoc15.tic[0]
Tictoc15.tic[1]
Tictoc15.tic[2]
Tictoc15.tic[3]
Tictoc15.tic[4]
Tictoc15.tic[5]
Inputs
Browse Data
Datasets
Chart:

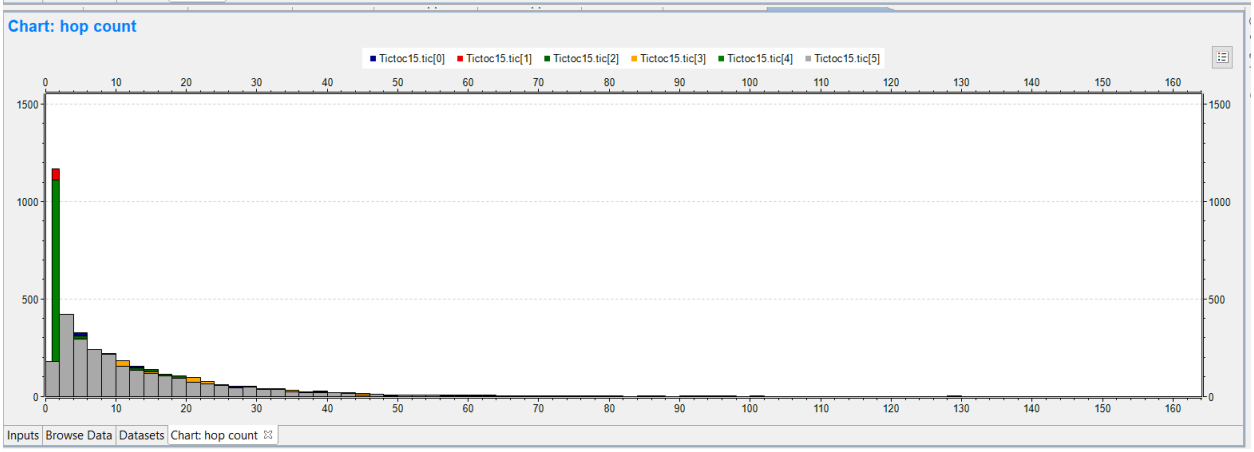


Chart: hop count
Tictoc15.tic[0]
Tictoc15.tic[1]
Tictoc15.tic[2]
Tictoc15.tic[3]
Tictoc15.tic[4]
Tictoc15.tic[5]
Inputs
Browse Data
Datasets
Chart: hop count

# LAB 03A

## 1. Giải thích ý nghĩa của các dòng mã nguồn dưới đây:

```
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);
}
```

Hàm này có tác dụng chuyển tiếp một thông điệp tới nút khác.

Cụ thể:
Tham số đầu vào là một thông điệp.
Trước tiên nó xác định xem mô-đun hiện tại là mô-đun bậc mấy (tức có kết nối với bao nhiêu mô-đun khác, hay nói cách khác là có bao nhiêu cổng ra).

```
        int n = gateSize("out");
```

Sau đó nó chọn ra ngẫu nhiên (theo phân phối đều) một nút trong số các nút kề:

```
        int k = intuniform(0, n-1);
```

Log ra màn hình rồi chuyển thông điệp đi:

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

## 2. Giải thích ý nghĩa của các dòng mã nguồn dưới đây:

```
if (hasGUI()) {
      char label[50];
      // Write last hop count to string
      sprintf(label, "last hopCount = %d", hopcount);
      // Get pointer to figure
      cCanvas *canvas = getParentModule()->getCanvas();
      cTextFigure *textFigure =
            check_and_cast<cTextFigure*>(canvas->getFigure("lasthopcount"));
      // Update figure text
      textFigure->setText(label);
```

Đoạn mã trên có tác dụng cập nhật lại thông tin về hopCount được hiển thị trên màn hình.

Cụ thể:

Đoạn mã này chỉ được thực hiện khi môi trường hiện tại là GUI:

```
if (hasGUI()) {
```

Đầu tiên, xác định xâu được in ra màn hình:

```
char label[50];
sprintf(label, "last hopCount = %d", hopcount);
```

Tiếp theo, cần xác định con trỏ trỏ đến đối tượng text hiển thị trên màn hình. Vì đoạn mã này được thực thi dưới góc nhìn là các nút tic[0]..tic[6], mà đối tượng text lại nằm ở mô-đun cha (tức Network17). Vì thế phải tiếp cận đến mô-đun cha:

```
cCanvas *canvas = getParentModule()->getCanvas();
```

Từ đối tượng canvas của mô-đun cha, lấy ra con trỏ trỏ đến đối tượng text có tên "lasthopcount" (được định nghĩa trong tệp .ned ở phần parameters)

```
canvas->getFigure("lasthopcount")
```

Sau đó ép kiểu nó sang kiểu cTextFigure*:

```
cTextFigure *textFigure =
check_and_cast<cTextFigure*>(canvas->getFigure("lasthopcount"));
```

Cuối cùng, cập nhật lại nội dung của text:

```
textFigure->setText(label);
```

## 3. Đặt a và b là số ký tự trong tên và họ tên đệm của bạn, sau đó đặt X=min{a,b} và Y=max{a,b}.

"Nguyễn Huy Hoàng"
X = 3, Y = 5

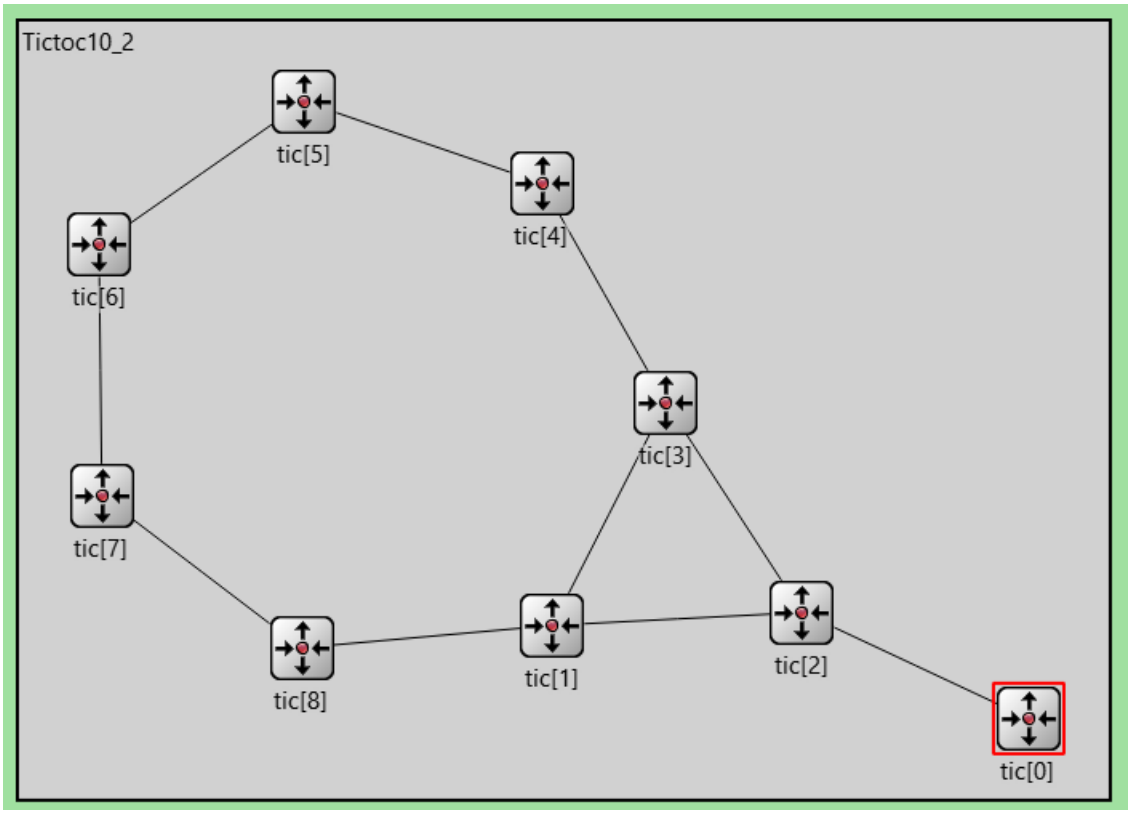
a) Dựa trên ví dụ của *Tictoc10* hãy tạo ra một network có trên (X+Y) nút trong đó có X nút bậc 3 (nghĩa là có 3 láng giềng) và Y nút bậc 2 (2 neighbors). Chẳng hạn với lên Trần An thì network trong Tictoc10 có 6 nút, 2 có bậc ba và 4 có bậc 2.

Xác định: Mạng có tổng cộng 9 nút bao gồm 3 nút bậc ba, 5 nút bậc hai và 1 nút bậc 1.

tictoc10_2.ned

```
simple Txc10_2
{
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

network Tictoc10_2
{
    types:
        channel Channel extends ned.DelayChannel
        {
            delay = 100ms;
        }
```

```
    submodules:
        tic[9]: Txc10_2;
    connections:
                tic[1].gate++ <--> Channel <--> tic[3].gate++;
                tic[0].gate++ <--> Channel <--> tic[2].gate++;

        tic[1].gate++ <--> Channel <--> tic[2].gate++;
                tic[2].gate++ <--> Channel <--> tic[3].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
        tic[5].gate++ <--> Channel <--> tic[6].gate++;
        tic[6].gate++ <--> Channel <--> tic[7].gate++;
        tic[7].gate++ <--> Channel <--> tic[8].gate++;
        tic[8].gate++ <--> Channel <--> tic[1].gate++;
}
```




b) Sửa mã nguồn sao cho mỗi gói tin (message) sẽ được gửi sao cho nó đi qua tất cả các hop (nút) và in ra giá trị hop count khi gói tin đến được nút cuối cùng. **Chú ý rằng người lập trình cho phép tùy chọn nút nguồn và nút đích qua file omnetpp.ini.**

Ý tưởng: Lưu thông tin về các nút đã đi qua vào một mảng nằm trong đối tượng Message.

Cụ thể: Thông điệp được gửi từ nút này tới một nút kề ngẫu nhiên. Mỗi khi thông điệp tới một nút X, đánh dấu vào mảng visited[X] = true. Tiếp đó kiểm tra xem đã đi hết tất cả các nút hay chưa, nếu chưa thì gửi tiếp, còn nếu rồi thì in ra thông tin và xoá thông điệp, dừng chương trình.

Source:

tictoc10_2.ned:

```
simple Txc10_2
{
```

```
    parameters:
        @display("i=block/routing");
        bool sendOnInit = default(false); // Tham số đánh dấu nút nguồn
    gates:
        inout gate[];
}

network Tictoc10_2
{
    types:
        channel Channel extends ned.DelayChannel
        {
            delay = 100ms;
        }
    submodules:
        tic[9]: Txc10_2;
    connections:
                tic[1].gate++ <--> Channel <--> tic[3].gate++;
                tic[0].gate++ <--> Channel <--> tic[2].gate++;

        tic[1].gate++ <--> Channel <--> tic[2].gate++;
                tic[2].gate++ <--> Channel <--> tic[3].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
        tic[5].gate++ <--> Channel <--> tic[6].gate++;
        tic[6].gate++ <--> Channel <--> tic[7].gate++;
        tic[7].gate++ <--> Channel <--> tic[8].gate++;
        tic[8].gate++ <--> Channel <--> tic[1].gate++;
}
```

txc10_2.cc:

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "tictoc10_2_m.h"

using namespace omnetpp;

class Txc10_2: public cSimpleModule {
protected:
    virtual void forwardMessage(TicTocMsg10_2 *msg);
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Txc10_2);

void Txc10_2::initialize() {
    if (par("sendOnInit").boolValue()) {
        TicTocMsg10_2 *msg = new TicTocMsg10_2("hello");
        for (int i = 0; i < 9; i++) {
            msg->setVisited(i, false);
        }
        scheduleAt(0.0, msg);
    }
}

void Txc10_2::handleMessage(cMessage *msg) {
    TicTocMsg10_2 *ttmsg = check_and_cast<TicTocMsg10_2*>(msg);
```

```
    // Đánh dấu nút đã đi qua
    ttmsg->setVisited(getIndex(), true);

    // Tăng giá trị hop count
    ttmsg->setHopCount(ttmsg->getHopCount() + 1);

    // Kiểm tra xem đã đi hết các nút hay chưa
    bool visitedAll = true;
    for (int i = 0; i < 9; i++) {
        if (ttmsg->getVisited(i) == false) {
            visitedAll = false;
            break;
        }
    }

    if (visitedAll) {
        // Done
        EV << "Visited all, hop count = " << ttmsg->getHopCount();
        delete msg;
    } else {
        forwardMessage(ttmsg);
    }
}

void Txc10_2::forwardMessage(TicTocMsg10_2 *msg) {
    int n = gateSize("gate");
    int k = intuniform(0, n-1);

    EV << "Forwarding message " << msg << " on gate[" << k << "]\n";
    send(msg, "gate$o", k);
}
```

tictoc10_2.msg

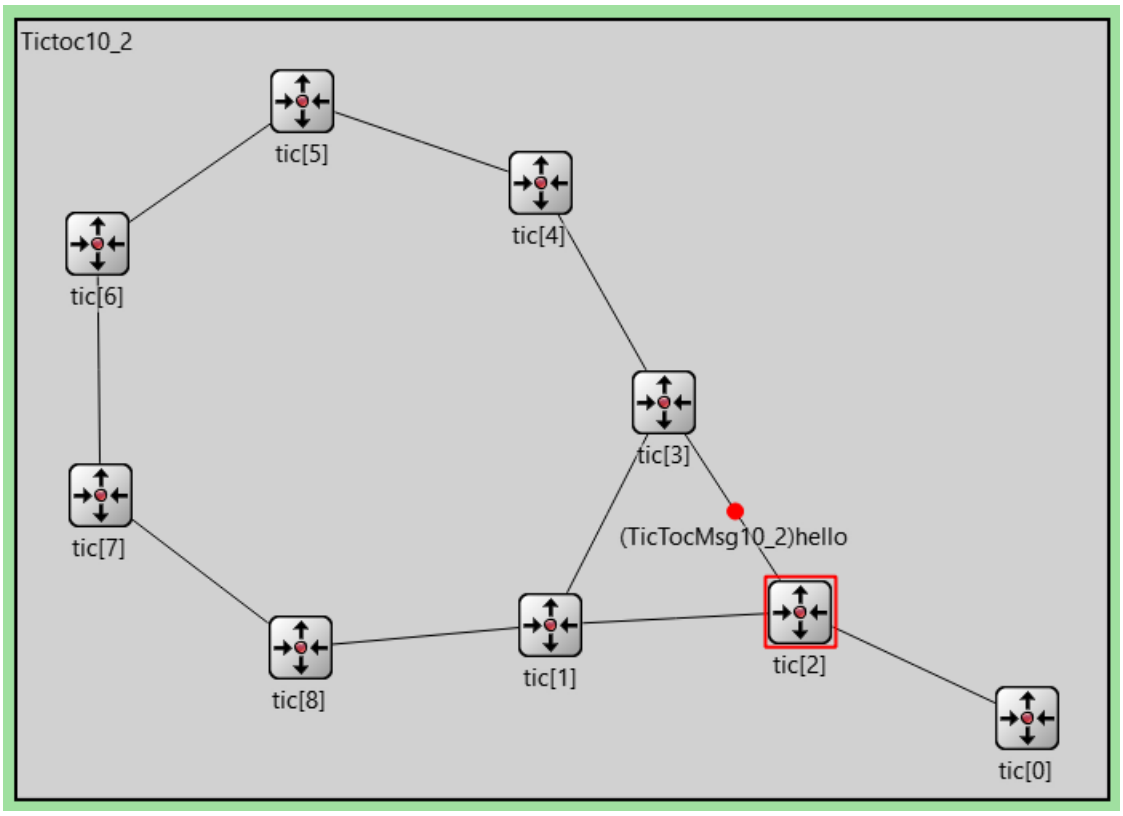
```
message TicTocMsg10_2
{
    int hopCount = 0;
    bool visited[9]; // Mảng lưu thông tin về các nút đã đi qua
}
```

omnetpp.ini:

```
[Config Tictoc10_2]
network = Tictoc10_2
Tictoc10_2.tic[0].sendOnInit = true
```

kết quả:

running




kết thúc:

```
Initializing module Tictoc10_2.tic[3], stage 0
Initializing module Tictoc10_2.tic[4], stage 0
Initializing module Tictoc10_2.tic[5], stage 0
Initializing module Tictoc10_2.tic[6], stage 0
Initializing module Tictoc10_2.tic[7], stage 0
Initializing module Tictoc10_2.tic[8], stage 0
** Event #1  t=0  Tictoc10_2.tic[0] (Txc10_2, id=2)  on selfmsg hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[0]
** Event #2  t=0.1  Tictoc10_2.tic[2] (Txc10_2, id=4)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[1]
** Event #3  t=0.2  Tictoc10_2.tic[1] (Txc10_2, id=3)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[0]
** Event #4  t=0.3  Tictoc10_2.tic[3] (Txc10_2, id=5)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[1]
** Event #5  t=0.4  Tictoc10_2.tic[2] (Txc10_2, id=4)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[1]
** Event #6  t=0.5  Tictoc10_2.tic[1] (Txc10_2, id=3)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[2]
** Event #7  t=0.6  Tictoc10_2.tic[8] (Txc10_2, id=10)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[0]
** Event #8  t=0.7  Tictoc10_2.tic[7] (Txc10_2, id=9)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[1]
** Event #9  t=0.8  Tictoc10_2.tic[8] (Txc10_2, id=10)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[0]
** Event #10  t=0.9  Tictoc10_2.tic[7] (Txc10_2, id=9)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[0]
** Event #11  t=1  Tictoc10_2.tic[6] (Txc10_2, id=8)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[0]
** Event #12  t=1.1  Tictoc10_2.tic[5] (Txc10_2, id=7)  on hello (TicTocMsg10_2, id=1)
INFO:Forwarding message (TicTocMsg10_2)hello on gate[0]
** Event #13  t=1.2  Tictoc10_2.tic[4] (Txc10_2, id=6)  on hello (TicTocMsg10_2, id=1)
Visited all, hop count = 13
<!> No more events, simulation completed -- at t=1.2s, event #13
** Calling finish() methods of modules
```

4. Dựa trên câu hỏi 3, tạo ra mã nguồn sao cho mỗi nút nắm được các láng giềng trực tiếp kết nối đến nó. Kịch bản nắm bắt nút láng giềng như sau: tất cả các nút gửi gói tin đến với nút láng giềng của nó. Một khi một nút biết được tất cả các láng giềng của nó (3 hoặc 2 láng giềng tùy vào bậc) thì nút này in ra danh sách tên các láng giềng cũng như bậc của láng giềng.

tictoc10_3.ned

```
simple Txc10_3
{
```

```
    parameters:
        @display("i=block/routing");
    gates:
        inout gate[];
}

network Tictoc10_3
{
    types:
        channel Channel extends ned.DelayChannel
        {
            delay = 100ms;
        }
    submodules:
        tic[9]: Txc10_3;
    connections:
                tic[1].gate++ <--> Channel <--> tic[3].gate++;
                tic[0].gate++ <--> Channel <--> tic[2].gate++;

        tic[1].gate++ <--> Channel <--> tic[2].gate++;
                tic[2].gate++ <--> Channel <--> tic[3].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
        tic[5].gate++ <--> Channel <--> tic[6].gate++;
        tic[6].gate++ <--> Channel <--> tic[7].gate++;
        tic[7].gate++ <--> Channel <--> tic[8].gate++;
        tic[8].gate++ <--> Channel <--> tic[1].gate++;
}
```

tictoc10_3.msg

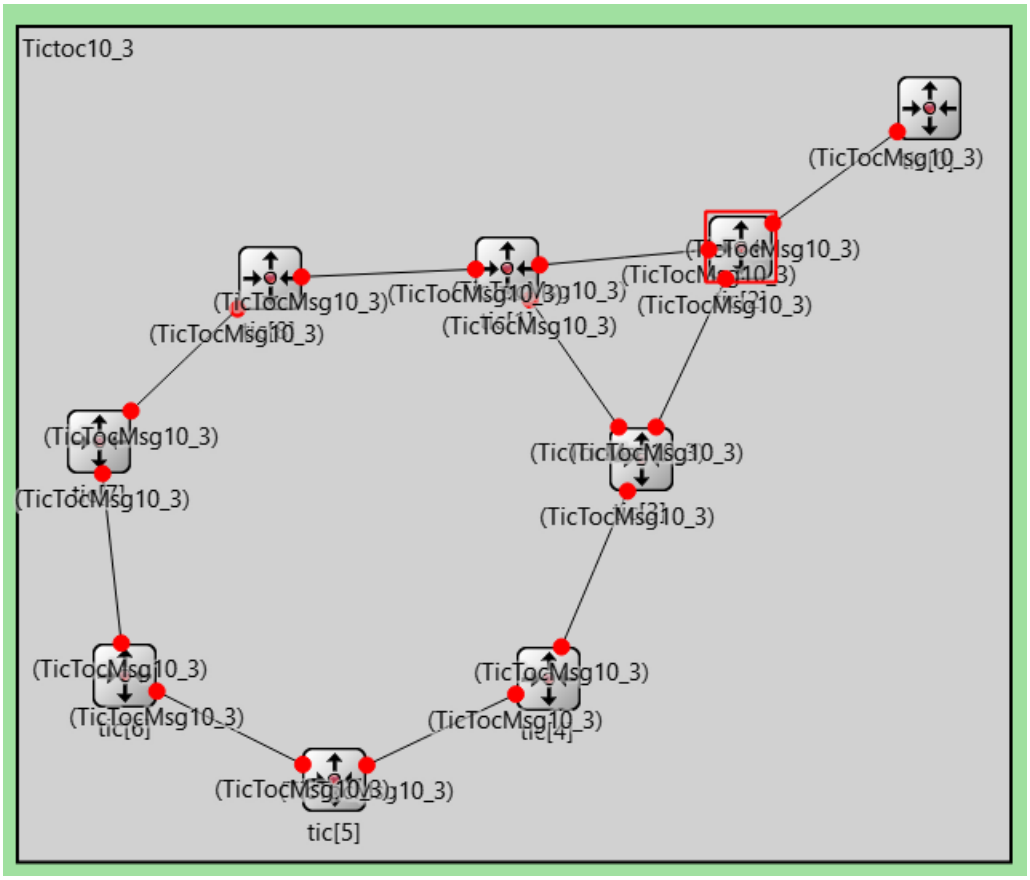
```
message TicTocMsg10_3
{
    string src_name;
        int src_degree;
}
```

tictoc10_3.cc

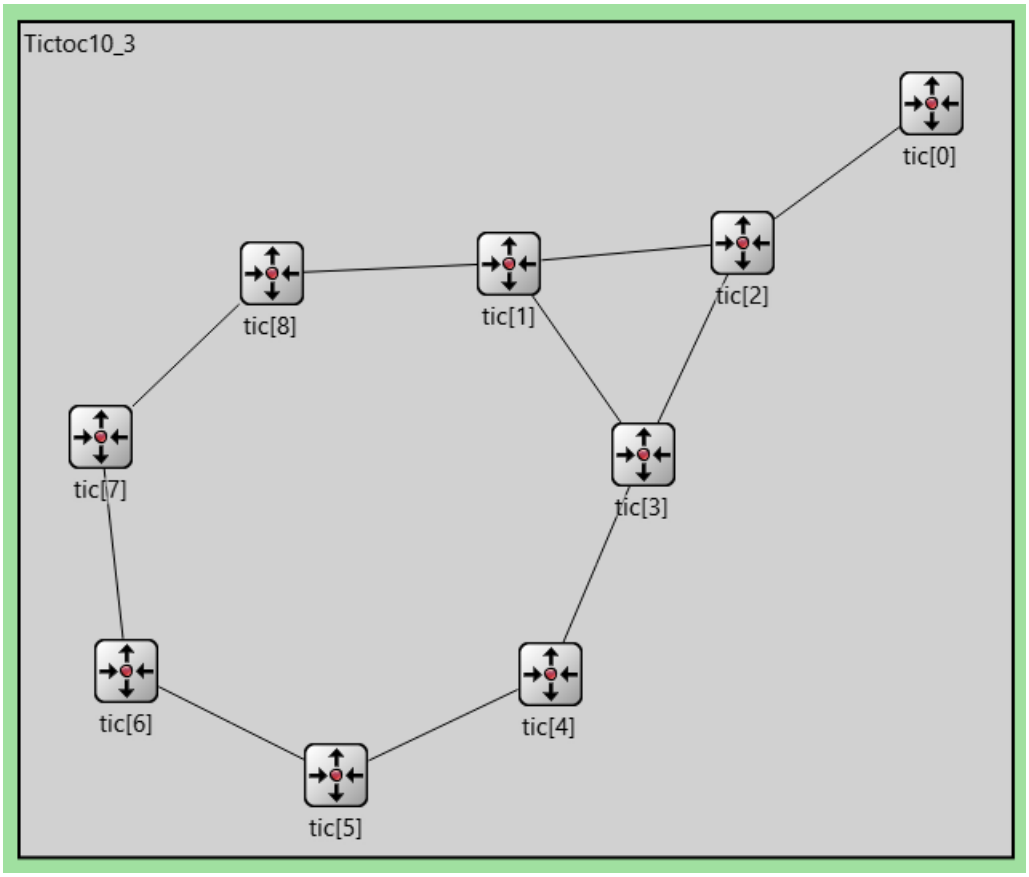
```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include <vector>
#include "tictoc10_3_m.h"

using namespace omnetpp;

class Txc10_3: public cSimpleModule {
private:
    std::vector<TicTocMsg10_3*> neighbors; // Lưu thông tin các nút láng giềng
protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Txc10_3);

void Txc10_3::initialize() {
    // Tạo một mẫu thông điệp chứa thông tin về tên và bậc của nút gửi
```

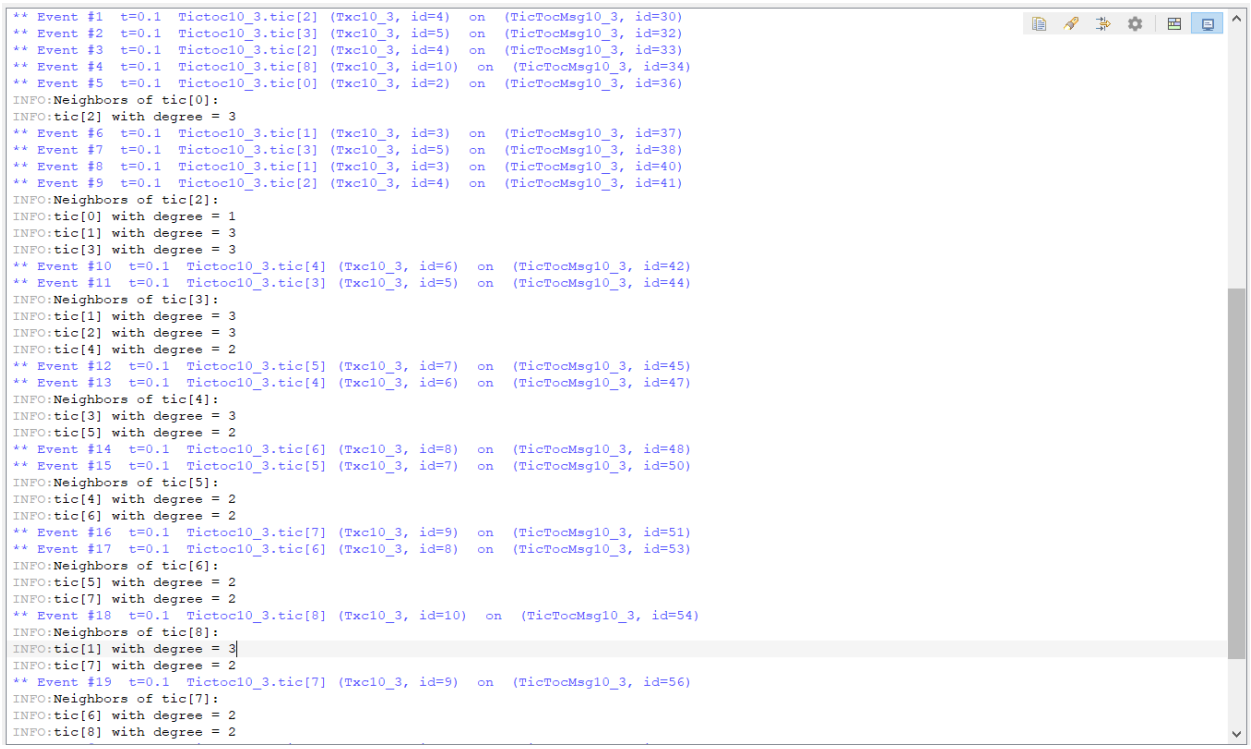
```
    TicTocMsg10_3 *msg = new TicTocMsg10_3();
    msg->setSrc_name(getFullName());
    msg->setSrc_degree(gateSize("gate"));

    // Gửi bản sao thông điệp tới tất cả các nút láng giềng
    for(int i = 0; i < gateSize("gate"); i++){
        TicTocMsg10_3 *copyMsg = new TicTocMsg10_3(*msg);
        send(copyMsg, "gate$o", i);
    }

    // Xoá mẫu
    delete msg;
}

void Txc10_3::handleMessage(cMessage *msg) {
    TicTocMsg10_3 *ttmsg = check_and_cast<TicTocMsg10_3*>(msg);

    // Tạm lưu lại thông điệp vào danh sách
    neighbors.push_back(ttmsg);

    if(neighbors.size() == gateSize("gate")){
        // Đã nhận đủ thông điệp, in ra danh sách các nút láng giềng
        EV << "Neighbors of " << getFullName() << ":" << endl;
        for(auto m : neighbors){
            EV << m->getSrc_name() << " with degree = " << m->getSrc_degree() << endl;
            delete m;
        }
    }
}
```

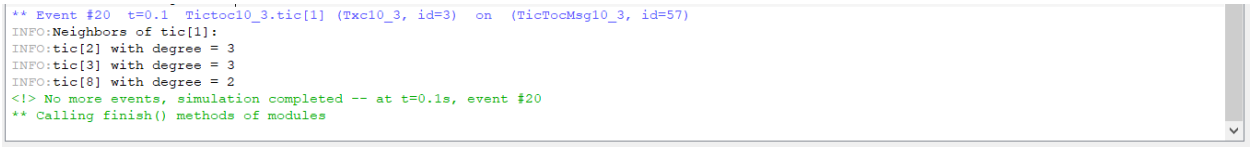
khởi tạo:


Tictoc10_3
(TicTocMsg10_3)
tic[5]

chạy xong:


Tictoc10_3
tic[0]
tic[8]
tic[1]
tic[2]
tic[7]
tic[3]
tic[6]
tic[4]
tic[5]




5. Dựa trên câu hỏi 3, tạo ra mã nguồn sao cho sử dụng tin nhắn di chuyển qua tất cả các nút trong mạng, về lại nút ban đầu và in ra ma trận kết nối dạng 0 1 giữa các nút [i, j]. **Chú ý rằng người lập trình cho phép tùy chọn nút nguồn qua file omnetpp.ini.**

Ý tưởng chính: Coi mạng như một đồ thị, duyệt cây theo thuật toán DFS

Chương trình cần thêm:
1. Lớp Step lưu thông tin: nút gửi, cổng gửi
2. Lớp TicToc10_4_msg kế thừa lớp cMessage, lưu các thông tin: chỉ số của nút gửi, ma trận kết nối, stack lưu các bước cần di chuyển.

Source:

tictoc10_4.ned:

```
simple Txc10_4
{
    parameters:
        @display("i=block/routing");
        bool sendOnInit = default(false);
    gates:
        inout gate[];
}

network Tictoc10_4
{
    types:
        channel Channel extends ned.DelayChannel
        {
```

```
            delay = 100ms;
        }
    submodules:
        tic[9]: Txc10_4;
    connections:
            tic[1].gate++ <--> Channel <--> tic[3].gate++;
            tic[0].gate++ <--> Channel <--> tic[2].gate++;

        tic[1].gate++ <--> Channel <--> tic[2].gate++;
            tic[2].gate++ <--> Channel <--> tic[3].gate++;
        tic[3].gate++ <--> Channel <--> tic[4].gate++;
        tic[4].gate++ <--> Channel <--> tic[5].gate++;
        tic[5].gate++ <--> Channel <--> tic[6].gate++;
        tic[6].gate++ <--> Channel <--> tic[7].gate++;
        tic[7].gate++ <--> Channel <--> tic[8].gate++;
        tic[8].gate++ <--> Channel <--> tic[1].gate++;
}
```

omnetpp.ini:

```
[Config Tictoc10_4]
network = Tictoc10_4
Tictoc10_4.tic[0].sendOnInit = true
```

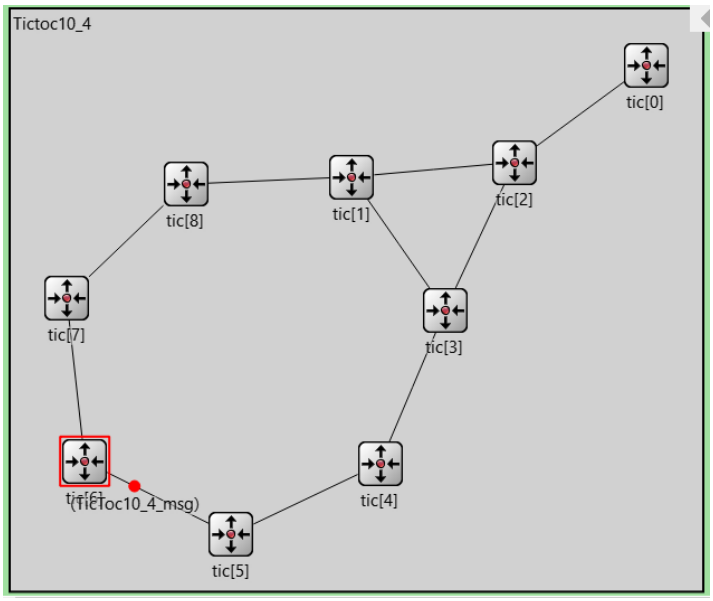
txc10_4.cc

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

using namespace omnetpp;
using namespace std;

// Lớp lưu thông tin về một bước di chuyển của tin nhắn
class Step {
public:
    string nodeName; // Tên của nút gửi
    int gateIndex; // Cổng dùng để gửi
    bool isReturnStep; // Có phải là bước quay lại hay không?

    Step(string nodeName, int gateIndex, bool isReturnStep) {
        this->nodeName = nodeName;
        this->gateIndex = gateIndex;
        this->isReturnStep = isReturnStep;
    }

    // So sánh bỏ qua thuộc tính isReturnStep
    bool equals(Step other) {
        return other.gateIndex == gateIndex && other.nodeName.compare(nodeName) == 0;
    }
};

// Lớp tin nhắn, dành riêng cho bài này
class TicToc10_4_msg: public cMessage {
public:
    int source; // Index của nút gửi
    int ** connectionMatrix; // Ma trận kết nối

    list<Step> stack; // Stack dùng để lưu các bước di chuyển theo cách duyệt DFS
```

```
    // Hàm kiểm tra xem một bước đi đã tồn tại trong stack hay chưa,
    // Nếu đã tồn tại thì xoá cái cũ đi
    // Sau đó push cái mới vào
    void checkAndPush(Step newStep){
        for(auto it = stack.begin(); it!=stack.end(); ++it){
            if(newStep.equals(*it)){
                stack.erase(it);
                break;
            }
        }
        stack.push_back(newStep);
    }
};

class Txc10_4: public cSimpleModule {
private:
    bool visited = false; // Đánh dấu nút đã thăm
protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Txc10_4);

void Txc10_4::initialize() {
    // Bắt đầu từ nút nguồn
    if (par("sendOnInit").boolValue()) {
        // Khởi tạo tin nhắn
        TicToc10_4_msg *msg = new TicToc10_4_msg();

        // Khởi tạo ma trận kết nối với tất cả các phần từ bằng 0
        msg->connectionMatrix = new int*[9];
        for (int i = 0; i < 9; i++) {
            msg->connectionMatrix[i] = new int[9];
            for (int j = 0; j < 9; j++) {
                msg->connectionMatrix[i][j] = 0;
            }
        }

        // Khởi tạo stack
        for (int i = 0; i < gateSize("gate"); i++) {
            msg->stack.push_back(*(new Step(getFullName(), i, false)));
        }

        // Đánh dấu đã thăm nút nguồn
        visited = true;

        msg->source = getIndex();

        // Lấy ra cổng đầu tiên để gửi (lấy top stack)
        int nextGate = (msg->stack.back()).gateIndex;

        send((cMessage*) msg, "gate$o", nextGate);
    }
}

void Txc10_4::handleMessage(cMessage *cMsg) {
    TicToc10_4_msg *msg = (TicToc10_4_msg*) cMsg;
    bool isReturnStep = msg->stack.back().isReturnStep;
    msg->stack.pop_back();
```

```
    if (!isReturnStep){
        // Điền vào ma trận
        int index1 = msg->source;
        int index2 = getIndex();
        msg->connectionMatrix[index1][index2] = 1;
        msg->connectionMatrix[index2][index1] = 1;

        // Tạo 1 bước đi để quay lại nút trước rồi push vào stack
        cGate* arrivalGate = msg->getArrivalGate();
        Step *returnStep = new Step(getFullName(), arrivalGate->getIndex(), true);
        msg->checkAndPush(*returnStep);


        if (!visited){
            // Tạo các bước đi mới (không trùng lặp với bước quay lại vừa tạo ở trên),
rồi push vào stack
            for (int i = 0; i < gateSize("gate"); i++){
                if (gate("gate$i", i) != arrivalGate){
                    Step *newStep = new Step(getFullName(), i, false);
                    msg->checkAndPush(*newStep);
                }
            }
        }

        // Đánh dấu đã thăm nút hiện tại
        visited = true;
    }

    // Nếu stack rỗng tức là đã duyệt xong
    if (msg->stack.empty()) {
        EV << "Connection matrix: " << endl;
        for(int i = 0; i < 9; i++){
            for(int j = 0; j < 9; j++){
                EV << msg->connectionMatrix[i][j] << " ";
            }
            EV << endl;
        }
        delete msg;
    }
    else{
        // Lấy ra cổng tiếp theo (top stack) rồi gửi đi.
        int nextGate = (msg->stack.back()).gateIndex;
        msg->source = getIndex();
        send((cMessage*) msg, "gate$o", nextGate);
    }
}
```

running:




output:

```
** Event #19  t=1.9  Tictoc
** Event #20  t=2  Tictoc
INFO:Connection matrix:
INFO:0 0 1 0 0 0 0 0 0
INFO:0 0 1 1 0 0 0 0 1
INFO:1 1 0 1 0 0 0 0 0
INFO:0 1 1 0 1 0 0 0 0
INFO:0 0 0 1 0 1 0 0 0
INFO:0 0 0 0 1 0 1 0 0
INFO:0 0 0 0 0 1 0 1 0
INFO:0 0 0 0 0 0 1 0 1
INFO:0 1 0 0 0 0 0 1 0
<!> No more events, simula
** Calling finish() methoc
```