**Hướng dẫn Tictoc cho OMNeT++**

Hướng dẫn OMNeT++ ngắn gọn này hướng dẫn bạn thông qua một ví dụ của mô hình hóa và mô phỏng, cho thấy các tính năng thường được sử dụng của OMNeT++.

Hướng dẫn này dựa trên mô phỏng ví dụ Tictoc, mà bạn có thể tìm thấy trong samples/tictoc trong thư mục cài đặt OMneT++ của bạn, vì vậy bạn có thể thử ngay cách hoạt động các ví dụ. Tuy nhiên, bạn sẽ thấy bài hướng dẫn này hữu ích hơn nhiều nếu bạn thực sự thực hiện ít nhất các bước đầu tiên được mô tả ở đây.

**Chú ý**

Ở đây, chúng tôi cho rằng bạn đã cài đặt môi trường OMNeT++. Chúng tôi cũng cho rằng bạn đã có kiến thức tốt về C++ và đã quen với việc phát triển C++( chỉnh sửa tệp nguồn, biên dịch và gỡ lỗi …) trên hệ điều hành của bạn. (Hai phần sau nằm ngoài phạm vi của chúng tôi ở đây - có những cuốn sách, hướng dẫn tuyệt vời trên web, v.v. nếu bạn cần cập nhật kiến ​​thức của mình về điều đó.) Chúng tôi thực sự khuyên bạn nên sử dụng Môi trường phát triển tích hợp OMNeT ++ để chỉnh sửa và xây dựng mô phỏng.

Để làm cho các ví dụ dễ theo dõi hơn, tất cả mã nguồn ở đây được liên kết chéo với tài liệu API OMNeT ++.

Tài liệu này và mô hình TicToc là phiên bản mở rộng của hướng dẫn TicToc gốc của Ahmet Sekercioglu (Đại học Monash).

# 1. Bắt đầu

Vì lĩnh vực ứng dụng phổ biến nhất của OMNeT ++ là mô phỏng các mạng viễn thông, chúng tôi sẽ mượn chủ đề của chúng tôi từ đó. Để bắt đầu, chúng ta hãy bắt đầu với một "mạng" bao gồm hai nút. Các nút sẽ thực hiện một việc đơn giản: một trong các nút sẽ tạo một gói tin và hai nút sẽ tiếp tục chuyển qua lại cùng một gói tin. Chúng tôi sẽ gọi các nút là "tic" và "toc".

Dưới đây là các bước bạn thực hiện để thực hiện mô phỏng đầu tiên của mình từ đầu:

1. Tạo một thư mục làm việc có tên là tictoc và cd vào thư mục này.

2. Mô tả mạng ví dụ của bạn bằng cách tạo tệp cấu trúc liên kết. Tệp cấu trúc liên kết là tệp văn bản xác định các nút của mạng và các liên kết giữa chúng. Bạn có thể tạo nó bằng trình soạn thảo văn bản yêu thích của mình. Hãy gọi nó là tictoc1.ned:

//

// This file is part of an OMNeT++/OMNEST simulation example.

//

// Copyright (C) 2003 Ahmet Sekercioglu

// Copyright (C) 2003-2008 Andras Varga

//

// This file is distributed WITHOUT ANY WARRANTY. See the file

// `license' for details on this and other legal matters.

//

simple [Txc1](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html)

{

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

{

submodules:

tic: [Txc1](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html);

toc: [Txc1](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html);

connections:

tic.out --> { delay = 100ms; } --> toc.in;

tic.in <-- { delay = 100ms; } <-- toc.out;

}

Tệp này tốt nhất nên đọc từ dưới lên. Đây là những gì nó nói:

* Tictoc1 là một mạng được tập hợp từ hai mô-đun con, tic và toc. tic và toc là các thể hiện của cùng một loại mô-đun được gọi là [**Txc1**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html) . Chúng tôi kết nối cổng đầu ra của tic (được đặt tên là out) với cổng đầu vào của toc (được đặt tên là in) và ngược lại ( network ... { ... }). Sẽ có độ trễ lan truyền 100ms theo cả hai cách;
* [**Txc1**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html) là một loại mô-đun đơn giản (có nghĩa là nó là nguyên tử ở cấp NED và sẽ được triển khai bằng C ++). [**Txc1**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html) có một cổng đầu vào có tên là in và một cổng đầu ra có tên là out ( simple ... { ... }).

3. Bây giờ chúng ta cần triển khai chức năng của mô-đun đơn giản [**Txc1**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html) . Điều này đạt được bằng cách viết tệp C ++ txc1 :: cc:

//

// This file is part of an OMNeT++/OMNEST simulation example.

//

// Copyright (C) 2003 Ahmet Sekercioglu

// Copyright (C) 2003-2008 Andras Varga

//

// This file is distributed WITHOUT ANY WARRANTY. See the file

// `license' for details on this and other legal matters.

//

#include <string.h>

#include <omnetpp.h>

class [Txc1](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html) : public [cSimpleModule](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html)

{

protected:

// The following redefined virtual function holds the algorithm.

virtual void [initialize](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html#a0ac4374257ba261d38f866f61790af82)();

virtual void [handleMessage](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html#a0d08f73d393d9f83d9eef862656f0927)([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*msg);

};

// The module class needs to be registered with OMNeT++

[Define\_Module](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/txc1_8cc.html#a5ac407c43df69e3adc074c5c56d511ff)([Txc1](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html));

void [Txc1::initialize](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html#a0ac4374257ba261d38f866f61790af82)()

{

// 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](https://doc.omnetpp.org/omnetpp4/api//classcNamedObject.html#a5b06a5fdb473bb997c939ba55435c041)()) == 0)

{

// create and send first message on gate "out". "tictocMsg" is an

// arbitrary string which will be the name of the message object.

[cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*msg = new [cMessage](https://doc.omnetpp.org/omnetpp4/api//classcOwnedObject.html#af46e516dfdf6a913fe913c4c2e496cf2)("tictocMsg");

[send](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html#a94590ba6481116f29f4c43d00733deea)(msg, "out");

}

}

void [Txc1::handleMessage](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html#a0d08f73d393d9f83d9eef862656f0927)([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*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](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html#a94590ba6481116f29f4c43d00733deea)(msg, "out");

}

Các mô-đun đơn giản kiểu [**Txc1**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html) được thể hiện bằng các lớp C ++ [**Txc1**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc1.html) , trong đó phải là lớp con từ [cSimpleModule](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html) , và ghi tại OMNeT ++ với lệnh [**Define\_Module()**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/txc1_8cc.html#a5ac407c43df69e3adc074c5c56d511ff). Chúng tôi định nghĩa lại hai phương thức từ [cSimpleModule](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html) : initialize () và handleMessage(). Chúng được gọi từ nhân mô phỏng: cái đầu tiên chỉ một lần và cái thứ hai bất cứ khi nào có thông báo đến mô-đun.

Trong intialize(), chúng tôi tạo một đối tượng thông điệp ( [cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) ) và gửi nó ra cổng out. Vì cổng này được kết nối với cổng đầu vào của mô-đun khác, nhân mô phỏng sẽ gửi thông báo này tới mô-đun khác trong đối số tới handleMessage () - sau độ trễ truyền 100ms được gán cho liên kết trong tệp NED. Mô-đun khác chỉ gửi lại nó (độ trễ 100ms nữa), vì vậy nó sẽ dẫn đến sự chuyền liên tục.

Thông điệp (gói, khung, công việc, v.v.) và sự kiện (bộ định thời gian, thời gian chờ) đều được biểu diễn bằng các đối tượng [cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) (hoặc các lớp con của nó) trong OMNeT ++. Sau khi bạn gửi hoặc lên lịch cho chúng, chúng sẽ được nhân mô phỏng giữ trong danh sách "các sự kiện đã lên lịch" hoặc "các sự kiện trong tương lai" cho đến khi đến thời điểm của chúng và chúng được chuyển đến các mô-đun thông qua handleMessage ().

Lưu ý rằng không có điều kiện dừng được tích hợp trong mô phỏng này: nó sẽ tiếp tục mãi mãi. Bạn sẽ có thể dừng nó khỏi GUI. (Bạn cũng có thể chỉ định giới hạn thời gian mô phỏng hoặc giới hạn thời gian CPU trong tệp cấu hình, nhưng chúng tôi không làm điều đó trong hướng dẫn.)

4. Bây giờ chúng tôi tạo Makefile sẽ giúp chúng tôi biên dịch và liên kết chương trình của chúng tôi để tạo tictoc thực thi:

$ opp\_makemake

Lệnh này bây giờ nên tạo một Makefile trong thư mục làm việc tictoc.

**Chú ý:**

Người dùng Windows + MSVC: lệnh là opp\_*n*makemake và nó sẽ tạo Makefile.vc. Lưu ý: MSVC chỉ được hỗ trợ trong phiên bản thương mại OMNEST.

5. Bây giờ chúng ta hãy biên dịch và liên kết mô phỏng đầu tiên của chúng ta bằng cách ra lệnh make:

$ make

**Chú ý:**

Windows + MSVC: gõ lệnh nmake -f Makefile.vc. Nếu bạn nhận được thông báo *'nmake' không được công nhận là lệnh nội bộ hoặc lệnh bên ngoài ...* , hãy tìm vcvars32.bat ở một nơi nào đó trong thư mục MSVC và chạy nó trước trong mọi cửa sổ lệnh mà bạn muốn biên dịch. Lưu ý: MSVC chỉ được hỗ trợ trong phiên bản thương mại OMNEST.

Nếu có lỗi biên dịch, bạn cần phải sửa những lỗi đó và lặp lại việc thực hiện cho đến khi bạn nhận được liên kết và biên dịch không có lỗi.

6. Nếu bạn khởi động tệp thực thi ngay bây giờ, nó sẽ thông báo rằng nó không thể tìm thấy tệp omnetpp.ini, vì vậy bạn phải tạo một tệp. omnetpp.ini cho chương trình mô phỏng biết mạng nào bạn muốn mô phỏng (vâng, một số mạng có thể sống trong cùng một chương trình mô phỏng), bạn có thể truyền các tham số cho mô hình, chỉ định rõ ràng seeds cho các trình tạo số ngẫu nhiên, v.v.

Tạo omnetpp.ini rất đơn giản sau:

[General]

network = Tictoc1

tictoc2 and và các bước tiếp theo sẽ chung tệp omnetpp.ini sau:

# This file is shared by all tictoc simulations.

# Lines beginning with `#' are comments

[General]

# nothing here

[Config Tictoc1]

network = Tictoc1

[Config Tictoc2]

network = Tictoc2

[Config Tictoc3]

network = Tictoc3

[Config Tictoc4]

network = Tictoc4

Tictoc4.toc.limit = 5

[Config Tictoc5]

network = Tictoc5

\*\*.limit = 5

[Config Tictoc6]

network = Tictoc6

[Config Tictoc7]

network = Tictoc7

# argument to exponential() is the mean; truncnormal() returns values from

# the normal distribution truncated to nonnegative values

Tictoc7.tic.delayTime = [exponential](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersCont.html#ga1f77033d9815fe55a847f5aefdbbb1a7)(3s)

Tictoc7.toc.delayTime = [truncnormal](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersCont.html#ga80b20109dbe65191c8404c58cef90917)(3s,1s)

[Config Tictoc8]

network = Tictoc8

[Config Tictoc9]

network = Tictoc9

[Config Tictoc10]

network = Tictoc10

[Config Tictoc11]

network = Tictoc11

[Config Tictoc12]

network = Tictoc12

[Config Tictoc13]

network = Tictoc13

[Config Tictoc14]

network = Tictoc14

[Config Tictoc15]

network = Tictoc15

record-eventlog = true

[Config Tictoc16]

network = Tictoc16

\*\*.tic[1].hopCount.result-recording-modes = +histogram

\*\*.tic[0..2].hopCount.result-recording-modes = -vector

7. Một khi bạn hoàn thành các bước trên, bạn có thể chạy mô phỏng bằng cách sử dụng lệnh sau:

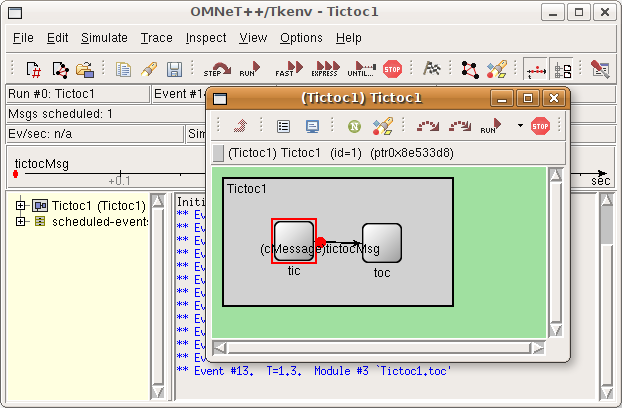
$ ./tictoc

và hy vọng bây giờ bạn sẽ nhận được cửa sổ mô phỏng OMNeT ++.

**Chú ý:**

Windows: lệnh chỉ là tictoc.

8.Nhấn nút Run trên thanh công cụ để bắt đầu mô phỏng. Điều bạn sẽ thấy là tic và toc đang trao đổi tin nhắn với nhau..



Thanh công cụ cửa sổ chính hiển thị thời gian mô phỏng. Đây là thời gian ảo, nó không liên quan gì đến thời gian thực tế (hoặc đồng hồ treo tường) mà chương trình cần để thực thi. Trên thực tế, bạn có thể mô phỏng bao nhiêu giây trong một giây trong thế giới thực phụ thuộc nhiều vào tốc độ phần cứng của bạn và thậm chí nhiều hơn vào bản chất và độ phức tạp của chính mô hình mô phỏng.

Lưu ý rằng không mất thời gian mô phỏng để một nút xử lý thông điệp. Điều duy nhất làm cho thời gian mô phỏng trôi qua trong mô hình này là độ trễ lan truyền trên các kết nối.

9. Bạn có thể làm chậm hoạt ảnh hoặc làm cho nó nhanh hơn bằng thanh trượt ở đầu cửa sổ đồ họa. Bạn có thể dừng mô phỏng bằng cách nhấn F8 (tương đương với nút STOP trên thanh công cụ), thực hiện một bước qua nó (F4), chạy nó với (F5) hoặc không (F6) hoạt ảnh. F7 (chế độ nhanh) tắt hoàn toàn các tính năng theo dõi để đạt tốc độ tối đa. Lưu ý đồng hồ đo sự kiện / giây và simsec / giây trên thanh trạng thái của cửa sổ chính.

10. Bạn có thể thoát khỏi chương trình mô phỏng bằng cách nhấp vào biểu tượng Đóng hoặc chọn File|Exit.

# 2. Nâng cao TicToc 2 nút

Bước 2: Tinh chỉnh đồ họa và thêm đầu ra gỡ lỗi

Ở đây chúng tôi làm cho mô hình trông đẹp hơn một chút trong GUI. Chúng tôi gán biểu tượng "block/routing" (tệp images/block/routing.png) và sơn nó màu lục lam cho tic và màu vàng cho toc. Điều này đạt được bằng cách thêm chuỗi hiển thị vào tệp NED. Các i=tag trong chuỗi hiển thị chỉ định biểu tượng.

// "block/routing" icon to the simple module. All submodules of type

// Txc2 will use this icon by default

//

simple [Txc2](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc2.html)

{

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](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc2.html) {

parameters:

@display("i=,cyan"); // do not change the icon (first arg of i=) just colorize it

}

toc: [Txc2](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc2.html) {

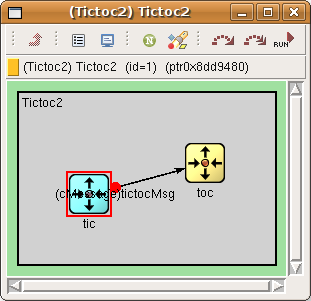
parameters:

@display("i=,gold"); // here too

}

connections:

Bạn có thể xem kết quả:



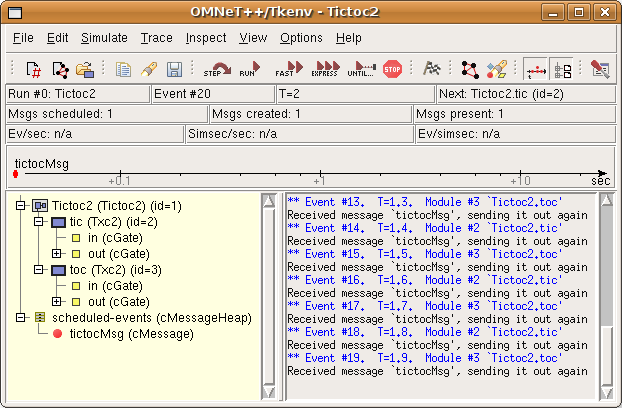
Chúng tôi cũng chỉnh sửa tệp C++ khi thêm một vài thông điệp gỡ lỗi tới Txc1 bằng cách viết đối tượng OMNeT EV như sau:

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Sending initial message\n";

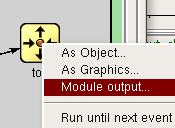
và

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Received message `" << msg->getName() << "', sending it out again\n";

Khi bạn chạy mô phỏng trong OMNeT ++ GUI Tkenv, kết quả sau sẽ xuất hiện trong cửa sổ văn bản chính:



Bạn cũng có thể mở các cửa sổ đầu ra riêng biệt cho tic và toc bằng cách nhấp chuột phải vào biểu tượng của chúng và chọn Module output từ menu. Tính năng này sẽ hữu ích khi bạn có một mô hình lớn ("hội chứng nhật ký cuộn nhanh") và bạn chỉ quan tâm đến các thông báo nhật ký của mô-đun cụ thể.



## Bước 3: Thêm các biến trạng thái

Trong bước này, chúng tôi thêm biến đếm vào mô-đun và xóa thông điệp sau mười lần trao đổi.

Chúng tôi thêm biến đếm counter làm một thành viên lớp:

class [Txc3](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc3.html) : public [cSimpleModule](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html)

{

private:

int [counter](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc3.html#aa0fbac7aacd8a61d6ce22603b7a7d55f); // Note the counter here

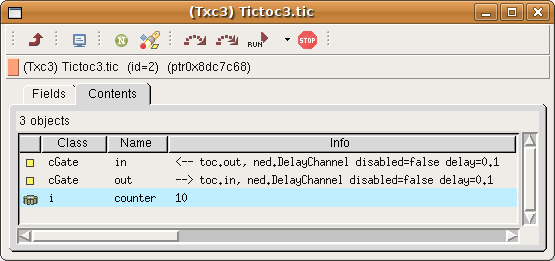
protected:

Chúng tôi đặt biến thành 10 trong intialize () và giảm trong handleMessage(), nghĩa là, mỗi khi có thông điệp tới. Sau khi nó về 0, mô phỏng sẽ hết các sự kiện và kết thúc.

Lưu ý

[WATCH](https://doc.omnetpp.org/omnetpp4/api//group__MacrosWatch.html#ga0878b62c3a2dcb0388c967a4acb2f18a)(counter);

dòng này trong tệp nguồn: điều này giúp bạn có thể thấy giá trị bộ đếm trong Tkenv. Bấm đúp vào biểu tượng của tic, sau đó chọn trang Contents từ cửa sổ thanh tra bật lên.



Khi bạn tiếp tục chạy mô phỏng, bạn có thể theo dõi khi bộ đếm tiếp tục giảm cho đến khi nó về 0.

## Bước 4: Thêm tham số

Trong bước này, bạn sẽ học cách thêm các tham số đầu vào vào mô phỏng: chúng tôi sẽ biến "số ma thuật" 10 thành một tham số và thêm một tham số boolean để quyết định xem liệu mô-đun có gửi thông điệp đầu tiên trong mã khởi tạo của nó hay không ( cho dù đây là mô-đun tic hay toc).

Các tham số mô-đun phải được khai báo trong tệp NED. Kiểu dữ liệu có thể là số, chuỗi, bool hoặc xml (kiểu sau là để dễ dàng truy cập vào các tệp cấu hình XML), trong số những kiểu khác.

simple [Txc4](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc4.html)

{

parameters:

bool sendMsgOnInit = default(false); // whether the module should send out a message on initialization

int limit = default(2); // another parameter with a default value

@display("i=block/routing");

gates:

Chúng ta cũng phải sửa đổi mã C ++ để đọc tham số trong intialize() và gán nó vào bộ đếm.

counter = par ( "limit" );

Chúng ta có thể sử dụng tham số thứ hai để quyết định có gửi tin nhắn ban đầu hay không:

if (par ( "sendMsgOnInit" ) .boolValue () == true )

Bây giờ, chúng ta có thể gán các tham số trong tệp NED hoặc từ omnetpp.ini. Các bài tập trong tệp NED được ưu tiên hơn. Bạn có thể xác định các giá trị mặc định cho các tham số nếu bạn sử dụng cú pháp default(...) trong tệp NED. Trong trường hợp này, bạn có thể đặt giá trị của tham số trong omnetpp.ini hoặc sử dụng giá trị mặc định do tệp NED cung cấp.

Ở đây, chúng tôi gán một tham số trong tệp NED:

network Tictoc4

{

submodules:

tic: [Txc4](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc4.html) {

parameters:

sendMsgOnInit = true;

@display("i=,cyan");

}

toc: [Txc4](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc4.html) {

parameters:

sendMsgOnInit = false;

@display("i=,gold");

}

connections:

và cái khác trong omnetpp.ini:

Tictoc4.toc.limit = 5

Lưu ý rằng vì omnetpp.ini hỗ trợ các ký tự đại diện và các tham số được gán từ tệp NED được ưu tiên hơn các tham số trong omnetpp.ini, chúng tôi có thể đã sử dụng

Tictoc4.t\*c.limit = 5

hoặc là

Tictoc4.\*.limit = 5

hoặc thậm chí

\*\*.limit = 5

với cùng một tác dụng. (Sự khác biệt giữa \* và \*\* là \* sẽ không khớp với dấu chấm và \*\* sẽ.)

Trong Tkenv, bạn có thể kiểm tra các tham số mô-đun trong cây đối tượng ở phía bên trái của cửa sổ chính hoặc trong trang Parameters của trình kiểm tra mô-đun (được mở bằng cách nhấp đúp vào biểu tượng mô-đun).

Mô-đun có giới hạn nhỏ hơn sẽ xóa thông điệp và do đó kết thúc mô phỏng.

Bước 5: Sử dụng kế thừa

Nếu chúng ta xem xét kỹ hơn tệp NED, chúng ta sẽ nhận ra điều đó tic và toc chỉ khác nhau về giá trị tham số và chuỗi hiển thị của chúng. Chúng ta có thể tạo một kiểu mô-đun đơn giản mới bằng cách kế thừa từ một kiểu mô-đun khác và chỉ định hoặc ghi đè một số tham số của nó. Trong trường hợp của chúng tôi, chúng tôi sẽ lấy ra hai loại mô-đun đơn giản ( Tic và Toc). Sau này, chúng ta có thể sử dụng các kiểu này khi xác định các mô-đun con trong mạng.

Dễ dàng bắt nguồn từ một mô-đun đơn giản hiện có. Đây là mô-đun cơ sở:

simple [Txc5](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc5.html)

{

parameters:

bool sendMsgOnInit = default(false);

int limit = default(2);

@display("i=block/routing");

gates:

input in;

output out;

}

Và đây là mô-đun dẫn xuất. Chúng tôi chỉ cần chỉ định các giá trị tham số và thêm một số thuộc tính hiển thị.

simple Tic5 extends [Txc5](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc5.html)

{

parameters:

@display("i=,cyan");

sendMsgOnInit = true; // Tic modules should send a message on init

}

Các mô-đun Toc tương tự, nhưng với các giá trị tham số khác nhau.

đơn giản Toc5 mở rộng [Txc5](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc5.html)

simple Toc5 extends [Txc5](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc5.html)

{

parameters:

@display("i=,gold");

sendMsgOnInit = false; // Toc modules should NOT send a message on init

}

**Chú ý:**

Việc triển khai C ++ được kế thừa từ mô-đun đơn giản cơ sở ( [**Txc4**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc4.html)).

Khi chúng tôi đã tạo các mô-đun đơn giản mới, chúng tôi có thể sử dụng chúng làm các loại mô-đun con trong mạng của mình:

network Tictoc5

{

submodules:

tic: Tic5; // the limit parameter is still unbound here. We will get it from the ini file

toc: Toc5;

connections:

Như bạn có thể thấy, định nghĩa mạng bây giờ ngắn hơn và đơn giản hơn nhiều. Kế thừa cho phép bạn sử dụng các kiểu phổ biến trong mạng của mình và tránh các định nghĩa và cài đặt tham số dư thừa.

Bước 6: Mô hình hóa độ trễ xử lý

Trong các mô hình trước, tic và toc ngay lập tức gửi lại tin nhắn nhận được. Ở đây chúng tôi sẽ thêm một số thời gian (timing): tic và toc sẽ giữ tin nhắn trong 1 giây mô phỏng trước khi gửi lại. Trong OMNeT ++, thời gian như vậy đạt được bằng cách mô-đun gửi thông điệp đến chính nó. Những thông điệp như vậy được gọi là thông điệp của chính nó (self-messages) (nhưng chỉ do cách chúng được sử dụng, nếu không chúng là đối tượng thông điệp bình thường).

Chúng tôi đã thêm hai biến [cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \* event và tictocMsg vào lớp, để ghi nhớ thông điệp chúng tôi sử dụng để định thời gian và thông điệp có độ trễ xử lý mà chúng tôi đang mô phỏng.

class [Txc6](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc6.html) : public [cSimpleModule](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html)

{

private:

[cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*[event](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc6.html#ad95df6e05498b9e86c080b570442c996); // pointer to the event object which we'll use for timing

[cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*[tictocMsg](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc6.html#a1a5623c53fcd6e163378eb39e2a87d46); // variable to remember the message until we send it back

public:

Chúng tôi "gửi" các thông điệp của chính nó bằng hàm scheduleAt (), chỉ định khi nào nó sẽ được gửi trở lại mô-đun.

scheduleAt(simTime()+1.0, event);

Trong handleMessage() bây giờ chúng ta phải phân biệt xem một thông điệp mới đã đến qua cổng đầu vào hay thông điệp của chính nó quay lại (bộ đếm thời gian đã hết hạn). Ở đây chúng tôi đang sử dụng

if (msg == event)

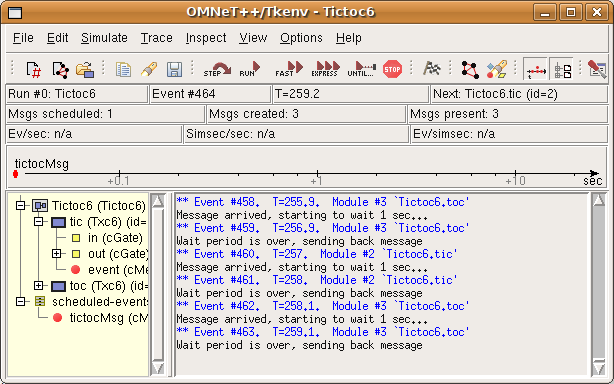
nhưng chúng tôi có thể đã viết

if (msg-> isSelfMessage ())

cũng được.

Chúng tôi đã loại bỏ bộ đếm, để giữ cho mã nguồn nhỏ.

Kết quả của việc chạy mô phỏng có thể được nhìn thấy bên dưới.



## Bước 7: Số ngẫu nhiên và tham số

Trong bước này, chúng tôi sẽ giới thiệu các số ngẫu nhiên. Chúng tôi thay đổi độ trễ từ 1s thành một giá trị ngẫu nhiên có thể được đặt từ tệp NED hoặc từ omnetpp.ini. Tham số mô-đun có thể trả về các biến ngẫu nhiên; tuy nhiên, để sử dụng tính năng này, chúng ta phải đọc tham số trong handleMessage() mỗi khi sử dụng.

// 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](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Message arrived, starting to wait " << delay << " secs...\n";

tictocMsg = msg;

scheduleAt(simTime()+delay, event);

Ngoài ra, chúng ta sẽ "mất" (xóa) gói tin với một xác suất (mã cứng) nhỏ.

if ([uniform](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersCont.html#ga4a63082fca94b8abf6d6164aea972a0f)(0,1) < 0.1)

{

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "\"Losing\" message\n";

delete msg;

Chúng tôi sẽ chỉ định các tham số trong omnetpp.ini:

Tictoc7.tic.delayTime = [exponential](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersCont.html#ga1f77033d9815fe55a847f5aefdbbb1a7)(3s)

Tictoc7.toc.delayTime = [truncnormal](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersCont.html#ga80b20109dbe65191c8404c58cef90917)(3s,1s)

Bạn có thể thử rằng bất kể bạn chạy lại mô phỏng bao nhiêu lần (hoặc khởi động lại nó, Simulate|Rebuild mục menu mạng), bạn sẽ nhận được kết quả chính xác. Điều này là do OMNeT ++ sử dụng thuật toán xác định (theo mặc định là Mersenne Twister RNG) để tạo các số ngẫu nhiên và khởi tạo nó cho cùng một seed. Điều này rất quan trọng đối với các mô phỏng có thể tái tạo. Bạn có thể thử nghiệm với các seed khác nhau nếu bạn thêm các dòng sau vào omnetpp.ini:

[General]

seed-0-mt=532569 # or any other 32-bit value

Từ cú pháp, bạn có thể đoán rằng OMNeT ++ hỗ trợ nhiều hơn một RNG. Đúng vậy, tuy nhiên, tất cả các mô hình trong hướng dẫn này đều sử dụng RNG 0.

## Bước 8: Thời gian trễ hết giờ, hủy hẹn giờ

Để tiến gần hơn một bước đến việc lập mô hình các giao thức mạng, chúng ta hãy chuyển đổi mô hình của mình thành một mô phỏng dừng và chờ. Lần này chúng ta sẽ có các lớp riêng biệt cho tic và toc. Kịch bản cơ bản tương tự như các kịch bản trước: tic và toc sẽ chuyển một tin nhắn cho nhau. Tuy nhiên, toc sẽ "mất" tin nhắn với một số xác suất khác, và trong trường hợp đó, tic sẽ phải gửi lại.

Đây là mã của toc:

void [Toc8::handleMessage](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classToc8.html#a0180ea73c49a6c8fc684a455a962ecd9)([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*msg)

{

if ([uniform](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersCont.html#ga4a63082fca94b8abf6d6164aea972a0f)(0,1) < 0.1)

{

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "\"Losing\" message.\n";

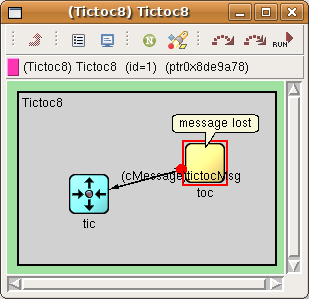
[bubble](https://doc.omnetpp.org/omnetpp4/api//classcComponent.html#a8b1160aa2bb4f94af176df5da5075afb)("message lost"); // making animation more informative...

delete msg;

}

else

Nhờ lệnh gọi bubble() trong mã, toc sẽ hiển thị chú thích bất cứ khi nào nó thả tin nhắn.



Vì vậy, tic sẽ bắt đầu hẹn giờ bất cứ khi nào nó gửi tin nhắn. Khi bộ hẹn giờ hết hạn, chúng tôi sẽ cho rằng tin nhắn đã bị mất và gửi một tin nhắn khác. Nếu phản hồi của toc đến, bộ hẹn giờ phải được hủy bỏ. Bộ đếm thời gian sẽ là (cái gì khác?) một thông điệp của chính nó.

ScheduleAt(simTime() + timeout, timeoutEvent);

Việc hủy hẹn giờ sẽ được thực hiện bằng lệnh cancelEvent(). Lưu ý rằng điều này không ngăn chúng tôi có thể sử dụng lặp đi lặp lại cùng một thông báo hết thời gian.

cancelEvent (timeoutEvent);

## Bước 9: Truyền lại cùng một thông điệp

Trong bước này, chúng tôi tinh chỉnh mô hình trước đó. Ở đó, chúng tôi vừa tạo một gói khác nếu chúng tôi cần truyền lại. Điều này không sao cả vì gói tin không chứa nhiều, nhưng trong cuộc sống thực, việc giữ lại một bản sao của gói tin gốc để chúng ta có thể gửi lại mà không cần phải xây dựng lại nó thường thực tế hơn.

Những gì chúng tôi làm ở đây là giữ gói tin gốc và chỉ gửi các bản sao của nó. Chúng tôi xóa bản gốc khi xác nhận của toc đến. Để giúp xác minh mô hình một cách trực quan dễ dàng hơn, chúng tôi sẽ bao gồm một số thứ tự thông điệp các tên thông điệp.

Để tránh phát triển quá lớn của handleMessage(), chúng tôi sẽ đặt mã tương ứng vào hai hàm mới, createNewMessage() và sendCopyOf() và gọi chúng từ handleMessage().

Các hàm:

[cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*[Tic9::generateNewMessage](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTic9.html#a729ebf1c6017dbe15c1b277ff32d9e9f)()

{

// Generate a message with a different name every time.

char msgname[20];

sprintf(msgname, "tic-%d", ++[seq](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTic9.html#a8cfab4487172256992228c7e5c58eadc));

[cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*msg = new [cMessage](https://doc.omnetpp.org/omnetpp4/api//classcOwnedObject.html#af46e516dfdf6a913fe913c4c2e496cf2)(msgname);

return msg;

}

void [Tic9::sendCopyOf](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTic9.html#a3f5dff4f46521d7f01a1fea3194dce8a)([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*msg)

{

// Duplicate message and send the copy.

[cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*copy = ([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*) msg->[dup](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html#a231c291bf665945146df73676f591bc2)();

[send](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html#a94590ba6481116f29f4c43d00733deea)(copy, "out");

}

# 3. Biến nó thành một mạng thực sự

## Bước 10: Nhiều hơn hai nút

Bây giờ chúng ta sẽ thực hiện một bước quan trọng: tạo một số mô-đun tic và kết nối chúng thành một mạng. Hiện tại, chúng ta sẽ đơn giản hóa những gì họ làm: một trong các nút tạo ra một thông điệp và các nút khác tiếp tục ném nó theo các hướng ngẫu nhiên cho đến khi nó đến một nút đích được xác định trước.

Tệp NED sẽ cần một vài thay đổi. Trước hết, mô-đun Txc sẽ cần có nhiều cổng đầu vào và đầu ra:

simple [Txc10](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc10.html)

{

parameters:

@display("i=block/routing");

gates:

input in[]; // declare in[] and out[] to be vector gates

output out[];

}

[] biến các cổng thành các vectơ cổng. Kích thước của vector (số lượng cổng) sẽ được xác định nơi chúng ta sử dụng Txc để xây dựng mạng.

network Tictoc10

{

submodules:

tic[6]: [Txc10](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc10.html);

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++;

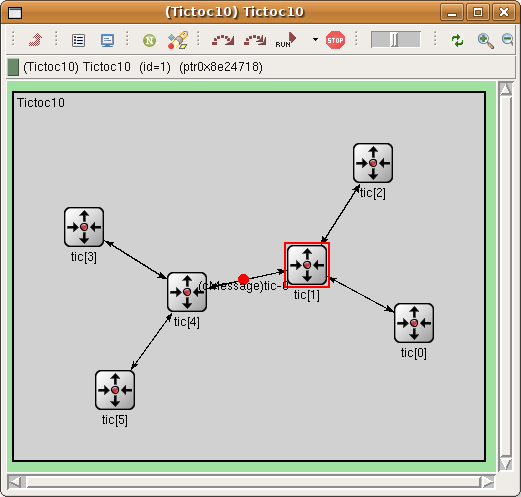
tic[4].out++ --> { delay = 100ms; } --> tic[5].in++;

tic[4].in++ <-- { delay = 100ms; } <-- tic[5].out++;

}

Ở đây, chúng tôi đã tạo 6 mô-đun dưới dạng vectơ mô-đun và kết nối chúng.

Cấu trúc liên kết kết quả trông giống như sau:



Trong phiên bản này, tic [0] sẽ tạo ra thông điệp để gửi đi. Điều này được thực hiện trong intialize(), với sự trợ giúp của hàm getIndex () trả về chỉ mục của mô-đun trong vectơ.

Phần lớn của mã là hàm forwardMessage() mà chúng ta gọi từ handleMessage () bất cứ khi nào có thông điệp đến nút. Nó vẽ một số cổng ngẫu nhiên (size () là kích thước của vector cổng) và gửi thông điệp trên cổng đó.

void [Txc10::forwardMessage](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc10.html#ac0a6e2b462a2615f6c0ce388ceec62a4)([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*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](https://doc.omnetpp.org/omnetpp4/api//classcModule.html#ac63b0b9c805c5f335d274195f2fdb28c)("out");

int k = [intuniform](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersDiscr.html#gaee8ce6ae7e43b06e968a023f25c91b4f)(0,n-1);

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Forwarding message " << msg << " on port out[" << k << "]\n";

[send](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html#a94590ba6481116f29f4c43d00733deea)(msg, "out", k);

}

Khi tin nhắn đến tic [3], thì handleMessage () của nó sẽ xóa tin nhắn.

*Bài tập: bạn sẽ nhận thấy rằng cách "định tuyến" đơn giản này không hiệu quả lắm: thường gói tin liên tục nảy giữa hai nút một lúc trước khi nó được gửi đến một hướng khác. Điều này có thể được cải thiện phần nào nếu các nút không gửi lại gói tin cho người gửi. Thực hiện điều này. Gợi ý:* [*cMessage :: getArrivalGate ()*](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html#a1aeaf09baf5a3584c17cd64edccc1037) *,* [*cGate :: getIndex ()*](https://doc.omnetpp.org/omnetpp4/api//classcGate.html#a0ec58c0c39b875a03cfc18bd20059d3b) *. Lưu ý rằng nếu thông điệp không đến qua cổng mà là một thông điệp của chính nó, thì getArrivalGate () trả về NULL.*

## Bước 11: Các kênh và định nghĩa loại bên trong

Định nghĩa mạng mới của chúng tôi ngày càng phức tạp và dài, đặc biệt là phần kết nối. Hãy cố gắng đơn giản hóa nó. Điều đầu tiên chúng ta nhận thấy là các kết nối luôn sử dụng cùng một tham số delay. Có thể tạo kiểu cho các kết nối (chúng được gọi là kênh) tương tự như các mô-đun đơn giản. Chúng ta nên tạo một kiểu kênh chỉ định tham số trễ và chúng ta sẽ sử dụng kiểu đó cho tất cả các kết nối trong mạng.

network Tictoc11

{

types:

channel Channel extends ned.DelayChannel {

delay = 100ms;

}

submodules:

Như bạn đã nhận thấy, chúng tôi đã xác định loại kênh mới bên trong định nghĩa mạng bằng cách thêm một phần types. Định nghĩa kiểu này chỉ hiển thị bên trong mạng. Nó được gọi là kiểu cục bộ hoặc kiểu bên trong. Bạn cũng có thể sử dụng các mô-đun đơn giản làm loại bên trong, nếu muốn.

**Chú ý:**

Chúng tôi đã tạo kênh bằng cách chuyên biệt hóa kênh DelayChannel tích hợp sẵn. (Có thể tìm thấy các kênh tích hợp bên trong gói ned. Đó là lý do tại sao chúng tôi sử dụng tên loại đầy đủ ned.DelayChannel) sau từ khóa extends.

Bây giờ hãy kiểm tra xem phần connections đã thay đổi như thế nào .

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++;

}

Như bạn thấy, chúng tôi chỉ xác định tên kênh bên trong định nghĩa kết nối. Điều này cho phép dễ dàng thay đổi tham số trễ cho toàn mạng.

## Bước 12: Sử dụng kết nối hai chiều

Nếu chúng ta kiểm tra phần connections nhiều hơn một chút, chúng ta sẽ nhận ra rằng mỗi cặp nút được kết nối với hai kết nối. Một cho mỗi hướng. OMNeT ++ 4 hỗ trợ kết nối hai chiều, vì vậy hãy sử dụng chúng.

Trước hết, chúng ta phải xác định cổng hai chiều (hoặc gọi là inout) thay vì cổng input và output riêng chúng tôi sử dụng trước đây.

simple [Txc12](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc12.html)

{

parameters:

@display("i=block/routing");

gates:

inout gate[]; // declare two way connections

}

Phần connections mới sẽ giống như sau:

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++;

}

Chúng tôi đã sửa đổi tên cổng nên chúng tôi phải thực hiện một số sửa đổi đối với mã C ++.

void [Txc12::forwardMessage](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc12.html#a0776d3640c16fdfcaa112ad4eb987415)([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*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](https://doc.omnetpp.org/omnetpp4/api//classcModule.html#ac63b0b9c805c5f335d274195f2fdb28c)("gate");

int k = [intuniform](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersDiscr.html#gaee8ce6ae7e43b06e968a023f25c91b4f)(0,n-1);

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "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](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html#a94590ba6481116f29f4c43d00733deea)(msg, "gate$o", k);

}

**Chú ý:**

Hậu tố $i và $o đặc biệt sau tên cổng cho phép chúng ta sử dụng hai hướng của kết nối riêng biệt .

## Bước 13: Xác định lớp thông điệp

Trong bước này, địa chỉ đích không còn được mã hóa cứng tic[3] nữa - chúng tôi vẽ một điểm đến ngẫu nhiên và chúng tôi sẽ thêm địa chỉ đích vào thông điệp.

Cách tốt nhất là lớp con [cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) và thêm đích làm thành viên dữ liệu. Mã hóa thủ công lớp thông điệp thường tẻ nhạt vì nó chứa nhiều mã soạn sẵn, vì vậy chúng tôi để OMNeT ++ tạo lớp cho chúng tôi. Đặc tả lớp thông điệp trong tictoc13.msg:

message TicTocMsg13

{

int source;

int destination;

int hopCount = 0;

}

Makefile được thiết lập để trình biên dịch thông điệp, opp\_msgc được gọi và nó tạo ra tictoc13\_m.h và tictoc13\_m.cc từ khai báo thông báo. Chúng sẽ chứa một lớp con TicTocMsg13 được tạo ra từ [cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) ; lớp sẽ có các phương thức getter và setter cho mọi trường.

Chúng tôi sẽ bao gồm tictoc13\_m.h vào mã C ++ và chúng tôi có thể sử dụng TicTocMsg13 như bất kỳ lớp khác.

#include "tictoc13\_m.h"

Ví dụ, chúng tôi sử dụng các dòng sau trong createMessage() để tạo thông báo và điền vào các trường của nó.

TicTocMsg13 \*msg = new TicTocMsg13(msgname);

msg->setSource(src);

msg->setDestination(dest);

return msg;

Sau đó, handleMessage () bắt đầu như sau:

void [Txc13::handleMessage](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc13.html#a0453de5127489ee026d598cdb64f28a8)([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*msg)

{

TicTocMsg13 \*ttmsg = [check\_and\_cast](https://doc.omnetpp.org/omnetpp4/api//group__Functions.html#gac05a322b9912da77f8b797eeab7805f4)<TicTocMsg13 \*>(msg);

if (ttmsg->getDestination()==[getIndex](https://doc.omnetpp.org/omnetpp4/api//classcModule.html#a604e26394d77938a860eeb4452f25606)())

Trong đối số của handleMessage(), chúng ta nhận được thông báo dưới dạng con trỏ [cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*. Tuy nhiên, chúng ta chỉ có thể truy cập các trường của nó được xác định trong TicTocMsg13 nếu chúng ta truyền msg sang TicTocMsg13 \*. Plain C-style cast ( (TicTocMsg13 \*)msg) không an toàn vì nếu thông điệp *không phải* là TicTocMsg13 sau tất cả chương trình sẽ sụp đổ, gây ra lỗi rất khó khám phá.

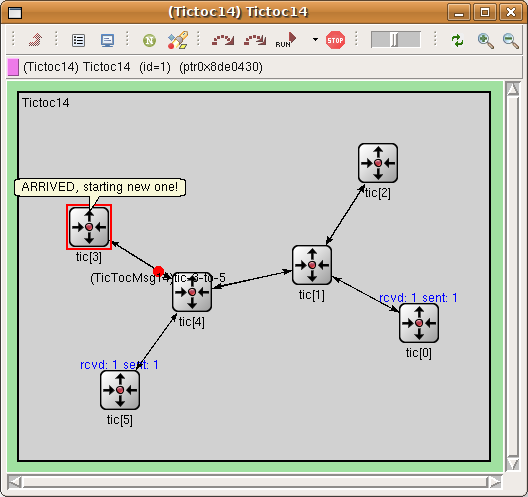
C ++ cung cấp một giải pháp được gọi là dynamic\_cast. Ở đây chúng tôi sử dụng [check\_and\_cast <> ()](https://doc.omnetpp.org/omnetpp4/api//group__Functions.html#gac05a322b9912da77f8b797eeab7805f4) được cung cấp bởi OMNeT ++: nó cố gắng truyền con trỏ qua dynamic\_cast và nếu không thành công, nó sẽ dừng mô phỏng với thông báo lỗi, tương tự như sau:



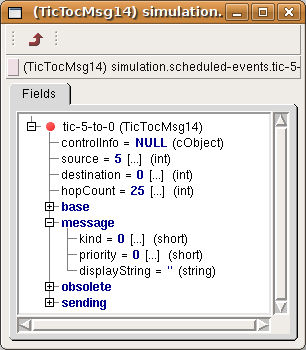
Trong dòng tiếp theo, chúng tôi kiểm tra xem địa chỉ đích có giống với địa chỉ của nút hay không. Hàm thành viên getIndex() trả về chỉ mục của mô-đun trong vectơ mô-đun con (hãy nhớ rằng, trong tệp NED, chúng tôi đã khai báo nó là tic: [**Txc13**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc13.html)[6], vì vậy các nút của chúng tôi có địa chỉ 0..5).

Để làm cho mô hình thực thi lâu hơn, sau khi một thông điệp đến đích của nó, nút đích sẽ tạo ra một thông điệp khác với địa chỉ đích ngẫu nhiên, v.v.

Khi bạn chạy mô hình, nó sẽ giống như sau:



Bạn có thể bấm đúp vào thư để mở trình kiểm tra cho chúng. (Bạn sẽ phải tạm thời dừng mô phỏng cho điều đó hoặc phải xử lý chuột rất nhanh). Cửa sổ thanh tra hiển thị nhiều thông tin hữu ích; các trường thông báo có thể được nhìn thấy trên trang Contents.



*Bài tập: Trong mô hình này, chỉ có một thông điệp được thực hiện tại bất kỳ thời điểm nào: các nút chỉ tạo ra một thông điệp khi một thông điệp khác đến với chúng. Chúng tôi đã làm theo cách này để làm cho việc theo dõi mô phỏng dễ dàng hơn. Thay đổi lớp mô-đun để thay vào đó, nó tạo thông điệp định kỳ. Khoảng thời gian giữa các thông điệp phải là một tham số mô-đun, trả về các số ngẫu nhiên được phân phối theo cấp số nhân.*

# 4. Thêm thu thập thống kê

## Bước 14: Hiển thị số lượng gói tin đã gửi / nhận

Để có cái nhìn tổng quan về thời gian chạy có bao nhiêu thông điệp mà mỗi nút đã gửi hoặc nhận, chúng tôi đã thêm hai bộ đếm vào lớp mô-đun: numSent và numReceive.

class [Txc14](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc14.html) : public [cSimpleModule](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html)

{

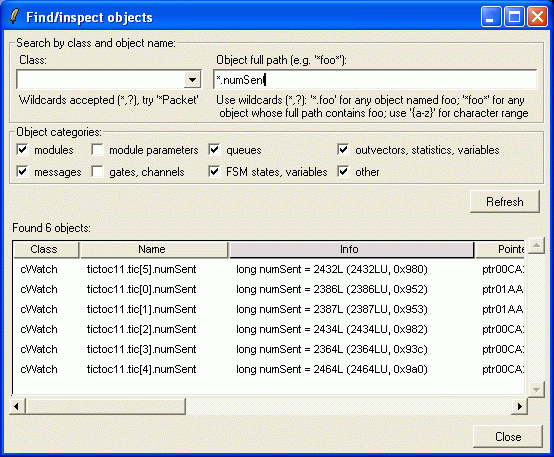
private:

long [numSent](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc14.html#a09037ffeeb42dc57de98a356744f462c);

long [numReceived](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc14.html#a817d6ff31634cd292f12d16ed7ed3b1f);

protected:

Chúng được đặt thành 0 và WATCH'ed trong phương thức intialize(). Bây giờ chúng ta có thể sử dụng hộp thoại Find/inspect objects (menu Inspect; nó cũng nằm trên thanh công cụ) để tìm hiểu có bao nhiêu gói được gửi hoặc nhận bởi các nút khác nhau.



Đúng là trong mô hình mô phỏng cụ thể này, các con số sẽ gần giống nhau, vì vậy bạn chỉ có thể học từ chúng để biết rằng [intuniform()](https://doc.omnetpp.org/omnetpp4/api//group__RandomNumbersDiscr.html#gaee8ce6ae7e43b06e968a023f25c91b4f) hoạt động đúng. Nhưng trong các mô phỏng thực tế, nó có thể rất hữu ích khi bạn có thể nhanh chóng có được cái nhìn tổng quan về trạng thái của các nút khác nhau trong mô hình.

Nó cũng có thể được sắp xếp rằng thông tin này xuất hiện phía trên các biểu tượng mô-đun. Các thẻ chuỗi hiển thị t= chỉ định các văn bản; chúng ta chỉ cần sửa đổi chuỗi hiển thị trong thời gian chạy. Đoạn mã sau thực hiện công việc:

if ([ev](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#gad68b7777ea70ef452e0aadb4ea2df455).isGUI())

updateDisplay();

void [Txc14::updateDisplay](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc14.html#a90062f9850bffd814b3979b54c805989)()

{

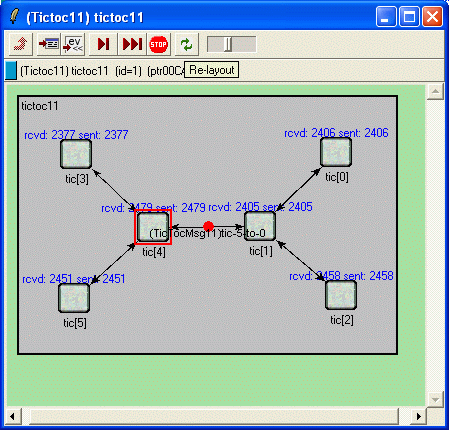
char buf[40];

sprintf(buf, "rcvd: %ld sent: %ld", [numReceived](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc14.html#a817d6ff31634cd292f12d16ed7ed3b1f), [numSent](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc14.html#a09037ffeeb42dc57de98a356744f462c));

[getDisplayString](https://doc.omnetpp.org/omnetpp4/api//classcComponent.html#abc85b4eac8e4fc6dbcb91844ea6dd670)().[setTagArg](https://doc.omnetpp.org/omnetpp4/api//classcDisplayString.html#ae021b0e1e95b8da261455a66f286e7ab)("t",0,buf);

}

Và kết quả như sau:



## Bước 15: Thêm thu thập thống kê

Nhân mô phỏng OMNeT ++ có thể tự động ghi lại nhật ký chi tiết về việc trao đổi thông điệp của bạn bằng cách thiết lập

record-eventlog = true

tùy chọn cấu hình trong tệp omnetpp.ini. Sau này IDE có thể hiển thị tệp nhật ký này (xem: Nhật [**ký sự kiện kết thúc biểu đồ tuần tự**](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/part5.html#logs) ).

**Chú ý:**

Tệp nhật ký kết quả có thể khá lớn, vì vậy chỉ nên bật tính năng này nếu bạn thực sự cần.

Mô hình mô phỏng trước đó làm điều gì đó đủ thú vị để chúng ta có thể thu thập một số thống kê. Ví dụ: bạn có thể quan tâm đến số bước nhảy trung bình mà một thông điệp phải di chuyển trước khi đến đích.

Chúng tôi sẽ ghi lại trong số bước nhảy của mọi thông diệpd khi đến một vector đầu ra (một chuỗi các cặp (thời gian, giá trị), một loại chuỗi thời gian). Chúng tôi cũng tính toán giá trị trung bình, độ lệch chuẩn, giá trị tối thiểu, tối đa trên mỗi nút và ghi chúng vào một tệp ở cuối mô phỏng. Sau đó, chúng tôi sẽ sử dụng các công cụ từ OMNeT ++ IDE để phân tích các tệp đầu ra.

Đối với điều đó, chúng tôi thêm một đối tượng vectơ đầu ra (sẽ ghi dữ liệu vào Tictoc15-0.vec) và một đối tượng biểu đồ (cũng tính toán giá trị trung bình, v.v.) vào lớp.

class [Txc15](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc15.html) : public [cSimpleModule](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html)

{

private:

long [numSent](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc15.html#a83a66970e1e6850b5746b32ce5f84bf7);

long [numReceived](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc15.html#a2c5e6be52c4f766354a80f0c1c724d05);

[cLongHistogram](https://doc.omnetpp.org/omnetpp4/api//classcLongHistogram.html) [hopCountStats](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc15.html#ad912c2b914f7f950dfbf8c2b20b36b5b);

[cOutVector](https://doc.omnetpp.org/omnetpp4/api//classcOutVector.html) [hopCountVector](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc15.html#a3cb52e4a25a813b46b7210e8d65e606a);

protected:

Khi một thông điệp đến nút đích, chúng tôi cập nhật số liệu thống kê. Mã sau đã được thêm vào handleMessage ():

hopCountVector.[record](https://doc.omnetpp.org/omnetpp4/api//classcOutVector.html#a8a7dddb86f9ea261a94f52bc3acf290d)(hopcount);

hopCountStats.[collect](https://doc.omnetpp.org/omnetpp4/api//classcLongHistogram.html#aa17adc3aeed42e79f81316bc43c3df27)(hopcount);

lời gọi hopCountVector.record() ghi dữ liệu vào Tictoc15-0.vec. Với một mô hình mô phỏng lớn hoặc thời gian thực thi lâu, Tictoc15-0.vectệp có thể phát triển rất lớn. Để xử lý tình huống này, bạn có thể tắt/bật vector trong omnetpp.ini và bạn cũng có thể chỉ định khoảng thời gian mô phỏng mà bạn quan tâm (dữ liệu được ghi bên ngoài khoảng thời gian này sẽ bị loại bỏ.)

Khi bạn bắt đầu một mô phỏng mới, tệp Tictoc15-0.vec/sca hiện có sẽ bị xóa.

Dữ liệu vô hướng (được thu thập bởi đối tượng biểu đồ trong mô phỏng này) phải được ghi lại theo cách thủ công, trong hàm finish(). finish() được gọi khi hoàn thành mô phỏng thành công, tức là không phải khi nó dừng lại do lỗi. Các lời gọi recordScalar() trong đoạn mã dưới đây được ghi vào tệpTictoc15-0.sca.

void [Txc15::finish](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc15.html#ae2cd699c10ee0549c8bb2331be21dce1)()

{

// This function is called by OMNeT++ at the end of the simulation.

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Sent: " << numSent << endl;

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Received: " << numReceived << endl;

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Hop count, min: " << hopCountStats.[getMin](https://doc.omnetpp.org/omnetpp4/api//classcStdDev.html#ab682269238bc1d82f8fa870de9351e0a)() << endl;

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Hop count, max: " << hopCountStats.[getMax](https://doc.omnetpp.org/omnetpp4/api//classcStdDev.html#a9dae2eb597d46cb90722048590f1385d)() << endl;

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Hop count, mean: " << hopCountStats.[getMean](https://doc.omnetpp.org/omnetpp4/api//classcStdDev.html#a3fde16a03e479d5dd00d38fe0847c16b)() << endl;

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Hop count, stddev: " << hopCountStats.[getStddev](https://doc.omnetpp.org/omnetpp4/api//classcStdDev.html#a73d4b346643b657814894367cd26c82d)() << endl;

[recordScalar](https://doc.omnetpp.org/omnetpp4/api//classcComponent.html#a733836a2f7dec4e1fb5c659df92a35aa)("#sent", numSent);

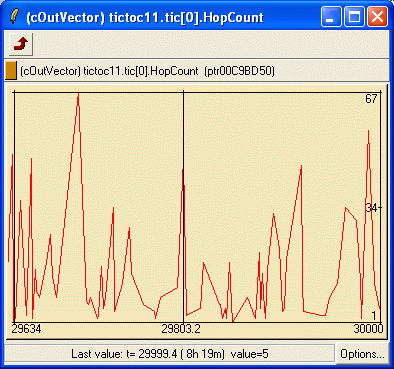
[recordScalar](https://doc.omnetpp.org/omnetpp4/api//classcComponent.html#a733836a2f7dec4e1fb5c659df92a35aa)("#received", numReceived);

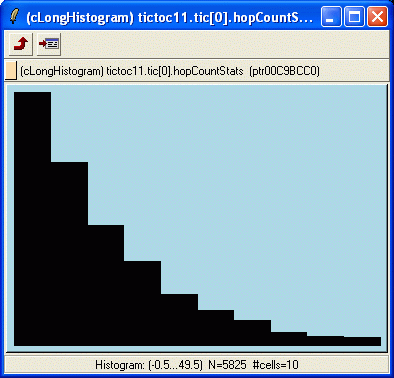
hopCountStats.[recordAs](https://doc.omnetpp.org/omnetpp4/api//classcStatistic.html#afb43a821ce893fe61db0172e0d3fbb3d)("hop count");

}

Các tệp được lưu trữ trong thư mục con results/.

Bạn cũng có thể xem dữ liệu trong quá trình mô phỏng. Trong trang Contents của trình kiểm tra mô-đun, bạn sẽ tìm thấy các đối tượng hopCountStats và hopCountVector và bạn có thể mở trình kiểm tra của chúng (nhấp đúp). Ban đầu chúng sẽ trống - hãy chạy mô phỏng ở chế độ Fast (hoặc thậm chí Express) để có đủ dữ liệu được hiển thị. Sau một thời gian, bạn sẽ nhận được một cái gì đó như thế này:





Khi bạn nghĩ rằng đã thu thập đủ dữ liệu, bạn có thể dừng mô phỏng và sau đó chúng tôi sẽ phân tích các tệp kết quả ( Tictoc15-0.vecvà Tictoc15-0.sca) ngoại tuyến. Bạn sẽ cần chọn Simulate|Call finish() từ menu (hoặc nhấp vào nút thanh công cụ tương ứng) trước khi thoát - điều này sẽ làm cho các hàm finish() chạy và dữ liệu được ghi vào Tictoc15-0.sca.

## Bước 16: Thu thập thống kê mà không cần sửa đổi mô hình của bạn

Trong bước trước, chúng tôi đã thêm thu thập thống kê vào mô hình của mình. Mặc dù chúng ta có thể tính toán và lưu bất kỳ giá trị nào mà chúng ta muốn, nhưng thường thì không biết tại thời điểm viết mô hình, người dùng cuối sẽ cần dữ liệu gì.

OMNeT ++ 4.1 cung cấp một cơ chế bổ sung để ghi lại các giá trị và sự kiện. Bất kỳ mô hình nào cũng có thể phát ra 'tín hiệu' có thể mang một giá trị hoặc một đối tượng. Người viết mô hình chỉ phải quyết định những tín hiệu sẽ phát ra, những dữ liệu nào để gắn vào chúng và khi nào phát ra chúng. Người dùng cuối có thể đính kèm 'người nghe' vào những tín hiệu có thể xử lý hoặc ghi lại các mục dữ liệu này. Bằng cách này, mã mô hình không phải chứa bất kỳ mã nào cụ thể cho thu thập thống kê và người dùng cuối có thể tự do thêm số liệu thống kê bổ sung mà không cần nhìn vào mã C ++.

Chúng tôi sẽ viết lại thu thập thống kê được giới thiệu trong bước cuối cùng để sử dụng các tín hiệu. Trước hết, chúng tôi có thể xóa tất cả các biến liên quan đến thống kê khỏi mô-đun của mình một cách an toàn. Không cần thiết phải có các lớp [cOutVector](https://doc.omnetpp.org/omnetpp4/api//classcOutVector.html) và [cLongHistogram](https://doc.omnetpp.org/omnetpp4/api//classcLongHistogram.html). Chúng tôi sẽ chỉ cần một tín hiệu duy nhất mang hopCount củathông điệp tại thời điểm thông điệp đến đích.

Đầu tiên chúng ta cần xác định tín hiệu của mình. Đây arrivalSignal chỉ là một số nhận dạng có thể được sử dụng sau này để dễ dàng tham khảo tín hiệu của chúng tôi.

class [Txc16](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc16.html) : public [cSimpleModule](https://doc.omnetpp.org/omnetpp4/api//classcSimpleModule.html)

{

private:

simsignal\_t [arrivalSignal](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc16.html#a8794039dc73e44779d67a9bedec79e80);

protected:

Chúng ta phải đăng ký tất cả các tín hiệu trước khi sử dụng chúng. Nơi tốt nhất để làm điều này là phương thức initialize() của mô-đun.

void [Txc16 :: khởi tạo](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc16.html#a79a503b3ea944bda9376e0467a418767) ()

{

đếnSignal = [registerSignal](https://doc.omnetpp.org/omnetpp4/api//classcComponent.html#aeaa2a3ca2cb5f2c798ff0bea82f38a91) ( "đến" );

// Mô-đun 0 gửi thông báo đầu tiên

if ( [getIndex](https://doc.omnetpp.org/omnetpp4/api//classcModule.html#a604e26394d77938a860eeb4452f25606) () == 0)

Bây giờ chúng ta có thể phát ra tín hiệu của mình, khi thông điệp đã đến nút đích.

void [Txc16::handleMessage](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc16.html#a025844a47db5345929a4db351953ca87)([cMessage](https://doc.omnetpp.org/omnetpp4/api//classcMessage.html) \*msg)

{

TicTocMsg16 \*ttmsg = [check\_and\_cast](https://doc.omnetpp.org/omnetpp4/api//group__Functions.html#gac05a322b9912da77f8b797eeab7805f4)<TicTocMsg16 \*>(msg);

if (ttmsg->getDestination()==[getIndex](https://doc.omnetpp.org/omnetpp4/api//classcModule.html#a604e26394d77938a860eeb4452f25606)())

{

// Message arrived

int hopcount = ttmsg->getHopCount();

// send a signal

[emit](https://doc.omnetpp.org/omnetpp4/api//classcComponent.html#a29022c0570003b4722ebc58df66967eb)(arrivalSignal, hopcount);

[EV](https://doc.omnetpp.org/omnetpp4/api//group__Envir.html#ga650ef3eff8a2900bef69dae29c05d2dd) << "Message " << ttmsg << " arrived after " << hopcount << " hops.\n";

Vì chúng ta không phải lưu hoặc lưu trữ bất cứ thứ gì theo cách thủ công, nên phương thức finish() này có thể bị xóa. Chúng tôi không cần nó nữa.

Bước cuối cùng là chúng ta phải xác định tín hiệu phát ra cũng trong tệp NED. Khai báo các tín hiệu trong tệp NED cho phép bạn có tất cả thông tin về mô-đun của mình ở một nơi. Bạn sẽ thấy các thông số mà nó cần, các cổng đầu vào và đầu ra của nó, cũng như các tín hiệu và số liệu thống kê mà nó cung cấp.

simple [Txc16](https://doc.omnetpp.org/omnetpp4/tictoc-tutorial/classTxc16.html)

{

parameters:

@signal[arrival](type="long");

@statistic[hopCount](title="hop count"; source="arrival"; record=vector,stats; interpolationmode=none);

@display("i=block/routing");

Bây giờ chúng tôi cũng có thể xác định một thống kê nên được thu thập theo mặc định. Ví dụ trước của chúng tôi đã thu thập số liệu thống kê (giá trị nhỏ nhất, giá trị lớn nhất, giá trị trung bình, số lượng, v.v.) về số bước nhảy của các thông điệp đến, vì vậy chúng ta hãy thu thập cùng một dữ liệu ở đây.

Từ khóa source quy định cụ thể chính tín hiệu chúng tôi muốn thống kê của chúng tôi để đính kèm vào. Từ khóa record có thể được sử dụng để nói những gì nên được thực hiện với các dữ liệu nhận được. Trong trường hợp của chúng tôi, chúng tôi chỉ ra rằng mỗi giá trị phải được lưu trong một tệp vector (vector) và chúng tôi cũng cần tính toán giá trị nhỏ nhất, giá trị lớn nhất, giá trị trung bình, số lượng, v.v. (thống kê). (LƯU Ý: stats chỉ là cách viết tắt của giá trị nhỏ nhất, giá trị lớn nhất, giá trị trung bình, số lượng v.v.) Với bước này, chúng ta đã hoàn thành mô hình của mình.

Bây giờ chúng tôi mới nhận ra rằng chúng tôi muốn xem một biểu đồ của hopCount trên mô-đun tic [1]. Mặt khác, chúng tôi thiếu khả năng lưu trữ trên đĩa và chúng tôi không quan tâm đến việc có dữ liệu vectơ cho ba mô-đun đầu tiên tic 0,1,2. Không vấn đề gì. Chúng tôi có thể thêm biểu đồ của mình và xóa bản ghi vectơ không cần thiết mà không cần chạm vào tệp C ++ hoặc NED. Chỉ cần mở tệp INI và sửa đổi bản ghi thống kê:

[Config Tictoc16]

network = Tictoc16

\*\*.tic[1].hopCount.result-recording-modes = +histogram

\*\*.tic[0..2].hopCount.result-recording-modes = -vector

Chúng tôi có thể định cấu hình một loạt các thống kê mà không cần nhìn vào mã C ++, miễn là mô hình gốc phát ra các tín hiệu cần thiết cho chúng tôi.

Bây giờ chúng tôi đã thu thập số liệu thống kê của mình, hãy xem và phân tích chúng trong IDE.

# 5. Trực quan hóa kết quả với OMNeT ++ IDE

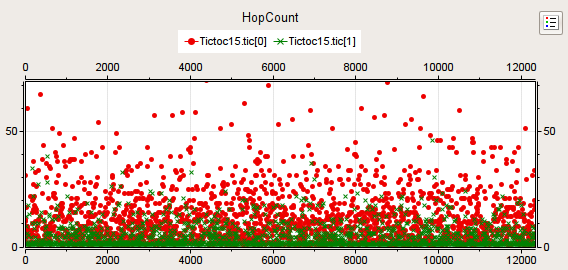
## Trực quan hóa các đầu ra vectơ và vô hướng

OMNeT ++ IDE có thể giúp bạn phân tích kết quả của mình. Nó hỗ trợ lọc, xử lý và hiển thị dữ liệu vectơ và vô hướng, đồng thời có thể hiển thị biểu đồ. Các sơ đồ sau đã được tạo bằng công cụ Phân tích Kết quả của IDE.

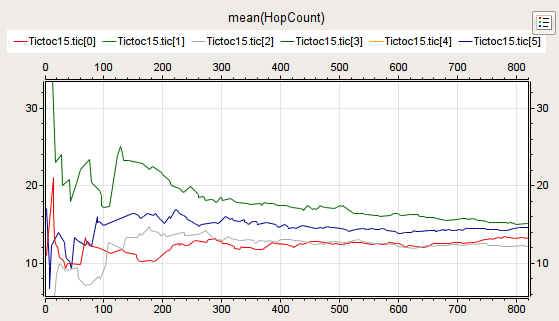
**Chú ý:**

Để biết thêm thông tin về khả năng xử lý và lập biểu đồ, vui lòng tham khảo **Hướng dẫn sử dụng OMNeT ++** .

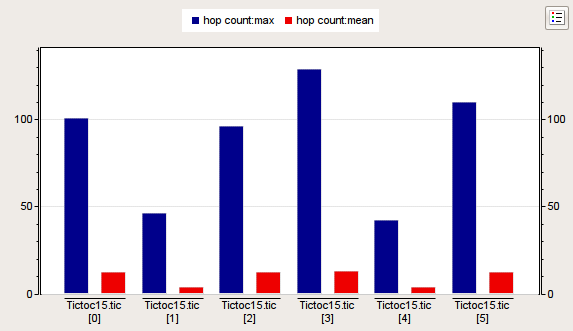
Mô hình cuối cùng của chúng tôi ghi lại hopCountthông điệp mỗi khi thông điệp đến đích. Biểu đồ sau đây cho thấy các vectơ này cho các nút 0 và 1.



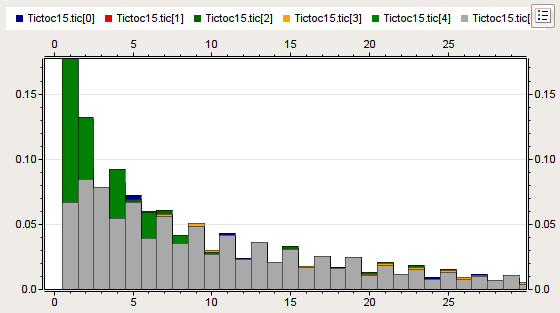
Nếu chúng ta áp dụng một phép toán mean, chúng ta có thể thấy cách các hopCountnút khác nhau hội tụ về mức trung bình:



Biểu đồ tiếp theo hiển thị giá trị trung bình và giá trị tối đa của các hopCount thông báo cho mỗi nút đích, dựa trên dữ liệu vô hướng được ghi lại ở cuối mô phỏng.



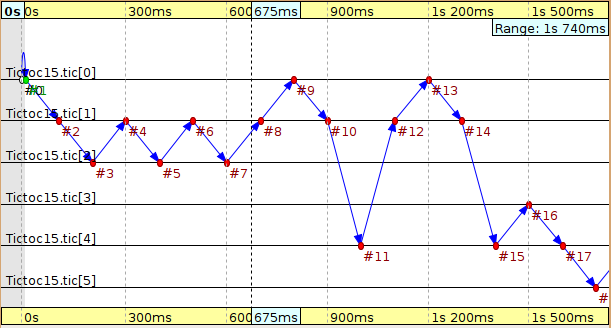
Biểu đồ sau đây cho thấy biểu đồ phân bố của hopCount.



## Biểu đồ tuần tự nhật ký sự kiện kết thúc

Nhân mô phỏng OMNeT ++ có thể ghi lại các thông báo trao đổi trong quá trình mô phỏng vào tệp nhật ký sự kiện . Tệp nhật ký này có thể được phân tích sau bằng công cụ Biểu đồ trình tự Sequence Chart.

Hình sau đây được tạo bằng công cụ Biểu đồ trình tự và cho thấy cách thông điệp được định tuyến giữa các nút khác nhau trong mạng. Trong trường hợp này, biểu đồ rất đơn giản, nhưng khi bạn có một mô hình phức tạp, biểu đồ tuần tự có thể rất có giá trị trong việc gỡ lỗi, khám phá hoặc ghi lại hành vi của mô hình.



## Phần kết luận

Hy vọng bạn đã thấy hướng dẫn này là một giới thiệu hữu ích về OMNeT ++.