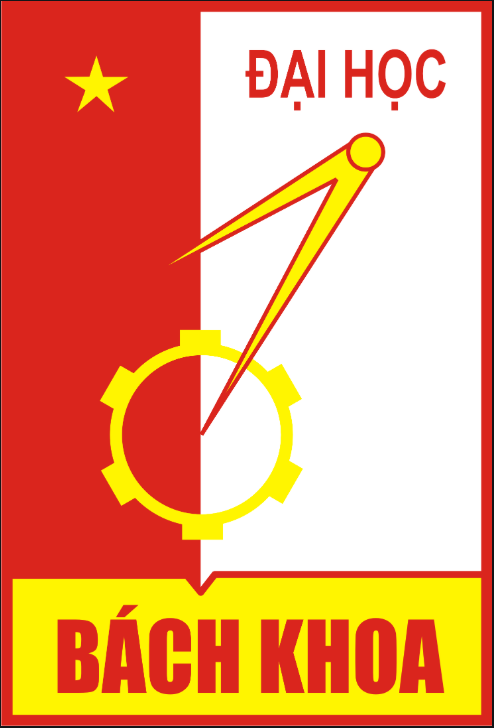


CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM

Độc Lập – Tự Do – Hạnh Phúc

-----🙞🙜🕮🙞🙜----

**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**



**BÁO CÁO MÔN HỌC PROJECT 2**

*Đề tài: Tính thông lượng của các loại hình data-center (torus2d và torus3d) sử dụng trình giả lập OMNeT++*

**Họ và tên: Nguyễn Huy Hoàng**

**MSSV: 20173132**

**Lớp: KHMT-05 K62**

**GVHD:**

*Hà Nội, 6 – 2020*

Lời cam kết

Tôi xin cam đoan đây là phần nghiên cứu và thể hiện đồ án môn học của riêng tôi, không sao chép các đồ án khác, nếu sai tôi xin chịu hoàn toàn trách nhiệm và chịu mọi kỷ luật của khoa và nhà trường đề ra.

Lời cảm ơn

Em xin chân thành cảm ơn thầy giáo hướng dẫn Nguyễn Khanh Văn vì sự giúp đỡ và dìu dắt tận tình của thầy trong suốt quá trình em thực hiện đồ án.

Em xin cảm ơn thầy giáo hướng dẫn Nguyễn Tiến Thành vì những chỉ bảo của thầy đối với những vấn đề kỹ thuật mà em gặp phải.

Em cũng xin gửi lời cảm ơn đến các thầy, cô giáo trong viện CNTT&TT, trường đại học Bách Khoa Hà Nội, những nhà giáo đã truyền dạy cho em những kiến thức quý báu trong suốt những năm học vừa qua.

Cuối cùng, xin cảm ơn gia đình và bè bạn đã tạo điều kiện và luôn động viên tôi hoàn thành đồ án này.

Do thời gian hoàn thành đồ án có hạn cho nên những suy nghĩ cũng như sự thể hiện ý đồ không tránh khỏi có những khiếm khuyết. Em rất mong được sự động viên và đóng góp ý kiến của các thầy cô giáo.

Sinh viên thể hiện

Nguyễn Huy Hoàng

# Giới thiệu đề tài

## 1.1 Đặt vấn đề

Công nghệ thông tin ngày càng phát triển, nhu cầu của con người cũng ngày càng tăng lên. Một trong những vấn đề vướng mắc mà chúng ta đang gặp phải đó là vấn đề về mạng máy tính. Việc kết nối một vài nút mạng với nhau thì không là vấn đề, nhưng khi có hàng trăm, hàng nghìn hay thậm chí hàng triệu nút kết nối với nhau thì khi đó bài toán trở nên rất khó khăn. Sắp tới đây, thế giới sẽ chuyển sang kỷ nguyên của IoT – Internet of Things (Internet của vạn vật). Khi đó, vạn vật đều được kết nối với nhau thông qua mạng internet. Bài toán đặt ra là làm như thế nào để ta có thể kết nối chúng lại với nhau. Chia nhỏ ra thì ta có các bài toán con: Các thiết bị được kết nối theo hình trạng (topology) nào, sử dụng công nghệ kết nối nào, thuật toán định tuyến như thế nào, triển khai tầng vật lý ra sao, có đảm bảo được lưu lượng mạng hay không,... Và quan trọng là phải làm sao để tối ưu chi phí triển khai. Để giải quyết bài toán thì ta cần tính toán trên lý thuyết trước, rồi xây dựng mô hình để kiểm thử. Tuy nhiên, việc xây dựng mô hình thực tế (ví dụ như xây dựng một data-center) để kiểm thử là cực kì khó khăn, tốn kém, đặc biệt là với tình hình kinh tế của nước ta hiện nay thì nhiều trường hợp là bất khả thi. Chính vì thế, ta cần một bước trung gian giữa lý thuyết và thực tế, đó là xây dựng một mô hình giả lập, mô phỏng và tính toán trên đó. Công đoạn này sẽ giúp giảm thiểu những sai sót trong quá trình chuyển từ lý thuyết sang thực tế.

Để xây dựng một mô hình giả lập mạng, trong thực tế có nhiều công cụ có thể sử dụng. Tuy nhiên đồ án này sẽ chỉ sử dụng trình giả lập OMNeT++.

Bài toán cụ thể được lấy ra làm ví dụ ở đây là tính thông lượng (throughput) cho mạng data-center torus2d và torus3d.

## 1.2 Mục tiêu và phạm vi của đề tài

Đề tài này được thực hiện với một mục tiêu chính, đó là sinh viên có thể sử dụng thành thạo một trình giả lập. Ngoài ra còn một số mục tiêu phụ như là sử dụng thành thạo một / một số ngôn ngữ lập trình, biết cách phân tích yêu cầu, tự mày mò nghiên cứu với công nghệ mới để giải quyết yêu cầu được đặt ra; thêm hiểu biết về các khái niệm, giải thuật trong mạng máy tính, ...

Sản phẩm được yêu cầu sau khi hoàn thành đề tài đó là chương trình giả lập hoạt động của mạng data-center torus2d và torus3d. Cùng với đó là các phân tích số liệu về độ hiệu quả của các mạng data-center này.

## 1.3 Định hướng và giải pháp

Để thực hiện đề tài này, sinh viên đã được giáo viên hướng dẫn chia nhỏ công việc ra thành từng lab như sau:

1. Cài đặt OMNeT++ và nghiên cứu, thực hiện 3 mục trong Tictoc tutorial của trang chính thức của OMNeT++.
2. Đi sâu vào Tictoc tutorial
3. Thực hiện các mục còn lại trong Tictoc tutorial và xây dựng một mạng đơn giản có thể gửi gói tin qua lại giữa các nút.
4. Xây dựng file NED cho mạng fattree
5. Xây dựng bảng định tuyến cho Fat Tree và ngẫu nhiên cặp nguồn đích.
6. Mô phỏng quá trình truyền tin trong mạng trung tâm dữ liệu.

Các lab được đặt ra với độ khó tăng dần, lab cuối cùng là sử dụng tổng hợp kiến thức, mã nguồn của các lab trước đó để xây dựng lên chương trình hoàn chỉnh.

## 1.4 Bố cục báo cáo

Báo cáo được chia thành 6 phần chính như sau:

1. Giới thiệu đề tài
2. Khảo sát và phân tích yêu cầu
3. Công nghệ sử dụng
4. Triển khai và phát triển ứng dụng
5. Các giải pháp và đóng góp nổi bật
6. Kết luận và hướng phát triển

# Khảo sát và phân tích yêu cầu

## 2.1 Mô tả OMNeT++

### 2.1.1 OMNeT++ là gì?

OMNeT ++ là một thư viện và khung mô phỏng C ++ dựa trên thành phần, có thể mở rộng, chủ yếu để xây dựng các trình mô phỏng mạng. Mạng có nghĩa rộng hơn bao gồm các mạng truyền thông có dây và không dây, mạng trên chip, mạng xếp hàng, v.v. Chức năng dành riêng cho tên miền như hỗ trợ mạng cảm biến, mạng ad-hoc không dây, giao thức Internet, mô hình hiệu suất, mạng quang tử, v.v., được cung cấp bởi các khung mô hình, được phát triển như các dự án độc lập. OMNeT ++ cung cấp IDE dựa trên Eclipse, môi trường thời gian chạy đồ họa và một loạt các công cụ khác. Có các phần mở rộng cho mô phỏng thời gian thực, mô phỏng mạng, tích hợp cơ sở dữ liệu, tích hợp SystemC và một số chức năng khác. OMNeT ++ được phân phối theo Academic Public License

### 2.1.2 Các thành phần của OMNeT++

Các thành phần chính của OMNeT ++ là:

* Thư viện nhân mô phỏng (C ++)
* Ngôn ngữ mô tả cấu trúc liên kết NED
* IDE mô phỏng dựa trên nền tảng Eclipse
* GUI thời gian chạy mô phỏng tương tác (Qtenv)
* Giao diện dòng lệnh để thực hiện mô phỏng (Cmdenv)
* Các tiện ích (công cụ tạo makefile, v.v.)
* Tài liệu, mô phỏng mẫu, v.v.

### 2.1.3 Nền tảng

Hạt nhân mô phỏng OMNeT ++ là C ++ tiêu chuẩn và chạy cơ bản trên tất cả các nền tảng có trình biên dịch C ++ hiện đại. IDE mô phỏng yêu cầu Windows, Linux hoặc macOS.

### 2.1.4. Cách sử dụng

Dưới đây là tổng quan nhanh về cách ta sẽ sử dụng OMNeT ++:

1. Một mô hình OMNeT ++ được xây dựng từ các thành phần (mô-đun) giao tiếp bằng cách trao đổi thông điệp. Các mô-đun có thể được lồng nhau, nghĩa là, một số mô-đun có thể được nhóm lại với nhau để tạo thành một mô-đun ghép. Khi tạo mô hình, ta cần ánh xạ hệ thống của mình thành một hệ thống phân cấp các mô-đun giao tiếp.
2. Xác định cấu trúc mô hình trong ngôn ngữ NED. Ta có thể chỉnh sửa NED trong trình soạn thảo văn bản hoặc trong trình soạn thảo đồ họa của IDE mô phỏng OMNeT ++ dựa trên Eclipse.
3. Các thành phần hoạt động của mô hình (các mô-đun đơn giản) phải được lập trình trong C ++, sử dụng thư viện mô phỏng và thư viện lớp.
4. Cung cấp một omnetpp.ini phù hợp để giữ các tham số và cấu hình OMNeT ++ cho mô hình của ta. Một tập tin cấu hình có thể mô tả một số mô phỏng chạy với các tham số khác nhau.
5. Xây dựng chương trình mô phỏng và chạy nó. Ta sẽ liên kết mã với hạt nhân mô phỏng OMNeT ++ và một trong những giao diện người dùng OMNeT ++ cung cấp. Có dòng lệnh (lô) và giao diện người dùng đồ họa, tương tác.
6. Kết quả mô phỏng được ghi vào vector đầu ra và tập tin vô hướng đầu ra. Ta có thể sử dụng Công cụ phân tích trong IDE mô phỏng để trực quan hóa chúng. Các tệp kết quả dựa trên văn bản, vì vậy ta cũng có thể xử lý chúng bằng R, Matlab hoặc các công cụ khác.

## 2.2 Data center và topology

### 2.2.1 Data center

Data center (tạm dịch: trung tâm dữ liệu) là một công trình tập trung vào các hoạt động IT và các thiết bị của một tổ chức. Đây đồng thời cũng là nơi lưu trữ, quản lý và phân phối dữ liệu của tổ chức đó.

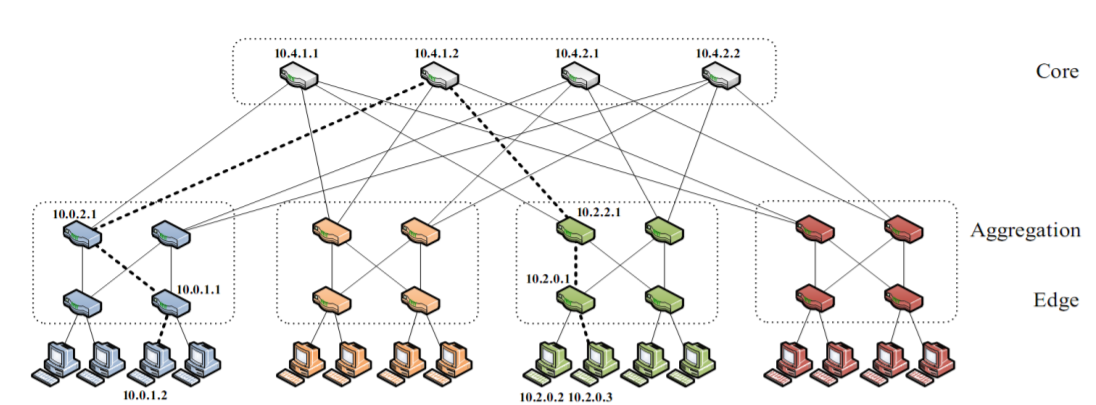
Hiểu đơn giản, data center là khu vực chuyên biệt chứa server hay phòng máy tính. Đây là nơi đặt, vận hành và quản lý server và và các thành phần liên quan (hệ thống truyền thông, hệ thống dữ liệu…).

Các server (và các thành phần liên quan) sẽ được kết nối với nhau thông qua một cấu trúc nhất định, gọi là topology.

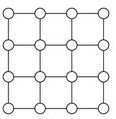
### 2.2.2 Topology

Topology của mạng là cấu trúc hình học không gian mà thực chất là cách bố trí phần tử của mạng cũng như cách nối giữa chúng với nhau. Ví dụ trong mạng LAN thường có 3 dạng cấu trúc là: Mạng dạng hình sao (Star Topology), mạng dạng vòng (Ring Topology) và mạng dạng tuyến (Linear Bus Topology).

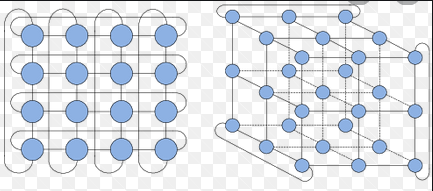
Dưới đây là một số hình ảnh trực quan về các topology hay được sử dụng trong các data center:



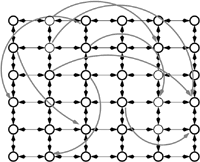
Hình 1: Fattree topology



Hình 2: Grid topology



Hình 3: Torus2d và torus3d



Hình 4: Random shortcut topology

Mỗi topology sẽ có những ưu điểm và nhược điểm khác nhau. Tuỳ thuộc vào các yếu tố mà người ta sẽ chọn một topology tương ứng cho data center, hoặc kết hợp chúng với nhau.

## 2.3 Giải thuật định tuyến

Trong mạng máy tính, định tuyến (routing) là quá trình chọn lựa các đường đi trên một mạng máy tính để gửi dữ liệu qua đó.

Routing chỉ ra hướng, sự di chuyển của các gói (dữ liệu) được đánh địa chỉ từ mạng nguồn của chúng, hướng đến đích cuối thông qua các node trung gian; thiết bị phần cứng chuyên dùng được gọi là router (bộ định tuyến). Tiến trình định tuyến thường chỉ hướng đi dựa vào bảng định tuyến, đó là bảng chứa những lộ trình tốt nhất đến các đích khác nhau trên mạng. Vì vậy việc xây dựng bảng định tuyến, được tổ chức trong bộ nhớ của router, trở nên vô cùng quan trọng cho việc định tuyến hiệu quả.

Việc định tuyến trong đồ án này được thực hiện như sau:

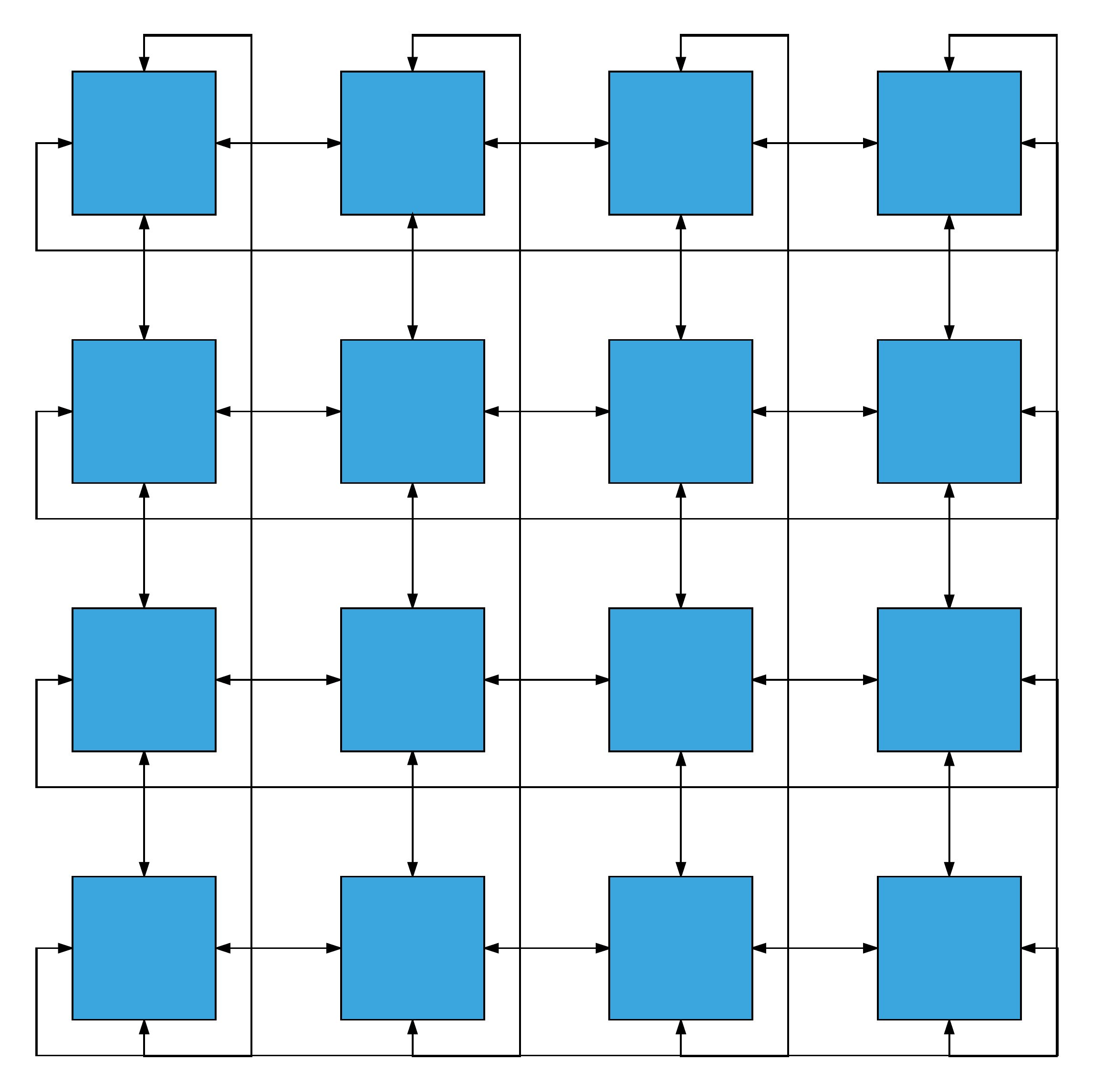
Mỗi switch sẽ lưu trữ bảng định tuyến của riêng mình, được tạo ra khi chương trình bắt đầu khởi động. Các bảng định tuyến được xác định ưu tiên đường đi ngắn nhất, sử dụng giải thuật tìm kiếm theo chiều sâu (BFS). Mỗi khi switch nhận được một gói tin, nó sẽ xác định xem nút đích của gói tin là nút nào, rồi tra bảng định tuyến để xác định cổng ra tương ứng, cuối cùng, switch chuyển tiếp gói tin qua cổng đó.

## 2.4 Phân tích yêu cầu

### 2.4.1 Xây dựng topo

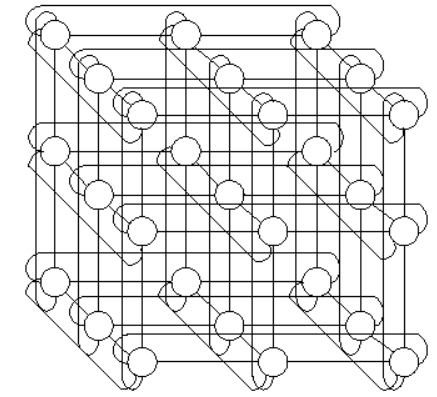
Trong phạm vi của đề tài, ta chỉ thực hiện mô phỏng và tính toán trên hai topology là torus2d và torus3d.

Trong mạng torus 2d (hình 5), các nút được bố trí thành hình vuông có cạnh N × N. Các nút cùng hàng, liền kề nhau sẽ được kết nối với nhau. Tại mỗi hàng, nút đầu tiên được kết nối với nút cuối cùng. Tương tự đối với các cột.



Hình 5: Torus 2D

Tương tự với torus 2d, torus 3d bao gồm các nút được bố trí thành khối lập phương có cạnh N×N×N. Các nút liền kề theo chiều dọc, ngang, sâu được kết nối với nhau, và các nút ở đầu mút có cùng chỉ số hàng, cột, sâu cũng được kết nối với nhau (hình 6).



Hình 6: Torus 3D

Mỗi nút ở đây sẽ tương ứng với một switch. Mỗi switch sẽ kết nối trực tiếp với một host (hay server) không được thể hiện trên hình vẽ. Thực tế, mỗi switch sẽ kết nối với nhiều host, tuy nhiên trong phạm vi bài toán của chúng ta, mục đích cuối cùng là tính thông lượng trong mạng, nên ta sẽ chỉ lấy đại diện một host kết nối với một switch.

### 2.4.2 Xây dựng giải thuật định tuyến

Mỗi switch đều lưu trữ một bảng định tuyến của riêng mình. Các bảng định tuyến này được xây dựng khi chương trình khởi động, cụ thể như sau:

Ta sẽ có một danh sách kết nối giữa các switch và host. Khi bắt đầu chạy giả lập, đọc danh sách này vào bộ nhớ. Tại mỗi switch, duyệt tất cả các đích đến có thể (các host), đối với mỗi đích đến, ta cần xác định cổng ra tương ứng. Danh sách bao gồm các host đích và cổng ra tương ứng sẽ tạo nên bảng định tuyến.

Đối với từng switch, việc xác định cổng ra đối với mỗi gói tin sẽ dựa trên thuật toán tìm kiếm BFS. Vì đặc trưng của giả thiết là trọng số của các tuyến đường đi trong mạng giữa 2 nút đều là bằng nhau, nên kết thúc quá trình tìm kiếm bằng BFS, ta chắc chắn sẽ thu được kết quả là đường đi ngắn nhất. Việc tìm kiếm sẽ dựa trên danh sách kết nối ở trên.

### 2.4.3 Công thức tính throughput

Throughput (thông lượng) là chỉ số đánh giá độ hiệu quả của một mạng. Công thức tính throghput theo từng khoảng thời gian (interval) như sau:

# Công nghệ sử dụng

Trình giả lập được sử dụng là OMNeT++ 5.6.1. Ngôn ngữ để lập trình trên trình giả lập này là C++ và NED. Ngoài ra, đồ án còn sử dụng một số ngôn ngữ khác (Python, Java).

## 3.1 C++

Việc xây dựng chương trình giả lập yêu cầu tác động sâu vào hệ thống, đòi hỏi cao về tốc độ thực thi, nên C++ là ngôn ngữ phù hợp trong trường hợp này.

## 3.2 NED

NED (NEtwork Description) là ngôn ngữ mô tả cấu trúc liên kết của OMNeT ++. Nó có một cú pháp đơn giản nhưng nó rất mạnh khi định nghĩa các cấu trúc liên kết thông thường như chuỗi, vòng, lưới, hypercube, cấu trúc cây, v.v.

**Các tính năng của ngôn ngữ NED**

* Cấu trúc mô đun kiểu phân cấp
  + viết loại mô-đun ghép một lần và tạo một số trường hợp
  + độ sâu làm tổ không giới hạn
* Mô tả cấu trúc liên kết linh hoạt
  + Các tham số có thể định nghĩa: số lượng mô hình con, loại mô hình con, cấu trúc kết nối
  + Mẫu cấu trúc liên kết: một phương tiện tái sử dụng cấu trúc kết nối
* Tham số
  + Có thể chứa các biến ngẫu nhiên (do đó chúng có thể được sử dụng làm nguồn của các số ngẫu nhiên)
  + Có thể được truyền tham chiếu hoặc tham trị
  + Có thể chứa biểu thức của các tham số khác
* Hỗ trợ phân vùng linh hoạt của mô hình để thực hiện song song

Dưới đây là một ví dụ về cách sử dụng của ngôn ngữ NED, cho thấy sự đơn giản và hiệu quả của ngôn ngữ này:

simple Hub

gates:

out: outport[];

endsimple

simple Station //...

module Star

submodules:

hub: Hub

gatesizes: outport[4];

station: Station[4];

connections:

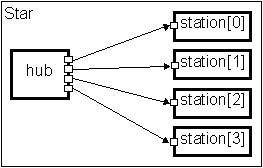
for i=0..3 do

hub.outport[i] --> station[i].in;

endfor

endmodule

Kết quả:



Hình 7: Tạo nhiều kết nối, sử dụng vòng lặp for

## 3.3 Java

Ngôn ngữ java được sử dụng để tạo file NED trong đồ án này. Chương trình chạy bằng java này nhận đầu vào là kích thước của mạng torus, đầu ra sẽ là file NED mô tả topology torus2d hoặc torus3d tương ứng.

## 3.4 Python

Ngôn ngữ python được sử dụng để vẽ biểu đồ phân tích kết quả. Thư viện vẽ biểu đồ được sử dụng ở đây là Matplotlib.

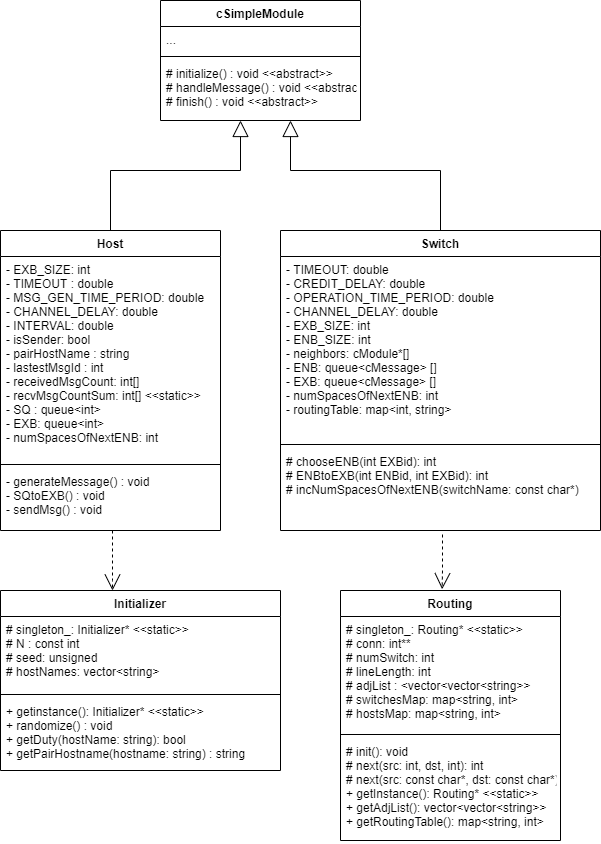
# Triển khai và phát triển ứng dụng

Ứng dụng được triển khai làm 3 phần.

* Thứ nhất là việc tạo ra file NED mô tả topology của mạng torus2d hoặc 3d. Ngoài ra, chương trình còn tạo ra file ConnectionsList.txt lưu danh sách kết nối của các nút. Chương trình này được viết bằng java.
* Thứ hai là chương trình chính, viết bằng C++ và chạy trên nền tảng của OMNeT++. Chương trình giả lập quá trình truyền tin trong mạng torus2d hoặc 3d. Kết thúc chương trình này, kết quả sẽ được lưu vào tệp Result.txt (chứa tổng số lượng các gói tin nhận được theo từng interval của tất cả các nút nhận)
* Thứ ba là chương trình tính toán throughput và vẽ ra biểu đồ, được viết bằng python.

## 4.1 Chương trình OMNeT++ chính

### 4.1.1 Biểu đồ lớp



Hình 8: Biểu đồ lớp của chương trình chính

### 4.1.2 Mô tả các lớp

|  |  |
| --- | --- |
| **Initializer** | |
| Lớp có tác dụng khởi tạo các thiết lập ban đầu cho chương trình.  Cụ thể ở đây mới chỉ có một tác chức năng là ghép cặp ngẫu nhiên các nút nguồn – đích | |
| **Thuộc tính** | **Chức năng** |
| singleton\_ | Biến lưu trữ thể hiện singleton của lớp |
| N | Hằng số kích thước của mạng torus |
| seed | Biến lưu seed để tạo giá trị ngẫu nhiên, được gán giá trị trong hàm khởi dựng |
| hostNames | Mảng các tên của các host |
| **Phương thức** | **Chức năng** |
| getInstance() | Lấy ra thể hiện singleton của lớp |
| randomize() | Xáo trộn ngẫu nhiên các cặp nguồn đích |
| getDuty(hostName: string): bool | Lấy ra nhiệm vụ của một nút host dựa theo tên của nó, true nếu như nó là nút gửi, false nếu như nó là nút nhận |
| getPairHostname(hostname: string) : string | Lấy ra tên host được ghép cặp với host hiện tại |

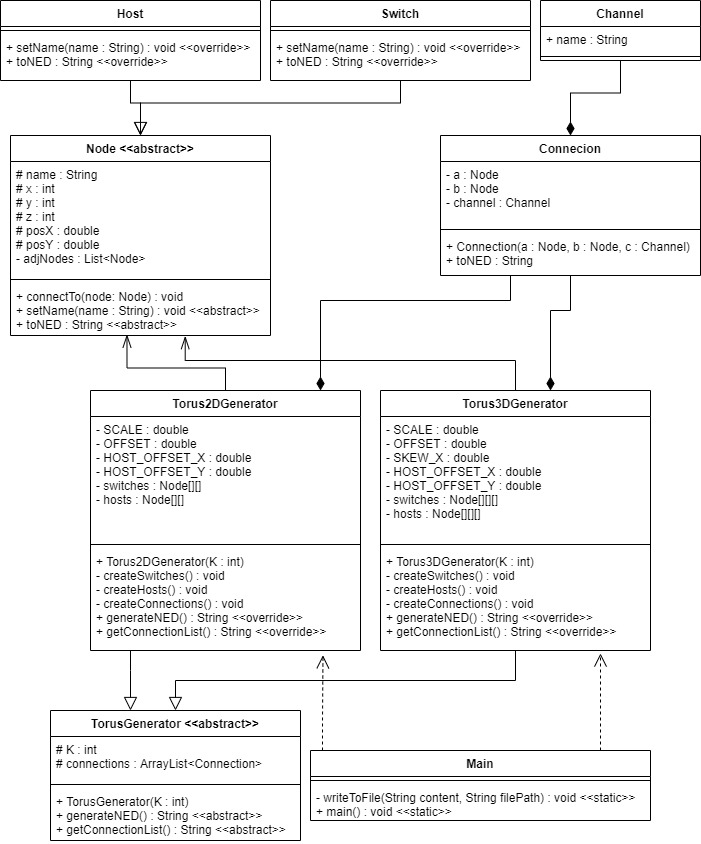
|  |  |
| --- | --- |
| **Routing** | |
| Lớp phục vụ cho chức năng định tuyến | |
| **Thuộc tính** | **Chức năng** |
| singleton\_ | Biến lưu trữ thể hiện singleton của lớp |
| conn | Mảng hai chiều lưu ma trận kết nối giữa các nút |
| numSwitch | Số lượng các switch |
| lineLength | Độ dài của một hàng trong file lưu danh sách kết nối (ConnectionsList.txt) |
| adjList | Vector 2 chiều lưu danh sách kết nối |
| switchesMap | map từ tên switch sang số hiệu của switch  ví dụ: “sw0\_0\_0” 🡪 0 |
| hostsMap | map từ tên hosts sang số hiệu của host  ví dụ: “h0\_0\_1” 🡪 1001 |
| **Phương thức** | **Chức năng** |
| init() : void | Khởi tạo các giá trị cần thiết cho việc định tuyến. Bao gồm việc đọc file ConnectionLists.txt, ghi ra vector adjList, khởi tạo hostsMap, switchesMap, conn, lineLength. |
| next(src: int, dst: int) : int | Hàm tìm kiếm theo BFS. src là id của nút nguồn, dst là id của nút đích. Giá trị trả về là số hiệu của cổng ra cần tìm. |
| next(src: const char\*, dst: const char\*): int | Hàm tìm cổng ra theo tên của nút nguồn và nút đích. Hàm này chỉ đơn giản là chuyển từ tên nút sang số hiệu rồi gọi hàm next(int, int) ở trên. |
| getInstance() | Lấy ra thể hiện singleton của lớp |
| getAdjList() : vector<vector<string>> | getter cho adjList |
| getRoutingTable(srcName: const char\*) : map<string, int> | Lấy ra bảng định tuyến theo tên của nút nguồn |

|  |  |
| --- | --- |
| **Host** | |
| Lớp mô tả hoạt động của một host | |
| **Thuộc tính** | **Chức năng** |
| EXB\_SIZE: int | Hằng số chỉ kích thước của EXB của host |
| TIMEOUT : double | Hằng số chỉ thời gian chạy giả lập |
| MSG\_GEN\_TIME\_PERIOD : double | Hằng số chỉ chu kỳ sinh gói tin |
| CHANNEL\_DELAY : double | Hằng số chỉ trễ đường truyền (cụ thể ở đây là trễ truyền tải, bằng kích thước gói tin chia cho băng thông đường truyền) |
| INTERVAL : double | Hằng số giá trị khoảng thời gian Interval |
| isSender : bool | Biến xác định xem host là nút gửi hay nút nhận. true nếu là nút gửi, false nếu là nút nhận. |
| pairHostName : string | Tên của nút được ghép cặp với nút hiện tại |
| lastestMsgId : int | Id của gói tin gần nhất vừa được tạo. Bởi id của các gói tin được tạo ra là các giá trị tăng dần, nên cần thuộc tính này. |
| receivedMsgCount : int[] | Mảng đếm số lượng gói tin nhận được theo từng interval |
| recvMsgCountSum : int[] <<static>> | Mảng đếm tổng số lượng gói tin nhận được theo từng interval của tất cả các nút nhận. Mảng này phục vụ cho việc ghi ra file kết quả cuối cùng |
| SQ : queue<int> | Source queue. Mỗi khi gói tin được tạo ra sẽ được đưa vào hàng đợi này để chuẩn bị cho việc gửi. |
| EXB: queue<int> | Exit buffer. Mỗi khi gói tin chuẩn bị được gửi đi sẽ được đưa vào hàng đợi này. |
| numSpacesOfNextENB : int | Số lượng chỗ trống còn lại của Entrance buffer của switch liền kề. |
| **Phương thức** | **Chức năng** |
| generateMessage() : void | Tạo các gói tin có id tăng dần |
| SQtoEXB() : void | Chuyển gói tin từ source queue sang exit buffer |
| sendMsg() : void | Gửi gói tin đi. |

|  |  |
| --- | --- |
| **Switch** | |
| Lớp mô tả hoạt động của một switch | |
| **Thuộc tính** | **Chức năng** |
| TIMEOUT : double | Hằng số thời gian giả lập. |
| CREDIT\_DELAY : double | Hằng số thời gian trễ cho việc báo lại từ nút hiện tại tới nút trước đó để nút trước đó biết rằng nút hiện tại đã có thêm khoảng trống ở ENB |
| OPERATION\_TIME\_PERIOD : double | Hằng số chỉ chu kỳ hoạt động của switch |
| CHANNEL\_DELAY : double | Hằng số chỉ trễ kênh truyền (tương tự trong lớp host) |
| EXB\_SIZE : int | Hằng số chỉ kích thước của exit buffer |
| ENB\_SIZE : int | Hằng số chỉ kích thước của entrance buffer |
| neighbors : cModule\*[] | Mảng lưu các nút kề, phục vụ cho việc báo lại |
| ENB : queue<cMessage> [] | Mảng các entrance buffer |
| EXB : queue<cMessage> [] | Mảng các exit buffer |
| numSpacesOfNextENB : int [] | Mảng lưu số lượng chỗ trống của ENB của các nút kế tiếp |
| routingTable: map<int, string> | Bảng định tuyến của switch hiện tại |
| **Phương thức** | **Chức năng** |
| chooseENB(int EXBid) : int | Chọn ra ENB có gói tin được ưu tiên để gửi sang một EXB nào đó. |
| ENBtoEXB(int ENBid, int EXBid) : void | Chuyển gói tin từ ENB sang EXB |
| incNumSpacesOfNextENB(switchName: const char\*) | Tăng giá trị của biến lưu số chỗ trống của ENB của nút kế |

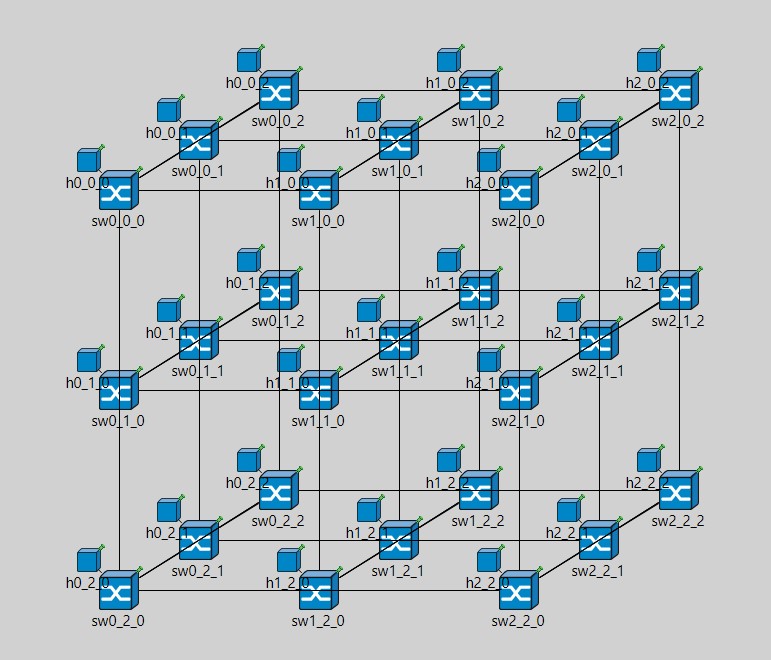
|  |  |
| --- | --- |
| **cSimpleModule** | |
| Lớp này được định nghĩa sẵn trong gói thư viện của OMNeT++  Có tác dụng là mô tả chức năng của một module. | |
| **Thuộc tính** | **Chức năng** |
| ... | Các thuộc tính không đề cập ở đây |
| **Phương thức** | **Chức năng** |
| Nhiều phương thức không đề cập ở đây, chỉ có 3 phương thức chính dưới đây là quan trọng.  Lập trình viên ghi đè 3 phương thức này ở lớp con, để thực hiện các chức năng mong muốn. | |
| initialize() : void <<abstract>> | Các khởi tạo ban đầu, hàm này được gọi khi trình giả lập được khởi động và trước khi sự kiện đầu tiên xảy ra. |
| handleMessage() : void <<abstract>> | Xử lí khi nhận được một gói tin. Hàm này được gọi mỗi khi module nhận được một gói tin |
| finish() : void <<abstract>> | Xử lí sau khi kết thúc giả lập, ví dụ như việc ghi ra file các giá trị phục vụ cho việc phân tích kết quả. Hàm này được gọi khi kết thúc quá trình giả lập. |

## 4.2 Chương trình tạo file NED



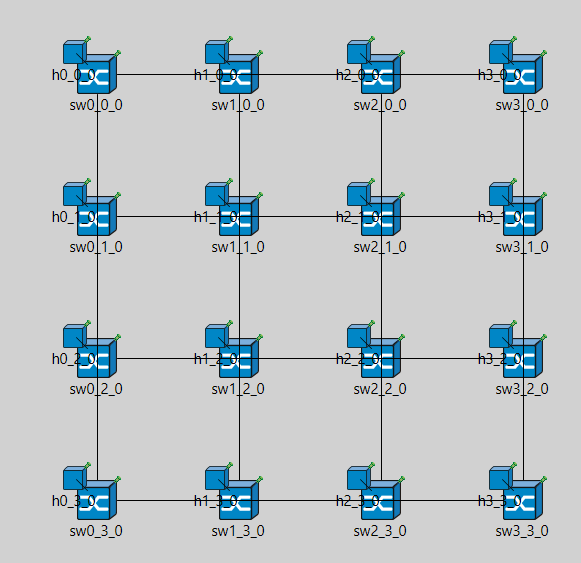
Hình 9: Biểu đồ lớp của chương trình tạo file NED

Chương trình này (viết bằng java) sẽ xuất ra mã nguồn file NED dựa theo kích thước của mạng torus và loại mạng 2D hay 3D. Hình ảnh dưới đây minh hoạ cho đầu ra của chương trình này, (xem ở chế độ design trong OMNeT++):



Hình 10: Torus3D 3x3

Một số kết nối ví dụ như của sw0\_0\_0 với sw0\_0\_2 chưa được thể hiện trực quan cho lắm, vì nó nằm đè lên các kết nối khác.



Hình 11: Torus2D 4x4

Bên cạnh đó, chương trình còn sinh thêm file ConnecitionsList.txt lưu danh sách các kết nối của mạng. Khuôn dạng của file ConnectionsList.txt như sau, (lấy ví dụ của mạng torus2d 4x4)

sw0\_0\_0 sw0\_1\_0 sw1\_0\_0 sw0\_3\_0 sw3\_0\_0 h0\_0\_0

sw0\_0\_0 sw0\_1\_0 sw1\_0\_0 sw0\_3\_0 sw3\_0\_0 h0\_0\_0

sw0\_0\_0 sw0\_1\_0 sw1\_0\_0 sw0\_3\_0 sw3\_0\_0 h0\_0\_0

sw0\_0\_0 sw0\_1\_0 sw1\_0\_0 sw0\_3\_0 sw3\_0\_0 h0\_0\_0  
...  
sw3\_3\_0 sw3\_2\_0 sw2\_3\_0 sw3\_0\_0 sw0\_3\_0 h3\_3\_0

sw3\_3\_0 sw3\_2\_0 sw2\_3\_0 sw3\_0\_0 sw0\_3\_0 h3\_3\_0

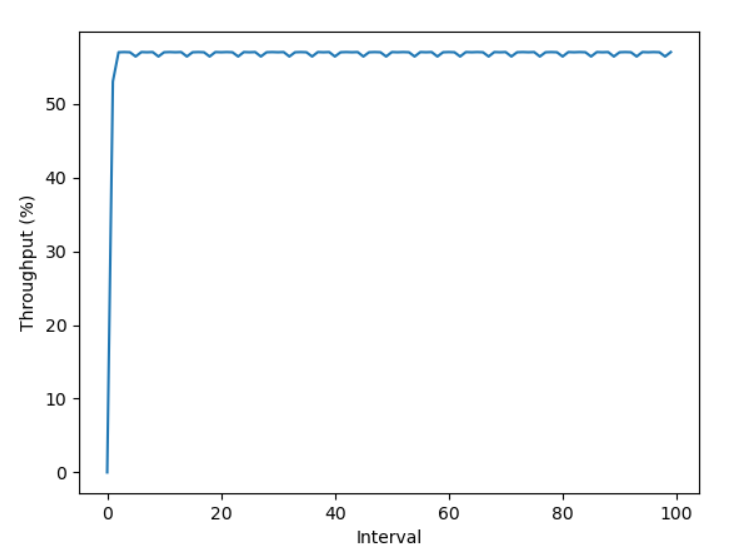
Các dòng ghi các tên switch và host. Đối với mỗi dòng, phần tử đầu tiên ghi tên switch, các phần tử tiếp theo ghi tên các nút có kết nối với nó theo thứ tự tăng dần của cổng ra.

Tệp tin này phục vụ cho việc định tuyến ở chương trình chính.

## 4.3 Vẽ biểu đồ

Chương trình ngắn này sẽ tính toán throughput theo công thức dưới đây và vẽ ra biểu đồ.

**import** matplotlib.pyplot **as** plt  
  
*# packet size = 100Kb*packetSize = 100e3  
*# bandwidth = 1Gbps*bandwidth = 1e9  
*# number of receiving host (torus2d 4x4)*numRecvHost = 8  
*# interval time = 10ms*intervalTime = 0.01  
*# interval count*intervalCount = int(1 / intervalTime)  
  
  
coefficient = packetSize / (bandwidth \* numRecvHost \* intervalTime) \* 100  
  
sumRecvMsg = [0 **for** i **in** range(0, intervalCount)]  
  
*# file lưu số gói tin nhận được của tất cả các nút theo từng đơn vị interval time*path0 = **"C:\\Users\\dranh\\Desktop\\Omnetpp-project\\Lab 06\\Source\\Lab06\_5\\Result"**triedTimes = 10  
  
**for** i **in** range(0, triedTimes):  
 path = path0 + str(i) + **".txt"** f = open(path)  
 t = f.read().strip().split(**' '**)  
 print(path)  
 print(t)  
 **for** j **in** range(0, intervalCount):  
 sumRecvMsg[j] += int(t[j])  
  
throughput = []  
  
**for** i **in** range(0, len(sumRecvMsg)):  
 throughput.append(int(sumRecvMsg[i]) \* coefficient / triedTimes)  
  
plt.plot(throughput)  
plt.xlabel(**'Interval'**)  
plt.ylabel(**'Throughput (%)'**)  
plt.show()



Hình 12: Kết quả trung bình của 10 lần chạy

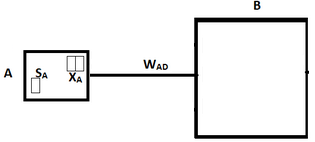
## 4.4 Kết quả

# Các giải pháp và đóng góp nổi bật

## 5.1 Nghiệp vụ bài toán

### 5.1.1 Cài đặt source queue và exit buffer

Trong trường hợp tổng quát, một gói tin khi được sinh ra ở nút nguồn chưa chắc sẽ được gửi đi ngay mà còn căn cứ vào tình trạng của các thiết bị lân cận (chẳng hạn như dây dẫn có đang bận truyền gói tin khác không).



Hình 13: Source queue và exit buffer của nút nguồn

Source queue là nơi lưu trữ các gói tin được sinh ra ở một nút còn EXB là nơi lưu trữ các gói tin chuẩn bị rời khỏi nút nguồn. SQ có kích thước vô hạn còn EXB là một mảng có kích thước giới hạn và kích thước này có thể là một tham số truyền vào cho file NED.

Các gói tin được sinh ra theo một chu kỳ nhất định, chu kỳ này cũng là tham số truyền vào file NED. Mỗi gói tin được truyền đi trên đường truyền sẽ mất một khoảng thời gian nhất định, giá trị khoảng thời gian này cũng là tham số được truyền vào file NED. Mỗi gói tin đều có một id của mình và id này là giá trị tự động tăng.

Mỗi nút đích đều có một mảng một chiều với phần tử thứ *i* lưu trữ số gói tin đến được nút đích tại interval time thứ *i*, giá trị interval time này cũng một tham số được truyền vào file NED

Một khi gói tin được sinh ra, nếu EXB còn chỗ trống thì gói tin được lưu trữ trong vị trí còn trống của EXB. Còn không thì gói tin vẫn nằm trong SQ.

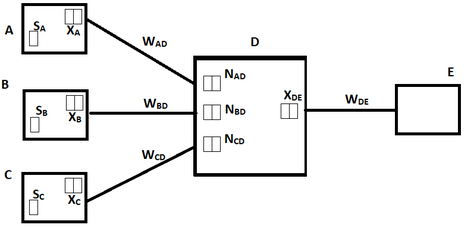
Một khi EXB có chỗ trống mới thì một gói tin của SQ sẽ được chuyển lên cho EXB ngay.

Một khi vị trí đầu tiên của EXB có gói tin xuất hiện thì EXB sẽ kiểm tra xem đường truyền có bận không để truyền gói tin lên đường truyền.

Một khi đường truyền hết bận, nếu trong EXB có gói tin thì gói tin đó sẽ được gửi lên đường truyền.

Khi một gói tin đến được với nút đích, nút đích sẽ xác định thời điểm gói tin đến nơi, nếu thời điểm thuộc về interval time nào thì sẽ tăng giá trị của mảng một chiều tại phần tử interval time đó.

### 5.1.2 Cài đặt Entrance buffer và Exit buffer ở các nút switch.



Hình 14: ENB và EXB ở các nút trung gian

Gói tin cần phải đi vào một entrance buffer (ENB), sau đó được chuyển qua EXB rồi mới rời khỏi nút trung gian switch.

Kích thước của ENB và EXB của switch cũng được lấy từ tham số trong file NED. Để đơn giản, mọi nút switch đều có kích thước này giống nhau và cũng bằng kích thước của EXB trong nút nguồn. Nếu nút switch có kết nối đến với k nút khác nhau thì trong switch đó có tới k ENB và k EXB để phục vụ việc nhận/gửi gói tin qua k cổng khác nhau. Trong hình 2, để đơn giản thì hình vẽ này đã được lược bỏ bớt đi (không vẽ vào) các ENB và EXB còn lại.

Cơ chế hoạt động của mạng trong hình 2 như sau:

Điều kiện một gói tin được phép rời khỏi EXB của nút nguồn sẽ bổ sung thêm ý sau: nếu ENB của nút switch D còn chỗ trống thì gói tin mới được đưa lên đường truyền.

Một khi có một gói tin được gửi lên đường truyền, nút nguồn sẽ lưu trữ thông tin về số lượng các chỗ trống còn lại trong ENB của switch, (giả sử đây là biến mang tên counter). Rõ ràng khi truyền đi một gói tin thì biến counter này sẽ giảm đi một.

Một khi gói tin đến được vị trí đầu tiên của ENB trong switch, switch sẽ xem gói tin này cần đi ra cổng EXB nào của nó; có bao nhiêu gói tin cũng muốn đi ra cổng đó (nếu nhiều hơn 1 thì chọn gói tin nào có id bé nhất); cổng EXB kế tiếp đã đầy hay chưa?

Một khi các điều kiện trên thỏa mãn thì switch sẽ tạo ra một sự kiện với thời điểm dự kiến thực thi sự kiện là bằng thời điểm hiện tại cộng với chu kỳ hoạt động của switch (chu kỳ này cũng là tham số từ bàn phím). Sự kiện này sẽ truyền gói tin đó từ ENB sang EXB.

Một khi gói tin được truyền từ ENB sang EXB, một sự kiện sẽ xảy ra ở nút trước đó (A hoặc B hoặc C tùy vào gói tin xuất phát từ nút nào), biến counter sẽ được tăng thêm một. Thời điểm dự kiến xảy ra sự kiện này là một giá trị CREDIT\_DELAY được khai báo là tham số trong file NED.

Một khi gói tin đến được vị trí đầu tiên của EXB của switch, switch sẽ xem đường truyền có trống không? Nếu có thì gói tin của EXB sẽ được truyền sau một khoảng thời gian là một chu kỳ hoạt động của switch.

Một khi EXB chuyển từ trạng thái đang đầy sang trạng thái có một chỗ trống mới thì switch sẽ kiểm tra trong danh sách các gói tin ở phần đầu của các ENB, xem có gói tin nào muốn đến EXB hay không? Nếu có thì chọn ra gói tin có id nhỏ nhất mà kích hoạt sự kiện chuyển gói tin đó sang EXB. Tất nhiên, thời điểm dự kiến xảy ra sự kiện là thời điểm hiện tại cộng với một chu kỳ hoạt động của switch.

Chú ý rằng những sự kiện chuyển từ ENB sang EXB vẫn có thể không được thực hiện mặc dù đã đến đúng thời điểm dự kiến xảy ra của nó. Nguyên nhân hàng đầu là vì một lý do nào đó khiến EXB bị đầy, khi ấy sự kiện đó không làm thay đổi vị trí của gói tin (vẫn lưu trữ tại ENB).

### 5.1.3 Sinh tự động mạng trung tâm dữ liệu

Ở trung tâm dữ liệu, các nút nguồn đều có các source queue, exit buffer, counter. Các nút trung gian đều có các ENB và EXB ở bên trong. Bản thân nút trung gian nếu kết nối đến với m switch khác thì sẽ có tổng cộng m biến counter dành riêng cho từng switch.

### 5.1.4 Viết mã nguồn của cho chức năng định tuyến:

Một khi gói tin đi đến một nút trung gian, switch đó sẽ kiểm tra xem gói tin này cần đi đến nút đích D nào. Sau đó kiểm tra trong bảng định tuyến của mình để biết xem gói tin này cần đi ra cổng nào.

### 5.1.5 Xây dựng chương trình giả lập quá trình truyền tin trong mạng trung tâm dữ liệu

Sau khi sinh mã nguồn file NED, cần cài đặt quá trình truyền tin như sau:

+ chọn ngẫu nhiên một nửa các nút nguồn trong số các nút nguồn của mạng.

+ Ghép cặp ngẫu nhiên của một nửa này với nửa kia. Một nút trong cặp này sẽ đảm nhận vai trò chỉ sinh ra gói tin rồi gửi, nút còn lại sẽ đảm nhận vai trò chỉ nhận gói tin rồi tính toán trong mảng lưu trữ số gói tin đã nhận của mình.

+ Các nút nguồn có vai trò sinh và gửi gói tin sẽ thực hiện việc sinh ra và gửi đi. Các nút trung gian sẽ nhận và truyền gói tin. Các nút đích sẽ nhận và tính toán.

+ kết thúc thời gian giả lập, ta sẽ có các mảng một chiều lưu trữ số các gói tin nhận được tại mỗi nút đích theo các interval time khác nhau. Ghép các mảng một chiều này để có được một ma trận hai chiều. Tiếp đó tính toán số lượng byte đã nhận trong một đơn vị thời gian, bằng cách nhân ma trận hai chiều kia với dung lượng của một gói tin rồi chia cho giá trị thời gian interval time.

+ Tính tổng tất cả các byte đã được bởi tất cả các nút đích trong từng interval time khác nhau rồi chia cho số lượng các nút đích. Cuối cùng chia cho băng thông của đường dây. Ở đây sẽ được một mảng một chiều trong đó số phần tử là số lượng các interval time, giá trị của mỗi phần tử là throughput của mạng trong một interval time. Tiếp đó vẽ đồ thị biểu diễn kết quả dưới dạng đồ thị.

Các thông số như sau:

+ chu kỳ sinh gói tin: 0.1ms

+ Chu kỳ hoạt động của switch: 3334ns

+ Dung lượng của gói tin: 100Kb, coi như xấp sỉ 100.000 bit.

+ Thời gian giả lập là 1s, credit\_delay là 1ns.

+ Giá trị interval time là 10ms

+ Các EXB, ENB có kích thước 5.

+ Băng thông của đường truyền là 1Gbps, coi như xấp sỉ 1 tỉ bit/s

## 5.2 Các khó khăn

### 5.2.1 Xử lí sự kiện trên các nút mạng

Để có thể làm cho các switch và host hoạt động đúng theo mô tả của yêu cầu, thì việc xử lí sự kiện trên chúng là một việc khó khăn. Các switch và host hoạt động theo chu kỳ rất nhỏ (cỡ nano giây), vì thế trong thời gian giả lập là 1s, có đến cả triệu sự kiện (gửi, nhận gói tin,...) diễn ra. Cơ chế xử lí sự kiện của OMNeT++ mặc định sẽ theo dõi và quan sát từng sự kiện một, ghi lại các thông tin cho từng sự kiện một. Vì thế, nếu như có quá nhiều sự kiện diễn ra, thời gian chạy của chương trình sẽ rất lâu, kéo dài thậm chí hàng tiếng đồng hồ.

Có 2 cách để xử lí sự kiện trong OMNeT++:

* Cách thứ nhất là sử dụng hàm handleMessage(). Cách này là cách thường được sử dụng.
* Cách thứ hai là sử dụng hàm activity() cùng vòng lặp while true.

Em đã nghiên cứu hàng giờ để tìm hiểu cách sử dụng thứ hai với mong muốn rằng chương trình sẽ thực hiện nhanh hơn bằng cách bỏ qua không ghi lại những sự kiện không cần thiết (vd như hoạt động của switch, chủ yếu là sự kiện chuyển tiếp chu kỳ mà không có gì đặc biệt). Tuy nhiên cách này cũng không khả thi.

Giải pháp cuối cùng đó là phải tăng giá trị chu kỳ hoạt động của switch lên, để số lượng sự kiện xảy ra giảm đi. Như thế thời gian chạy của chương trình mới nhanh được.

### 5.2.2 Triển khai cơ chế báo lại cho nút trước

Theo yêu cầu của bài toán, tại mỗi nút trung gian (tạm gọi là A), mỗi khi ENB có chỗ trống thì sẽ có một sự kiện xảy ra ở nút kế trước (tạm gọi là B), đó là tăng biến counter lên 1 đơn vị. Sự kiện này mất một khoảng thời gian là CREDIT\_DELAY.

Để nút B biết và tăng biến counter lên 1 đơn vị, thì chỉ có cách là nút A phải gửi một thông điệp (hoặc gọi hàm hay sử dụng cơ chế gì đó, tức là phải xuất phát từ A) tới nút B. B không thể tự căn chỉnh thời gian để thực hiện công việc này, vì thời gian đó là không cố định và việc tính toán thì gần như bất khả thi). Có 3 hướng giải quyết như sau:

* Hướng thứ nhất là sử dụng trực tiếp kênh truyền đã được thiết lập giữa các nút để gửi một gói tin báo lại cho nút B. Tuy nhiên cách này không khả thi, vì giá trị CREDIT\_DELAY rất nhỏ so với trễ truyền tin, mà gửi trên kênh truyền thì lại mất một khoảng thời gian lớn.
* Hướng thứ hai cũng là gửi đi một thông điệp từ nút A tới nút B, nhưng không dựa trên kênh truyền đã được thiết lập, mà sử dụng hàm sendDirect(), tức là không cần dùng đến kênh truyền. Tuy nhiên cách này lại phức tạp, bởi vì muốn sử dụng hàm này thì ta cần có các cổng được tạo sẵn với annotation @directIn, và các cổng này không được kết nối với bất kể kênh truyền nào. Như thế rất bất tiện cho việc định tuyến sau này, vì số hiệu các cổng có liên quan đến việc định tuyến. Hơn nữa, việc tạo file NED cũng phức tạp hơn.
* Hướng cuối cùng (được sử dụng trong chương trình) là từ module A, gọi trực tiếp hàm tăng giá trị biến counter của module B. Với cách này, khi các nút trung gian được khởi tạo, ta cần xác định các nút kề với nó và lưu các con trỏ trỏ tới chúng vào một mảng. Sau đó, chỉ cần gọi hàm incNumSpacesOfNextENB() của nút B từ nút A sau một khoảng thời gian CREDIT\_DELAY là được.

### 5.2.3

# Kết luận và hướng phát triển

Tài liệu tham khảo

* Tài liệu gốc trên trang chính thức của OMNeT++   
  <https://omnetpp.org/>
* NED language overview  
  <https://www.ewh.ieee.org/soc/es/Nov1999/18/ned.htm>