BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI BÁO CÁO THỰC TẬP CƠ SỞ**

**CHỦ ĐỀ**

**ĐỊNH TUYẾN TRÊN ROUTER MỀM DÙNG MININET**

**Giảng viên hướng dẫn: ThS. Cấn Thị Phượng**

**Sinh viên thực hiện: Nguyễn Hữu Tân**

**Mã số sinh viên: 61134319**

Khánh Hòa, Tháng 12/2021

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI BÁO CÁO THỰC TẬP CƠ SỞ**

**CHỦ ĐỀ**

**ĐỊNH TUYẾN TRÊN ROUTER MỀM DÙNG MININET**

Giảng viên hướng dẫn: ThS. Cấn Thị Phượng

Sinh viên thực hiện: Nguyễn Hữu Tân

Mã số sinh viên: 61134319

Khánh Hòa, Tháng 12/2021

LỜI CAM ĐOAN

Tôi xin được cam đoan : Đề tài báo cáo thực tập cơ sở “Định tuyến trên Router mềm dùng Mininet” là kết quả dựa trên sự cô gắng, nỗ lực của bản thân với sự hướng dẫn nhiệt tình của giảng viên Cấn Thị Phượng. Các kết quả và nghiên cứu trong đề tài là trung thực và hoàn toàn không sao chép hoặc sử dụng kết quả của đề tài nghiên cứu nào tương tự.

Những phần sử dụng tài liệu tham khảo trong đồ án đã trích dẫn đầy đủ.

Nếu phát hiện có sự sao chép kết quả nghiên cứu của đề tài khác, tôi xin chịu hoàn toàn trách nhiệm và kỷ luật của giảng viên đưa ra.

*Khánh Hòa, ngày 30 tháng 12 năm 2021*

**Sinh viên thực hiện**

*Nguyễn Hữu Tân*

MỤC LỤC

[CHƯƠNG 1: TỔNG QUAN VỀ ĐỊNH TUYẾN 7](#_Toc93006042)

[1.1. Giới thiệu về định tuyến 7](#_Toc93006043)

[1.2. Phân loại định tuyến 7](#_Toc93006044)

[1.2.1. Định tuyến tĩnh 7](#_Toc93006045)

[1.2.2. Định tuyến động: 8](#_Toc93006046)

[1.2.2.1. Mục đích của định tuyến động 9](#_Toc93006047)

[1.2.2.2. Phân loại định tuyến động 9](#_Toc93006048)

[1.2.2.3. Nhưng thông số quan trọng 9](#_Toc93006049)

[CHƯƠNG 2: TỔNG QUAN VỀ MININET 11](#_Toc93006050)

[2.1. Giới thiệu về Mininet 11](#_Toc93006051)

[2.2. Một số chức năng của Mininet 11](#_Toc93006052)

[2.3. Hạn chế của Mininet 12](#_Toc93006053)

[CHƯƠNG 3: TÌM HIỂU CONTROLLER POX 13](#_Toc93006054)

[3.1. Controller POX là gì? 13](#_Toc93006055)

[3.2. Mô phỏng: 13](#_Toc93006056)

[CHƯƠNG IV: LẬP TRÌNH MÔ PHỎNG ĐỊNH TUYẾN TRÊN MININET 17](#_Toc93006057)

[4.1. Các câu lệnh cơ bản trong Mininet 17](#_Toc93006058)

[4.2 Mô phỏng định tuyến trên Mininet: 17](#_Toc93006059)

MỤC LỤC HÌNH ẢN**H**

[Hình 1.1 Định tuyến 8](#_Toc93006187)

[Hình 1.2.1 Định tuyến tĩnh 9](#_Toc93006188)

[Hình 1.2.2 Định tuyến động 10](#_Toc93006189)

[Hình 3.1 14](file:///D:\ThucTapCoSo\Báo%20cáo.docx#_Toc93006190)

[Hình 3.2: Mô phỏng POX 14](file:///D:\ThucTapCoSo\Báo%20cáo.docx#_Toc93006191)

LỜI CẢM ƠN

Để hoàn thành đề tài báo cáo thực tập cơ sở này, trước hết em xin gửi đến quý thầy, cô Khoa Công nghệ Thông tin – Trường Đại Học Nha Trang lời cảm ơn chân thành.

Em xin gửi đến cô Cấn Thị Phượng, người đã tận tình hướng dẫn, giúp đỡ em hoàn thành đề tài báo cáo này lời cảm ơn sâu sắc nhất.

Trong quá trình nghiên cứu thực hiện đề tài, cũng như trong quá trình làm bài báo cáo thực tập cơ sở, khó tránh khỏi sai sót, rất mong các thầy, cô bỏ qua. Đồng thời do kiến thức cũng như kinh nghiệm thực tiễn của bản thân còn hạn chế, bài báo cáo này khó tránh khỏi những thiếu sót, em rất mong nhận được ý kiến đóng góp từ quý thầy, cô để em học hỏi thêm được nhiều kinh nghiệm, cũng như kỹ năng cần thiết.

Em xin chân thành cảm ơn!

LỜI MỞ ĐẦU

Một router khá khác với một switch hoặc hub vì chức năng chính của nó là định tuyến các gói dữ liệu tới các mạng khác, thay vì chỉ các máy tính cục bộ. Router khá phổ biến để tìm trong nhà và doanh nghiệp vì nó cho phép mạng của bạn giao tiếp với các mạng khác bao gồm cả Internet.

Bộ định tuyến router là một thiết bị nối mạng chuyển tiếp các gói dữ liệu giữa các mạng máy tính . Các loại bộ định tuyến thực hiện các chức năng định hướng lưu lượng trên Internet . Một gói dữ liệu thường được chuyển tiếp từ một thiết bị định tuyến đến một bộ định tuyến khác thông qua các mạng cấu thành một mạng nội bộ cho đến khi nó đến nút đích của nó.

Một bộ định tuyến router sẽ thu hẹp khoảng cách giữa các mạng khác và cho phép truy cập mạng của bạn vào nhiều tính năng hơn, ví dụ như tường lửa, QoS, giám sát lưu lượng truy cập, VPN và hơn thế nữa.

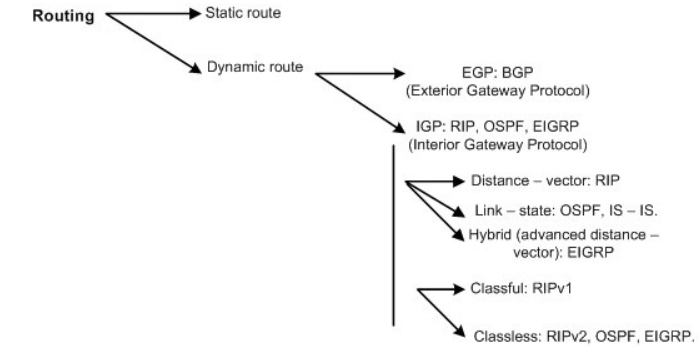
Mininet là một công cụ giả lập mạng, bao gồm tập hợp các hosts đầu cuối, các switches, routers và các liên kết trên một Linux kernel. Mininet sử dụng công nghệ ảo hóa (ở mức đơn giản) để tạo nên hệ thống mạng hoàn chỉnh, chạy chung trên cùng một kernel, hệ thống và user code.

CHƯƠNG 1: TỔNG QUAN VỀ ĐỊNH TUYẾN

* 1. Giới thiệu về định tuyến

Trong ngành [mạng máy tính](https://vi.wikipedia.org/wiki/M%E1%BA%A1ng_m%C3%A1y_t%C3%ADnh), định tuyến (Routing) là quá trình tìm kiếm và xác định đường đi tốt nhất trên một mạng máy tính để gói tin tới được đích thông qua các thiết bị định tuyến. Để làm được điều đó thì các thiết bị định tuyến cần phải dựa vào thông tin bẳng định tuyến (Routing Table) và giao thức định tuyến ( Routing Protocol).

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.



Hình 1.1 Định tuyến

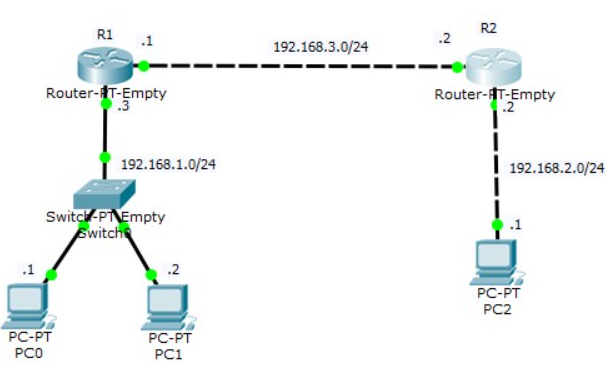
* 1. Phân loại định tuyến

Static route là kỹ thuật mà người quản trị phải tự tay khai báo các route trên các router. Kỹ thuật này đơn giản, dễ thực hiện, ít hao tốn tài nguyên mạng và CPU xử lý trên router (do không phải trao đổi thông tin định tuyến và không phải tính toán định tuyến). Tuy nhiên kỹ thuật này không hội tụ với các thay đổi diễn ra trên mạng và không thích hợp với những mạng có quy mô lớn (khi đó số lượng route quá lớn, không thể khai báo tay được) .

Với dynamic route, các router sẽ trao đổi thông tin định tuyến với nhau. Từ thông tin nhận được, mỗi router sẽ thực hiện tính toán định tuyến từ đó xây dựng bảng định tuyến gồm các đường đi tối ưu nhất đến mọi điểm trong hệ thống mạng. Với Dynamic route, các router phải chạy các giao thức định tuyến (routing protocol).

* + 1. Định tuyến tĩnh

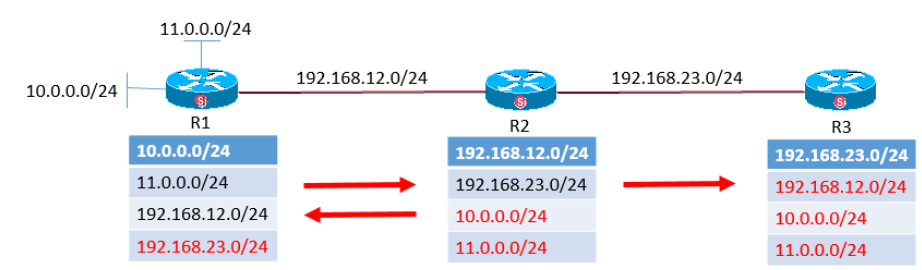
Định tuyến tĩnh là phương pháp định tuyến theo phương thức người quản trị khai báo thông tin định tuyến cho thiết bị định tuyến theo phương thức thủ công.



Hình 1.2.1 Định tuyến tĩnh

* Ưu điểm:
* Sử dụng ít băng thông hơn so với các phương thức định tuyến khác.
* Không tiêu tốn tài nguyên để tính toàn và phân tích gói tin định tuyến.
* Dễ dàng triển khai, cấu hình.
* Có tính bảo mật tốt hơn.
* Nhược điểm:
* Không có khả năng tự động cập nhật đường đi.
* Phải cấu hình thủ công khi mạng có sự thay đổi.
* Khả năng mở rộng kém, phù hợp với mô hình mạng nhỏ.
* Những trường hợp sử dụng định tuyến tĩnh:
* Đường truyền có băng thông thấp.
* Người quản trị cần kiểm soát các kết nối trong hệ thống.
* Hệ thống co các tuyến kết nối ít.
* Kết nối dùng định tuyến tĩnh là đường dự phòng cho đường kết nối dùng giao thức định tuyến động.
* Phương thức triển khai định tuyến tĩnh: Next hop hoặc Exit Interface
* Next hop: thông tin sẽ chuyển đến Router kế tiếp nào trước khi đến đích.
* Exit Interface: thông tin sẽ được đưa ra cổng nào trước khi đến đích.
  + 1. Định tuyến động:

Định tuyến động là phương thức tự động chia sẻ, trao đổi thông tin giữa các thiết bị định tuyến dựa trên các giao thức định tuyến động. Tự động cập nhật thông tin bảng định tuyến nếu hệ thống có sự thay đổi. Tính toán và đưa ra tuyến đường chuyển thông tin tốt nhất.



Hình 1.2.2 Định tuyến động

* Ưu điểm:
* Đơn giản trong việc cấu hình.
* Tự động tìm ra những tuyến đường thay thế khi mạng thay đổi.
* Nhược điểm:
* Yêu cầu xử lí của CPU của router cao hơn so với định tuyến tĩnh.
* Tiêu tốn một phần băng thông trên mạng để xây dựng bảng định tuyến.
  + - 1. **Mục đích của định tuyến động**

Trong một mạng rất lớn có rất nhiều bộ định tuyến như mạng Internet, việc cập nhật bảng định tuyến bằng tay là không thể, vì vậy cần phải có giao thức định tuyến, giao thức định tuyến cho phép các Router xây dựng bảng định tuyến một cách linh hoạt đó là:

* Khám phá mạng từ xa.
* Duy trì việc cập nhật thông tin định tuyến.
* Tính toán và chọn tuyến đường đi tốt nhất đến đích.
* Nếu tuyến đường chuyển thông tin chính bị lỗi, tự tính toán và đưa ra tuyến đường chuyển thông tin backup.
  + - 1. Phân loại định tuyến động

Exterior Gateway Protocols: có giao thức BGP

Interior Gateway Protocols: **Distance Vector** Protocols và **Link- State** Protocols

* Giao thức Distance Vector: có giao thức RIPv1, RIPv2 và IGRP, EIGRP.
* Giao thức Link- State: có giao thức OSPF và IS-IS
  + - 1. Nhưng thông số quan trọng

Metric: là chỉ số riêng của từng giao thức định tuyến động, cho phép tính toán và tìm ra tuyến đường chuyển thông tin tốt nhất.

Metric bao gồm nhiều chỉ số, cụ thể:

* RIP: Hop Count.
* OSPF: Cost
* EIGRP: Bandwidth, Delay, Load, Reliability, MTU

AD- Administrative Distance: là chỉ số ưu tiên của một giao thức trong hệ thống mạng có từ 2 hay nhiều giao thức định tuyến được triển khai.

AD của các giao thức và phương thức định tuyến:

|  |  |
| --- | --- |
| Route Source | Administrative Distance |
| Connected | 0 |
| Static | 1 |
| EIGRP summary route | 5 |
| External BGP | 20 |
| Internal EIGRP | 90 |
| IGRP | 100 |
| OSPF | 110 |
| IS-IS | 115 |
| RIP | 120 |
| External EIGRP | 170 |
| Internal BGP | 200 |

Bảng 1.2.2.3. AD của các giao thức và phương thức định tuyến

CHƯƠNG 2: TỔNG QUAN VỀ MININET

2.1. Giới thiệu về Mininet

Mininet là một trình giả lập mạng*,*hoặc có lẽ chính xác hơn là một hệ thốngdàn dựng mô phỏng mạng. Nó chạy một bộ sưu tập các máy chủ cuối, công tắc, bộ định tuyến và liên kết trên một nhân Linux duy nhất. Nó sử dụng ảo hóa nhẹ để làm cho một hệ thống duy nhất trông giống như một mạng hoàn chỉnh, chạy cùng một hạt nhân, hệ thống và mã người dùng. Một máy chủ Mininet hoạt động giống như một cỗ máy thực sự; bạn có thể vào nó (nếu bạn khởi động và kết nối mạng với máy chủ của bạn) và chạy các chương trình tùy ý (bao gồm bất cứ điều gì được cài đặt trên hệ thống Linux cơ bản.) Các chương trình bạn chạy có thể gửi các gói thông qua những gì có vẻ như một giao diện Ethernet thực sự, với tốc độ liên kết nhất định và độ trễ. Các gói được xử lý bởi những gì trông giống như một công tắc Ethernet thực sự, bộ định tuyến hoặc hộp giữa, với một số lượng nhất định của hàng đợi. Khi hai chương trình, như máy khách và máy chủ, giao tiếp thông qua Mininet, hiệu suất đo được sẽ phù hợp với hai máy gốc (chậm hơn).

Nói tóm lại, máy chủ ảo, công tắc, liên kết và bộ điều khiển ảo của Mininet là điều thực sự - chúng chỉ được tạo ra bằng phần mềm chứ không phải phần cứng - và phần lớn hành vi của chúng tương tự như các yếu tố phần cứng rời rạc. Thông thường có thể tạo ra một mạng Mininet giống như một mạng phần cứng hoặc một mạng phần cứng giống như mạng Mininet và chạy cùng một mã nhị phân và các ứng dụng trên một trong hai nền tảng.

2.2. Một số chức năng của Mininet

Cung cấp một network testbed đơn giản và rẻ (do không tốn kém chi phí mua các switch Openflow) để phát triển các ứng dụng mạng. Do các switch Openflow trong Mininet có tất cả các tính chất mà switch OpenFlow thật có được nên việc sử dụng mạng emulator bằng Mininet là có ý nghĩa về mặt thực tế.

* Cho phép các nhà phát triển ứng dụng làm việc đồng thời, một cách độc lập trên cùng đồ hình.
* Cho phép kiểm thử các đồ hình phức tạp mà không cần phải nối dây cho mạng vật lý.
* Cho phép debug và chạy các test của các mạng lớn, sử dụng CLI.
* Hỗ trợ thiết lập các đồ hình tùy biến bất kỳ, gồm tập cơ bản các thông số đồ hình.
* Có thể đem các ứng dụng trên mininet đi triển khai trên mạng thật với code hoàn toàn không cần thay đổi.
* Cung cấp Python API dễ dàng sử dụng và có khả năng mở rộng.

Mininet cho ta một phương pháp dễ dàng để thu được chính xác các đặc tính mạng và thử nghiệm các đồ hình tùy ý. Quan trọng nhất, code mà chúng ta phát triển và kiểm tra bằng Mininet cho Openflow controller, switch hay host, có thể đưa ra triển khai với hệ thống phần cứng thật không cần thay đổi.

2.3. Hạn chế của Mininet

Chạy trên một hệ thống đơn lẻ rất thuận tiện, nhưng nó áp đặt giới hạn tài nguyên: nếu máy chủ của bạn có CPU 3 GHz và có thể chuyển đổi khoảng 10 Gbps lưu lượng truy cập mô phỏng, thì các tài nguyên đó sẽ cần được cân bằng và chia sẻ giữa các máy chủ ảo và thiết bị chuyển mạch của bạn.

Mininet sử dụng một nhân Linux duy nhất cho tất cả các máy chủ ảo; điều này có nghĩa là bạn không thể chạy phần mềm phụ thuộc vào BSD, Windows hoặc các nhân hệ điều hành khác. (Mặc dù bạn có thể gắn máy ảo vào Mininet.)

Mininet sẽ không viết bộ điều khiển OpenFlow cho bạn; nếu bạn cần hành vi chuyển đổi hoặc định tuyến tùy chỉnh, bạn sẽ cần tìm hoặc phát triển một bộ điều khiển với các tính năng bạn yêu cầu.

Theo mặc định, mạng Mininet của bạn được cách ly khỏi mạng LAN và với internet - đây thường là một điều tốt! Tuy nhiên, bạn có thể sử dụng NATđối tượng và / hoặc --nattùy chọn để kết nối mạng Mininet với mạng LAN của bạn thông qua Dịch địa chỉ mạng. Bạn cũng có thể đính kèm giao diện phần cứng thực (hoặc ảo) vào mạng Mininet của mình (xem examples/hwintf.pyđể biết chi tiết.)

Theo mặc định, tất cả các máy chủ Mininet chia sẻ hệ thống tệp máy chủ và không gian PID; điều này có nghĩa là bạn có thể phải cẩn thận nếu bạn đang chạy các daemon yêu cầu cấu hình trong / etc, và bạn cần phải cẩn thận để không giết nhầm các quy trình do nhầm lẫn. (Lưu ý rằng bind.py ví dụ minh họa cách có các thư mục riêng trên mỗi máy chủ lưu trữ.)

Không giống như một trình mô phỏng, Mininet không có khái niệm mạnh mẽ về thời gian ảo; điều này có nghĩa là các phép đo thời gian sẽ dựa trên thời gian thực và kết quả nhanh hơn thời gian thực (ví dụ: mạng 100 Gbps) không thể dễ dàng được mô phỏng.

CHƯƠNG 3: TÌM HIỂU CONTROLLER POX

3.1. Controller POX là gì?

POX là SDN Controller đươc viết trên nền tảng Python phục vụ cho việc nghiên cứu và giáo dục. Để cài đặt POX, ta cần làm như sau:

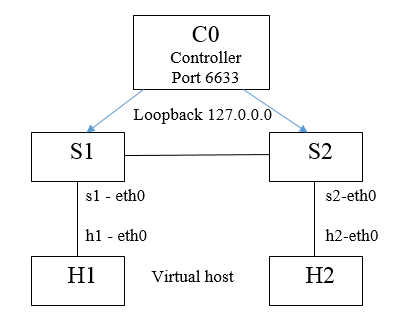
**$git clone** [**http://github.com/noxrepo/pox**](http://github.com/noxrepo/pox)

**$cd pox**

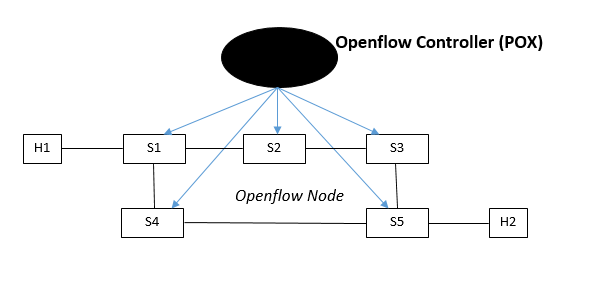
Sau khi khởi tạo được Topo bằng Mininet, để các Host có thể ping thông được với nhau chúng ta cần phải dựng POX lên. POX có nhiệm vụ như một Controller giúp điều khiển các Flow trong Topo. Ví dụ:

**mininet> exit**

**$ sudo mn --topo linear --switch ovsk --controller remote**

Lệnh trên sẽ tạo 1 Topo với 2 Switch, mỗi Switch sẽ kết nối với 1 Host. Sau đó chúng ta sử dụng pingall để kiểm tra kết nối giữa các host:

Hình 3.1

3.2. Mô phỏng:

Hình 3.2: Mô phỏng POX

Sử dụng python để tạo Topo như yêu cầu:

**from mininet.topo import Topo**

**class mytopo(Topo):**

**"Topo 2 host and 6 switches."**

**def \_\_init\_\_( self ):**

**Topo.\_\_init\_\_( self )**

**#add host + switches**

**h1=self.addHost('h1')**

**h2=self.addHost('h2')**

**s1=self.addSwitch('s1')**

**s2=self.addSwitch('s2')**

**s3=self.addSwitch('s3')**

**s4=self.addSwitch('s4')**

**s5=self.addSwitch('s5')**

**#add links**

**self.addLink(h1,s1)**

**self.addLink(h2,s5)**

**self.addLink(s1,s2)**

**self.addLink(s1,s4)**

**self.addLink(s4,s5)**

**self.addLink(s2,s3)**

**self.addLink(s3,s5)**

**topos = {'mytopo':(lambda:mytopo() ) }**

Để topo có thể chạy được, ta cần dựng 1 POX để làm Controller của Topo:

**~/pox$ ./pox.py openflow.discovery forwarding.l2\_learning openflow.spanning\_tree**

**Openflow.discovery**

Module Discovery này giúp tìm ra các kết nối giữa các OpenFlow Switch bằng các gửi đi các LLDP packets và theo dõi xem chúng có đển nơi không. Để biết được các thông tin gửi về, module này lắng nghe các LinkEvents ở core.openflow\_discovery.

**Forwarding.l2\_learning**

Learning switch liên kết trực tiếp với một OpenFlow switch

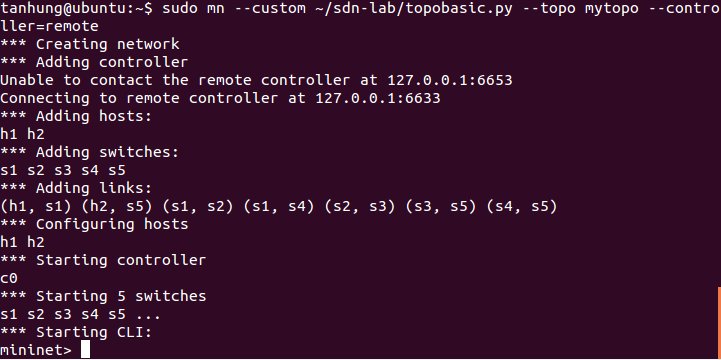
Khi switch gặp 1 gói tin,nó sẽ muốn tìm 1 port nào đó để đẩy gói tin đến đích. Để thực hiện điểu đó, Switch tạo 1 bảng, bảng đó có chức năng tham chiếu 1 địa chỉ với port tương ứng. Khi muốn chuyển tiếp lưu lượng, switch sẽ nhìn vào phần đích trong Table để chuyển đến Port tương ứng. Nếu không biết phải chuyển ra port nào, nó sẽ gửi bản tin nhận được ra tất cả các port.

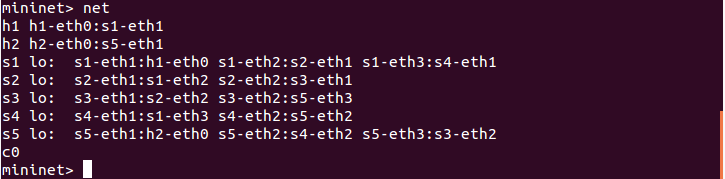
**Openflow.spaning\_tree**

Module này giúp tạo Spanning Tree. Module sẽ xác định các phần tử tạo nên Topo mạng, cấu trúc của Spanning Tree và vô hiệu hóa Flooding trên các Port của switch. Kết quả thu đươc là 1 Topo hạn chế được tối thiểu các vòng lặp không mong muốn.

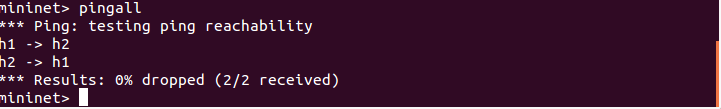
Sau khi dựng được POX, ta khởi động mininet để dựng Topo theo yêu cầu:

**sudo mn --custom ~/sdn-lab/topobasic.py --topo mytopo --controller=remote**





Kiểm tra liên kết giữa các node



CHƯƠNG IV: LẬP TRÌNH MÔ PHỎNG ĐỊNH TUYẾN TRÊN MININET

4.1. Các câu lệnh cơ bản trong Mininet

**$sudo mn**

Khởi động mininet và tạo 1 Topo mặc định bao gồm một OpenFlow switch kết nối với 2 host và 1 Controller. Topo này cũng có thể được gọi ra bằng –topo=minimal.

Hiển thị các câu lệnh trong CLI:

**mininet>help**

Hiển thị node:

**mininet>nodes**

Hiển thị liên kết:

**mininet>dump**

Nết tham số đầu tiên trong lệnh là tên của Host, Switch hay Controller thì lệnh sẽ được thực hiện trên node đó. Ví dụ như:

**mininet>h1 ifconfig –a**

Kiểm tra kết nối giữa các host:

Để kiểm tra kết nối giữa host 1 và host 2:

**mininet>h1 ping –c 1 h2**

Hoặc để kiểm tra kết nối giữa tất cả các cặp host:

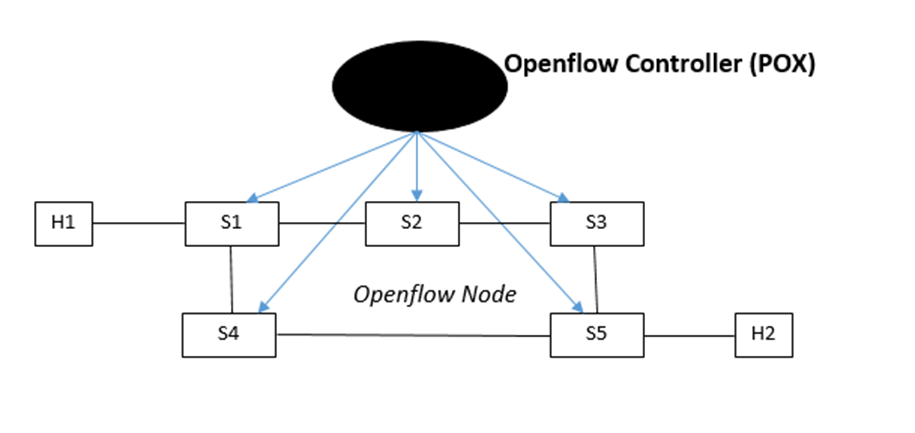
**mininet>pingall**

Nếu Mininet bị crash vì 1 lý do nào đó ta dùng lệnh để dọn sạch Mininet:

**$sudo mn –c**

Để thoát mininet:

**mininet>exi**

4.2 Mô phỏng định tuyến trên Mininet:

Ta dựng 1 POX để làm Controller cho topo

**~/pox$ ./pox.py forwarding.l2\_routing**

Trong đó module **forwarding.l2\_learning** là module thay thế cho 3 **module openflow.discovery, forwarding.l2\_learning, openflow.spanning\_tree.**

**from datetime import datetime**

**from pox.lib.revent.revent import EventMixin, Event**

**import pox.lib.util as util**

**from pox.core import core**

**import pox.openflow.libopenflow\_01 as of**

**from collections import defaultdict**

**import pox.lib.packet as pkt**

**from collections import namedtuple**

**log = core.getLogger()**

**switches = {}**

**switch\_ports = {}**

**adj = defaultdict(lambda:defaultdict(lambda:None))**

**mac\_learning = {}**

**class ofp\_match\_withHash(of.ofp\_match):**

**##Our additions to enable indexing by match specifications**

**@classmethod**

**def from\_ofp\_match\_Superclass(cls, other):**

**match = cls()**

**match.wildcards = other.wildcards**

**match.in\_port = other.in\_port**

**match.dl\_src = other.dl\_src**

**match.dl\_dst = other.dl\_dst**

**match.dl\_vlan = other.dl\_vlan**

**match.dl\_vlan\_pcp = other.dl\_vlan\_pcp**

**match.dl\_type = other.dl\_type**

**match.nw\_tos = other.nw\_tos**

**match.nw\_proto = other.nw\_proto**

**match.nw\_src = other.nw\_src**

**match.nw\_dst = other.nw\_dst**

**match.tp\_src = other.tp\_src**

**match.tp\_dst = other.tp\_dst**

**return match**

**def \_\_hash\_\_(self):**

**return hash((self.wildcards, self.in\_port, self.dl\_src, self.dl\_dst, self.dl\_vlan, self.dl\_vlan\_pcp, self.dl\_type, self.nw\_tos, self.nw\_proto, self.nw\_src, self.nw\_dst, self.tp\_src, self.tp\_dst))**

**class Path(object):**

**def \_\_init\_\_(self, src, dst, prev, first\_port):**

**self.src = src**

**self.dst = dst**

**self.prev = prev**

**self.first\_port = first\_port**

**def \_\_repr\_\_(self):**

**ret = util.dpid\_to\_str(self.dst)**

**u = self.prev[self.dst]**

**while(u != None):**

**ret = util.dpid\_to\_str(u) + "->" + ret**

**u = self.prev[u]**

**return ret**

**def \_tuple\_me(self):**

**list = [self.dst,]**

**u = self.prev[self.dst]**

**while u != None:**

**list.append(u)**

**u = self.prev[u]**

**#log.debug("List path: %s", list)**

**#log.debug("Tuple path: %s", tuple(list))**

**return tuple(list)**

**def \_\_hash\_\_(self):**

**return hash(self.\_tuple\_me())**

**def \_\_eq\_\_(self, other):**

**return self.\_tuple\_me() == other.\_tuple\_me()**

**def \_get\_path(src, dst):**

**#Bellman-Ford algorithm**

**keys = switches.keys()**

**distance = {}**

**previous = {}**

**for dpid in keys:**

**distance[dpid] = float("+inf")**

**previous[dpid] = None**

**distance[src] = 0**

**for i in range(len(keys)-1):**

**for u in adj.keys(): #nested dict**

**for v in adj[u].keys():**

**w = 1**

**if distance[u] + w < distance[v]:**

**distance[v] = distance[u] + w**

**previous[v] = u**

**for u in adj.keys(): #nested dict**

**for v in adj[u].keys():**

**w = 1**

**if distance[u] + w < distance[v]:**

**log.error("Graph contains a negative-weight cycle")**

**return None**

**first\_port = None**

**v = dst**

**u = previous[v]**

**while u is not None:**

**if u == src:**

**first\_port = adj[u][v]**

**v = u**

**u = previous[v]**

**return Path(src, dst, previous, first\_port) #path**

**def \_install\_path(prev\_path, match):**

**dst\_sw = prev\_path.dst**

**cur\_sw = prev\_path.dst**

**dst\_pck = match.dl\_dst**

**msg = of.ofp\_flow\_mod()**

**msg.match = match**

**msg.idle\_timeout = 10**

**msg.flags = of.OFPFF\_SEND\_FLOW\_REM**

**msg.actions.append(of.ofp\_action\_output(port = mac\_learning[dst\_pck].port))**

**log.debug("Installing forward from switch %s to output port %s", util.dpid\_to\_str(cur\_sw), mac\_learning[dst\_pck].port)**

**switches[dst\_sw].connection.send(msg)**

**next\_sw = cur\_sw**

**cur\_sw = prev\_path.prev[next\_sw]**

**while cur\_sw is not None: #for switch in path.keys():**

**msg = of.ofp\_flow\_mod()**

**msg.match = match**

**msg.idle\_timeout = 10**

**msg.flags = of.OFPFF\_SEND\_FLOW\_REM**

**log.debug("Installing forward from switch %s to switch %s output port %s", util.dpid\_to\_str(cur\_sw), util.dpid\_to\_str(next\_sw), adj[cur\_sw][next\_sw])**

**msg.actions.append(of.ofp\_action\_output(port = adj[cur\_sw][next\_sw]))**

**switches[cur\_sw].connection.send(msg)**

**next\_sw = cur\_sw**

**cur\_sw = prev\_path.prev[next\_sw]**

**def \_print\_rev\_path(dst\_pck, src, dst, prev\_path):**

**str = "Reverse path from %s to %s over: [%s->dst over port %s]" % (util.dpid\_to\_str(src), util.dpid\_to\_str(dst), util.dpid\_to\_str(dst), mac\_learning[dst\_pck].port)**

**next\_sw = dst**

**cur\_sw = prev\_path[next\_sw]**

**while cur\_sw != None: #for switch in path.keys():**

**str += "[%s->%s over port %s]" % (util.dpid\_to\_str(cur\_sw), util.dpid\_to\_str(next\_sw), adj[cur\_sw][next\_sw])**

**next\_sw = cur\_sw**

**cur\_sw = prev\_path[next\_sw]**

**log.debug(str)**

**class NewFlow(Event):**

**def \_\_init\_\_(self, prev\_path, match, adj):**

**Event.\_\_init\_\_(self)**

**self.match = match**

**self.prev\_path = prev\_path**

**self.adj = adj**

**class Switch(EventMixin):**

**\_eventMixin\_events = set([**

**NewFlow,**

**])**

**def \_\_init\_\_(self, connection):**

**self.connection = connection**

**connection.addListeners(self)**

**for p in self.connection.ports.itervalues(): #Enable flooding on all ports until they are classified as links**

**self.enable\_flooding(p.port\_no)**

**def \_\_repr\_\_(self):**

**return util.dpid\_to\_str(self.connection.dpid)**

**def disable\_flooding(self, port):**

**msg = of.ofp\_port\_mod(port\_no = port,**

**hw\_addr = self.connection.ports[port].hw\_addr,**

**config = of.OFPPC\_NO\_FLOOD,**

**mask = of.OFPPC\_NO\_FLOOD)**

**self.connection.send(msg)**

**def enable\_flooding(self, port):**

**msg = of.ofp\_port\_mod(port\_no = port,**

**hw\_addr = self.connection.ports[port].hw\_addr,**

**config = 0, # opposite of of.OFPPC\_NO\_FLOOD,**

**mask = of.OFPPC\_NO\_FLOOD)**

**self.connection.send(msg)**

**def \_handle\_PacketIn(self, event):**

**def forward(port):**

**"""Tell the switch to forward the packet"""**

**msg = of.ofp\_packet\_out()**

**msg.actions.append(of.ofp\_action\_output(port = port))**

**if event.ofp.buffer\_id is not None:**

**msg.buffer\_id = event.ofp.buffer\_id**

**else:**

**msg.data = event.ofp.data**

**msg.in\_port = event.port**

**self.connection.send(msg)**

**def flood():**

**"""Tell all switches to flood the packet, remember that we disable inter-switch flooding at startup"""**

**#forward(of.OFPP\_FLOOD)**

**for (dpid,switch) in switches.iteritems():**

**msg = of.ofp\_packet\_out()**

**if switch == self:**

**if event.ofp.buffer\_id is not None:**

**msg.buffer\_id = event.ofp.buffer\_id**

**else:**

**msg.data = event.ofp.data**

**msg.in\_port = event.port**

**else:**

**msg.data = event.ofp.data**

**ports = [p for p in switch.connection.ports if (dpid,p) not in switch\_ports]**

**if len(ports) > 0:**

**for p in ports:**

**msg.actions.append(of.ofp\_action\_output(port = p))**

**switches[dpid].connection.send(msg)**

**def drop():**

**"""Tell the switch to drop the packet"""**

**if event.ofp.buffer\_id is not None: #nothing to drop because the packet is not in the Switch buffer**

**msg = of.ofp\_packet\_out()**

**msg.buffer\_id = event.ofp.buffer\_id**

**event.ofp.buffer\_id = None # Mark as dead, copied from James McCauley, not sure what it does but it does not work otherwise**

**msg.in\_port = event.port**

**self.connection.send(msg)**

**#log.debug("Received PacketIn")**

**packet = event.parsed**

**SwitchPort = namedtuple('SwitchPoint', 'dpid port')**

**if (event.dpid,event.port) not in switch\_ports: # only relearn locations if they arrived from non-interswitch links**

**mac\_learning[packet.src] = SwitchPort(event.dpid, event.port) #relearn the location of the mac-address**

**if packet.effective\_ethertype == packet.LLDP\_TYPE:**

**drop()**

**log.debug("Switch %s dropped LLDP packet", self)**

**elif packet.dst.is\_multicast:**

**flood()**

**#log.debug("Switch %s flooded multicast 0x%0.4X type packet", self, packet.effective\_ethertype)**

**elif packet.dst not in mac\_learning:**

**flood() #Let's first learn the location of the recipient before generating and installing any rules for this. We might flood this but that leads to further complications if half way the flood through the network the path has been learned.**

**log.debug("Switch %s flooded unicast 0x%0.4X type packet, due to unlearned MAC address", self, packet.effective\_ethertype)**

**elif packet.effective\_ethertype == packet.ARP\_TYPE:**

**#These packets are sent so not-often that they don't deserve a flow**

**#Instead of flooding them, we drop it at the current switch and have it resend by the switch to which the recipient is connected.**

**#flood()**

**drop()**

**dst = mac\_learning[packet.dst]**

**#print dst.dpid, dst.port**

**msg = of.ofp\_packet\_out()**

**msg.data = event.ofp.data**

**msg.actions.append(of.ofp\_action\_output(port = dst.port))**

**switches[dst.dpid].connection.send(msg)**

**log.debug("Switch %s processed unicast ARP (0x0806) packet, send to recipient by switch %s", self, util.dpid\_to\_str(dst.dpid))**

**else:**

**log.debug("Switch %s received PacketIn of type 0x%0.4X, received from %s.%s", self, packet.effective\_ethertype, util.dpid\_to\_str(event.dpid), event.port)**

**dst = mac\_learning[packet.dst]**

**prev\_path = \_get\_path(self.connection.dpid, dst.dpid)**

**if prev\_path is None:**

**flood()**

**return**

**log.debug("Path from %s to %s over path %s", packet.src, packet.dst, prev\_path)**

**match = ofp\_match\_withHash.from\_packet(packet)**

**\_install\_path(prev\_path, match)**

**#forward the packet directly from the last switch, there is no need to have the packet run through the complete network.**

**drop()**

**dst = mac\_learning[packet.dst]**

**msg = of.ofp\_packet\_out()**

**msg.data = event.ofp.data**

**msg.actions.append(of.ofp\_action\_output(port = dst.port))**

**switches[dst.dpid].connection.send(msg)**

**self.raiseEvent(NewFlow(prev\_path, match, adj))**

**log.debug("Switch %s processed unicast 0x%0.4x type packet, send to recipient by switch %s", self, packet.effective\_ethertype, util.dpid\_to\_str(dst.dpid))**

**def \_handle\_ConnectionDown(self, event):**

**log.debug("Switch %s going down", util.dpid\_to\_str(self.connection.dpid))**

**del switches[self.connection.dpid]**

**#pprint(switches)**

**class NewSwitch(Event):**

**def \_\_init\_\_(self, switch):**

**Event.\_\_init\_\_(self)**

**self.switch = switch**

**class Forwarding(EventMixin):**

**\_core\_name = "myforwarding"**

**\_eventMixin\_events = set([NewSwitch,])**

**def \_\_init\_\_ (self):**

**log.debug("Forwarding is initialized")**

**def startup():**

**core.openflow.addListeners(self)**

**core.openflow\_discovery.addListeners(self)**

**log.debug("Forwarding started")**

**core.call\_when\_ready(startup, 'openflow', 'openflow\_discovery')**

**def \_handle\_LinkEvent(self, event):**

**link = event.link**

**if event.added:**

**log.debug("Received LinkEvent, Link Added from %s to %s over port %d", util.dpid\_to\_str(link.dpid1), util.dpid\_to\_str(link.dpid2), link.port1)**

**adj[link.dpid1][link.dpid2] = link.port1**

**switch\_ports[link.dpid1,link.port1] = link**

**else:**

**log.debug("Received LinkEvent, Link Removed from %s to %s over port %d", util.dpid\_to\_str(link.dpid1), util.dpid\_to\_str(link.dpid2), link.port1)**

**def \_handle\_ConnectionUp(self, event):**

**log.debug("New switch connection: %s", event.connection)**

**sw = Switch(event.connection)**

**switches[event.dpid] = sw;**

**self.raiseEvent(NewSwitch(sw))**

**def launch (postfix=datetime.now().strftime("%Y%m%d%H%M%S")):**

**from log.level import launch**

**launch(DEBUG=True)**

**from samples.pretty\_log import launch**

**launch()**

**#from openflow.keepalive import launch**

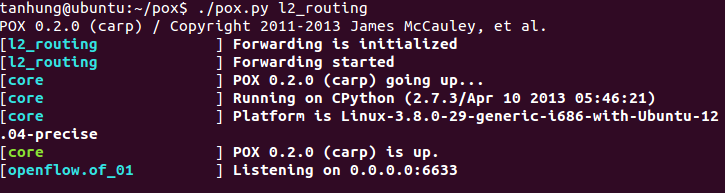
**#launch(interval=15) # 15 seconds**

**from openflow.discovery import launch**

**launch()**

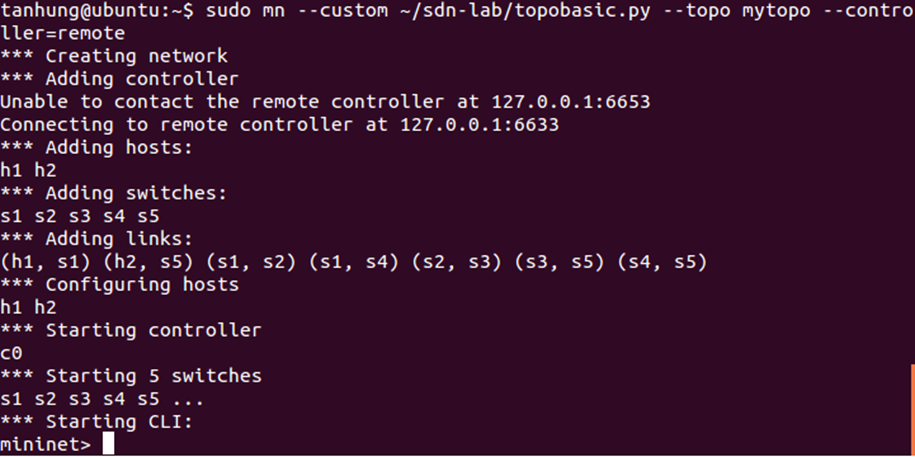
**core.registerNew(Forwarding)**

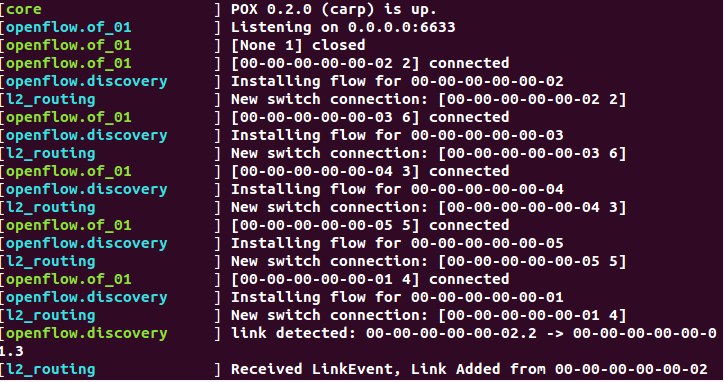
Kêt quả thu được:



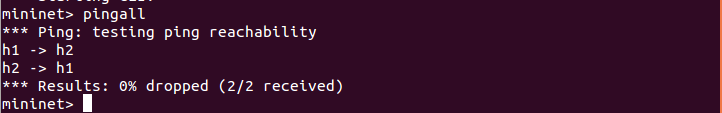
Sau khi dựng được POX, ta khởi động mininet để dựng Topo theo yêu cầu:

**sudo mn --custom ~/sdn-lab/topobasic.py --topo mytopo --controller=remote**

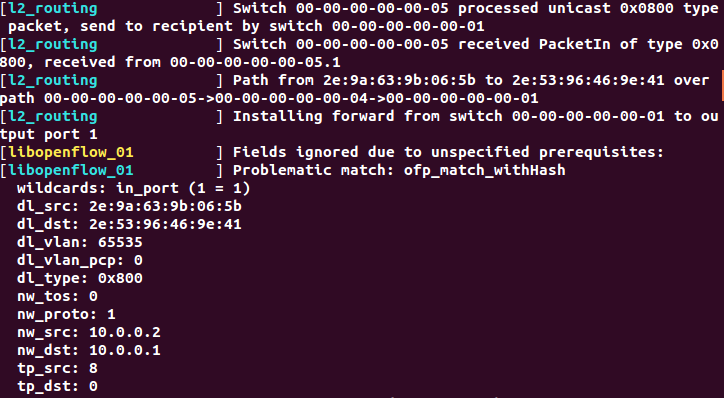




Thực hiện lệnh **pingall** trên mininet để kiểm tra các node



Khi đó trên POX ta được



**TÀI LIỆU THAM KHẢO**

1. <https://github.com/mininet/mininet/wiki/Introduction-to-Mininet>

2. Python - Nеtwork Progrаmming, Еngr. Michаеl Dаvid.

3. Software Defined Networks using Mininet, Pramod B Patil., Kanchan S. Bhagat.,

D K Kirange, S. D. Patil, 2020.