

ĐẠI HỌC QUỐC GIA HÀ NỘI TRƯỜNG ĐẠI HỌC CÔNG NGHỆ



BÁO CÁO PROJECT CUỐI KỲ

MÔN: MẠNG ĐIỀU KHIỂN MỀM

CHỦ ĐỀ:

**SDN ROUTING: TÌM ĐƯỜNG ĐI NGẮN NHẤT, PHÁT HIỆN ĐƯỜNG LINK BỊ
SẬP VÀ GỬI TIN KHI XUẤT HIỆN CÁC HOST KHÔNG ĐÁNG TIN DỰA
TRÊN POX CONTROLLER**

NHÓM THỰC HIỆN: NHÓM 7

Lê Vũ Tuấn Anh - 20021486

Nguyễn Như Phúc - 20020091

Nguyễn Quang Vinh - 20021601

Lâm Thiên Phong - 20021567

Tóm tắt

Sự tiến bộ của Mạng máy tính đã cách mạng hóa công nghệ truyền thông và mở ra một cách tốt hơn và hiệu quả hơn để chia sẻ và quản lý dữ liệu. Thiết bị mạng truyền thống, thiết kế và triển khai là đáng chú ý, nhưng trong thời đại này, nó không đủ. Mạng được xác định bằng phần mềm (SDN) là một công nghệ mới, phù hợp và phát triển, hoàn thiện để đáp ứng các yêu cầu ngày càng cao của viễn thông hiện đại. Do tính linh hoạt và tính khả dụng của nó, SDN đang được triển khai trong mạng cấp nhà cung cấp dịch vụ. Tính sẵn sàng cao đòi hỏi phải phục hồi nhanh hơn các liên kết bị lỗi và việc định tuyến để có thể tăng cường bảo mật và việc có thể gửi các bản tin một cách nhanh nhất. Do đó, một số phương pháp đã được đề xuất và thực hiện mô phỏng trong bài báo cáo này, bằng cách xử lý riêng rẽ các bài toán đơn giản trong bài toán định tuyến lớn: truyền tin khi có một host nguy hiểm, phát hiện đường dẫn bị sập và tìm cách khôi phục, cùng với đó là việc tìm đường đi ngắn nhất để gửi các gói tin qua mạng SDN bằng thuật toán Dijkstra. Qua đó, có thể kết hợp các bài toán lại để có thể giải quyết một bài toán định tuyến đơn giản. Các bài toán được thực hiện trong bài báo cáo sẽ được thực hiện trên POX controller.

Keyword: SDN, định tuyến, thuật toán Dijkstra, host không đáng tin, đường dẫn bị sập, Pox controller

Mục lục

Tóm tắt	2
Mục lục	3
Danh mục hình vẽ, biểu đồ	4
1: Giới thiệu	5
2: Nội dung bài báo cáo	5
2.1: Các nghiên cứu liên quan	5
2.2: Tổng qua về SDN	6
2.2.1: SDN	6
2.2.2: Pox controller	8
2.2.3: Routing in SDN	9
2.3: Bài toán phát hiện đường dẫn bị hỏng, và tìm đường đi ngắn nhất của bản tin trong mạng SDN	10
2.3.1: Giới thiệu bài toán	10
2.3.2: Cơ sở lý thuyết	10
2.3.3: Mô hình, thuật toán, mô phỏng	11
2.3.4: Kết quả, đánh giá	15
2.4: Bài toán truyền tin khi có host không đáng tin	18
2.4.1: Giới thiệu bài toán	18
2.4.2: Cơ sở lý thuyết	18
2.4.3: Mô hình, thuật toán, mô phỏng	20
2.4.4: Kết quả, đánh giá	21
3: Đánh giá	22
4: Kết luận	23
5: Tài liệu tham khảo	23

Danh mục hình vẽ, biểu đồ

Hình 1: Mạng SDN và mạng truyền thống	7
Hình 2: Cấu trúc của mạng SDN.....	7
Hình 3: Sơ đồ xử lý bài toán.....	12
Hình 4: Giao thức được tạo bởi Miniedit.....	13
Hình 5: Dijitra Algorithm.....	14
Hình 6: Chạy Pox controller cùng với những thành phần	15
Hình 7: Ping từ host 1 đến host 6.....	16
Hình 8: Nét đứt màu xanh đại diện cho những link bị down.....	16
Hình 9: Nhật ký của openflow.Discovery và forwarding.l2._learning.....	17
Hình 10: Đồ thị so sánh giữa thời gian truyền package khi sử dụng và không sử dụng thuật toán Dijitra.....	17
Hình 11: Giao thức xử lý bài toán untrusted host	20
Hình 12: Bảng lưu lượng bài toán untrusted host	22
 Bảng 1: Xử lý gói tin qua SDN	19

1: Giới thiệu

Mạng SDN (Software-Defined Networking) là một kiến trúc mạng linh hoạt và dễ dàng điều khiển, trong đó bộ điều khiển tách biệt với phần cứng mạng. Tuy nhiên, việc bảo mật hay định tuyến vẫn còn nhiều vấn đề. Việc truyền đi và nhận các bản tin đến từ các máy chủ không đáng tin trong mạng SDN vẫn tồn tại một số nguy cơ bị tấn công hệ thống. Kẻ tấn công có thể giả mạo địa chỉ MAC hoặc địa chỉ IP của một thiết bị trong mạng để gửi các gói tin, làm cho bộ điều khiển SDN định tuyến sai hoặc cho phép kẻ tấn công tiếp cận các tài nguyên mạng trái phép. Ngoài ra, kẻ tấn công có thể gửi nhiều yêu cầu tới bộ điều khiển SDN để làm cho nó quá tải và gây ra sự cố về hoạt động mạng. Cùng với đó, việc xuất hiện các node mạng bị sập có thể dẫn tới việc bản tin không thể truyền đi. Do đó, trong bài báo cáo này sẽ tập trung xử lý bài toán truyền tin đi khi xuất hiện các host đáng ngờ, cùng với việc phát hiện các đường dẫn bị sập trong mạng và tìm đường đi ngắn nhất để truyền bản tin trong một hệ thống mạng phức tạp.

Do trong mạng SDN thì thành phần định tuyến đã được tách khỏi switch nên ta không cần thực hiện những bài toán được nêu trên từng switch mà ta chỉ cần thực hiện trên controller bằng cách can thiệp vào các thành phần bên trong Pox controller. Qua việc thực hiện những bài toán trên, bài báo cáo hy vọng đóng góp phần nhỏ vào công việc định tuyến của SDN qua đó giúp mạng SDN ngày càng hoàn thiện.

2: Nội dung bài báo cáo

2.1: Các nghiên cứu liên quan

Bài báo "Link Failure Recovery Using Shortest Path Fast Rerouting Technique in SDN"^[3] của 2 tác giả V. Muthumanikandan & C. Valliyammai trình bày một phương pháp phục hồi sự cố mất kết nối trong mạng SDN bằng cách sử dụng kỹ thuật định tuyến nhanh trên đường tuyến ngắn nhất. Phương pháp này sử dụng mô hình mạng đường tuyến ngắn nhất và thuật toán định tuyến tiết kiệm chi phí (Cost-Saving Routing Algorithm) để tìm đường tuyến ngắn nhất giữa các thiết bị mạng.

Bài báo "RL-Routing: An SDN Routing Algorithm Based on Deep Reinforcement Learning"^[4] của tác giả Yi-Ren Chen và đội ngũ nghiên cứu, giới thiệu một thuật toán định tuyến mới cho mạng SDN, dựa trên học tăng cường sâu (Deep Reinforcement Learning - DRL). Thuật toán này được gọi là RL-Routing. Điểm khác biệt của RL-Routing so với các thuật toán định tuyến truyền thống là nó sử dụng học tăng cường để tìm ra đường tuyến tối ưu, dựa trên việc tối ưu hóa mục tiêu được đưa ra. Thuật toán này sử dụng mô hình Markov Decision Process (MDP) để mô hình hóa quá trình định tuyến và tối ưu hóa đường tuyến thông qua một mô hình học tăng cường sâu (DRL).

Bài báo "Cost optimization of secure routing with untrusted devices in software defined networking" ^[5] của tác giả Abbas Yazdinejad và đội ngũ nghiên cứu của mình, bài báo

trình bày vấn đề của việc định tuyến an toàn trong mạng SDN, nơi mà các thiết bị có thể bị tin tặc tấn công. Bài báo đề xuất một giải pháp định tuyến an toàn mới, sử dụng một phương pháp tối ưu chi phí để giải quyết vấn đề này. Thuật toán tối ưu chi phí này sử dụng một mô hình toán học để tính toán các đường tuyến an toàn, đồng thời tối ưu chi phí cần thiết để duy trì an toàn và hiệu suất của mạng.

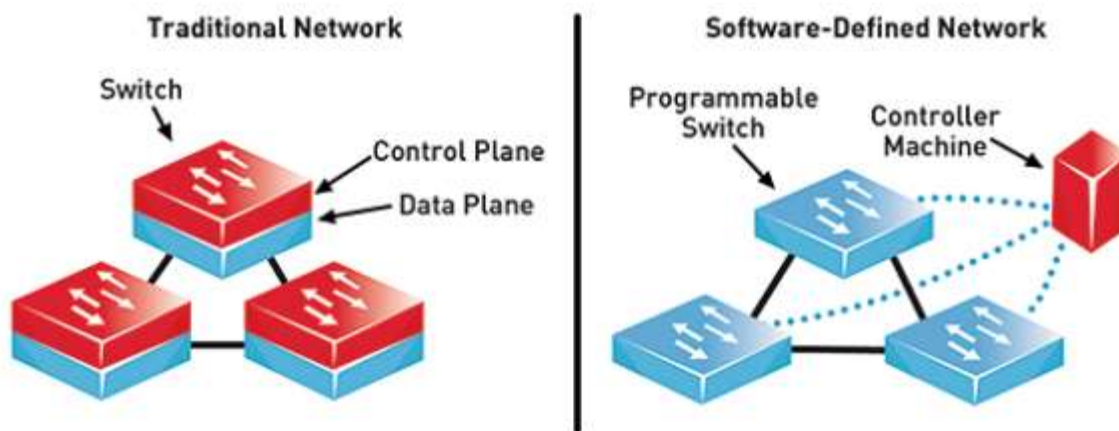
Bài báo "Demonstration of single link failure recovery using Bellman Ford and Dijkstra algorithm in SDN" ^[6] của những tác giả Syed Waleed; Muhammad Faizan; Maheen Iqbal; Muhammad Irfan Anis, trình bày về việc triển khai các giải thuật Bellman Ford và Dijkstra trong mô hình SDN để phục hồi mạng trong trường hợp một liên kết bị lỗi. Bài báo giới thiệu kiến trúc mạng, giải thuật Bellman Ford và Dijkstra, cách triển khai giải thuật và kết quả thực nghiệm. Bài báo cũng đưa ra một số hướng phát triển và cải tiến cho giải thuật Bellman Ford để cải thiện hiệu suất phục hồi mạng trong các trường hợp đặc biệt.

Bài báo "Link Failure Recovery in SDN: High Efficiency, Strong Scalability and Wide Applicability"^[7] của đội ngũ tác giả Jue Chen, Jinbang Chen, Junchen Ling, Junlong Zhou, and Wei Zhang, trình bày về một phương pháp phục hồi mạng SDN sau khi liên kết bị lỗi, với tính hiệu quả cao, khả năng mở rộng mạnh và sự áp dụng rộng rãi. Bài báo giới thiệu các giải thuật sử dụng trong phương pháp phục hồi này, bao gồm giải thuật kết hợp định tuyến dựa trên Dijkstra và Bellman-Ford (DFR) và giải thuật chuyển hướng cục bộ dựa trên đồ thị (LGD).

2.2: Tổng qua về SDN

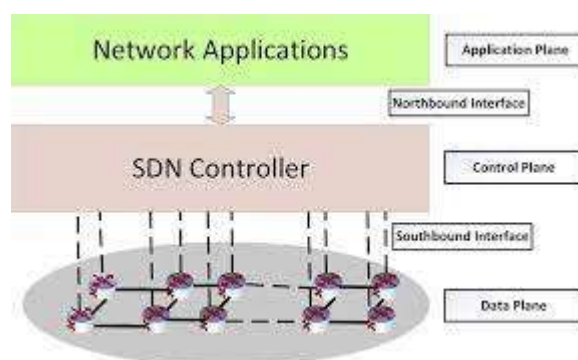
2.2.1: SDN

Mạng được xác định bằng phần mềm (SDN) là một mô hình cho phép tách mặt phẳng điều khiển và dữ liệu trong mạng. SDN thực hiện điều đó bằng cách trích xuất các chức năng mặt phẳng điều khiển từ các thiết bị chuyển tiếp, như thiết bị chuyển mạch và bộ định tuyến, và tập trung hóa các chức năng này trên bộ điều khiển SDN. Trong các thực thể mạng truyền thống, như thiết bị chuyển mạch và bộ định tuyến, mặt phẳng điều khiển được kết hợp chặt chẽ trong mỗi thực thể, cùng với mặt phẳng dữ liệu, do đó gây khó khăn trong việc quản lý, sửa chữa. Khi muốn thay thế, sửa chữa thì buộc phải thực hiện trên từng thiết bị, gây tốn kém về mặt thời gian, chi phí. So với mạng truyền thống thì mạng điều khiển bằng phần mềm có nhiều lợi ích hơn. Lợi ích của việc tập trung mặt phẳng điều khiển trong một thực thể bao gồm đơn giản hóa việc quản lý mạng và giới thiệu khả năng lập trình trong mạng. Nó cũng giúp quản trị viên mạng dễ dàng nâng cấp các dịch vụ do mạng cung cấp từ nguồn tập trung, so với mạng truyền thống nơi mỗi thiết bị phải được cấu hình thủ công.



Hình 1: Mạng SDN và mạng truyền thống

Cấu trúc của mạng SDN gồm 3 lớp: Lớp ứng dụng, lớp điều khiển, lớp hạ tầng nằm trên ba mặt phẳng với các chức năng khác nhau:



Hình 2: Cấu trúc của mạng SDN

- Mặt phẳng quản lý: điều khiển, giám sát việc chuyển tiếp các bản tin
- Mặt phẳng điều khiển: xác định đường đi cho bản tin (định tuyến)
- Mặt phẳng dữ liệu: xử lý dữ liệu người dùng, chuyển tiếp các bản tin, thu nhận xử lý bản tin.

Trong mạng SDN, bộ điều khiển được coi như là trái tim của hệ thống mạng. Nó -chịu trách nhiệm quyết định hành động nào sẽ được thực hiện trên các gói, cài đặt các quy tắc trong các phần tử chuyển tiếp, ví dụ: chuyển mạch. Các quy tắc này được gọi là flow rules và mỗi phần tử chuyển tiếp duy trì các quy tắc này trong một bảng được gọi là flow table. Flow table này ra lệnh cho hoạt động của một thiết bị chuyển tiếp. Bộ điều khiển SDN giao tiếp với các thiết bị chuyển tiếp trên giao diện hướng nam và giao thức giao tiếp được sử dụng được gọi là giao thức OpenFlow. Giao thức OpenFlow được chuẩn hóa bởi Open Net working Foundation (ONF) và lợi ích chính của nó là nó cho phép khả

năng tương tác giữa các thiết bị trong môi trường mạng đa nhà cung cấp. Mặt khác, các ứng dụng mạng khác nhau chạy trên giao diện hướng bắc của bộ điều khiển.

Ngoài ra, mạng SDN còn gồm thành phần trừu tượng hóa mạng (network abstraction) giúp ta có thể mô tả mạng thông qua các câu lệnh, các chức năng... và thành phần “global network view” giúp bộ điều khiển có thể nhận biết được toàn bộ, có cái nhìn tổng thể về hệ thống mạng giúp cho việc quản lý, điều khiển trở lên dễ dàng hơn.

2.2.2: Pox controller

POX là phiên bản mới hơn, dựa trên Python của NOX (hoặc NOX trong Python). Ý tưởng đằng sau sự phát triển của nó là đưa NOX trở lại nguồn gốc C++ của nó và phát triển một nền tảng dựa trên Python riêng biệt (Python 2.7). Nó có API SDN cấp cao bao gồm biểu đồ cấu trúc liên kết có thể truy vấn và hỗ trợ ảo hóa.

Những ưu điểm của POX:

- POX có giao diện Pythonic OpenFlow
- POX có các thành phần mẫu có thể tái sử dụng để lựa chọn đường dẫn, khám phá cấu trúc liên kết
- POX chạy ở bất cứ đâu và có thể đi kèm với thời gian chạy PyPy không cần cài đặt để triển khai dễ dàng
- POX đặc biệt nhắm mục tiêu Linux, Mac OS và Windows
- POX hỗ trợ các công cụ GUI và trực quan hóa giống như NOX
- POX hoạt động tốt so với các ứng dụng NOX được viết bằng Python

Cả NOX và POX hiện đang giao tiếp với các thiết bị chuyển mạch OpenFlow v1.0 và bao gồm hỗ trợ đặc biệt cho Open vSwitch.

Không có GUI chính thức cho POX, mặc dù các dự án của bên thứ ba, chẳng hạn như POXDesk1, tồn tại. Đặc biệt, POXDesk cung cấp chức năng cơ bản, chẳng hạn như trực quan hóa các bảng luồng, các sự kiện được ghi lại và cấu trúc liên kết mạng. Giao tiếp giữa POXDesk và lõi của POX sử dụng API REpresentational State Transfer (REST) có sẵn với bộ điều khiển

Một trong những mục đích chính của việc sử dụng POX là để phát triển các ứng dụng điều khiển OpenFlow - nghĩa là nơi POX hoạt động như một bộ điều khiển cho bộ chuyển mạch OpenFlow (hoặc, theo thuật ngữ thích hợp hơn, đường dẫn dữ liệu OpenFlow).

Vì POX thường được sử dụng với OpenFlow, nên có một cơ chế tải nhu cầu đặc biệt, thường sẽ phát hiện khi đang cố gắng sử dụng OpenFlow và tải lên các thành phần liên quan đến OpenFlow với các giá trị mặc định.

Một phần chính của API POX OpenFlow là đối tượng "nexus" OpenFlow. Thông thường, có một đối tượng duy nhất như vậy được đăng ký là `core.openflow` như một phần của quá trình tải nhu cầu.

Thành phần POX thực sự giao tiếp với các thiết bị chuyên mạch OpenFlow là `openflow.of_01` (01 đề cập đến thực tế là thành phần này nói giao thức dây OpenFlow 0x01). Một lần nữa, tính năng demand-loading thường sẽ khiến thành phần này được khởi tạo với các giá trị mặc định (nghe trên cổng 6633). Tuy nhiên, có thể gọi nó tự động thay vào đó để thay đổi các tùy chọn hoặc nếu muốn chạy nó nhiều lần (ví dụ: để nghe trên TCP và SSL đơn giản hoặc trên nhiều cổng).

2.2.3: Routing in SDN

Định tuyến trong SDN (Software Defined Networking) là quá trình chuyển tiếp các gói dữ liệu giữa các mạng hoặc các đoạn mạng bằng cách sử dụng một đường dẫn được xác định trước. Trong mạng truyền thống, các quyết định định tuyến được thực hiện bởi các bộ định tuyến riêng lẻ sử dụng các giao thức định tuyến như OSPF, BGP hoặc RIP. Tuy nhiên, trong SDN, các quyết định định tuyến được thực hiện Controller.

Controller thu thập thông tin về topology mạng từ các switch, chẳng hạn như trạng thái liên kết, và sử dụng các thuật toán để tính toán đường dẫn tối ưu cho các gói dữ liệu. Các thuật toán định tuyến có thể là các thuật toán đường dẫn ngắn nhất truyền thống như Dijkstra hoặc Bellman-Ford hoặc các thuật toán nâng cao hơn dựa trên học máy và trí tuệ nhân tạo.

Các quyết định định tuyến được thực hiện bởi bộ điều khiển SDN có thể linh hoạt và thích ứng hơn so với mạng truyền thống, vì bộ điều khiển có thể phản ứng trong thời gian thực với các thay đổi trong topology mạng, tải lưu lượng và các thông số khác. Ngoài ra, SDN cho phép kiểm soát định tuyến được tinh vi hơn, chẳng hạn như định tuyến dựa trên ứng dụng hoặc chính sách.

Nhìn chung, SDN cung cấp sự linh hoạt và kiểm soát lớn hơn về định tuyến mạng so với các phương pháp mạng truyền thống. Nó cho phép quản trị mạng điều chỉnh động các chính sách định tuyến mạng phản ứng với các điều kiện mạng thay đổi và các mô hình lưu lượng.

2.3: Bài toán phát hiện đường dẫn bị hỏng, và tìm đường đi ngắn nhất của bản tin trong mạng SDN

2.3.1: Giới thiệu bài toán

Lỗi liên kết gây mất gói, dịch vụ không khả dụng, định tuyến không ổn định, suy giảm chất lượng dịch vụ. Đi đường các luồng dữ liệu bị gián đoạn từ liên kết thất bại sang một đường dẫn thay thế. Phục hồi phân tán làm cho sự chậm trễ trong việc tìm kiếm sự thất bại. Thiết kế tập trung của SDN làm giảm độ trễ trong việc phát hiện lỗi trong đó bộ điều khiển ngay lập tức sửa đổi mục nhập dòng chảy của mỗi switch. Khi phát hiện lỗi, các bộ định tuyến riêng lẻ, cập nhật các mục đường dẫn chuyển tiếp bị ảnh hưởng bởi lỗi và thông báo các thay đổi cập nhật cho các nút lân cận thông qua giao thức spanning tree. Trong bài này, đường dẫn dự thừa được coi là đường dẫn thay thế khi liên kết giữa 2 máy chủ xuống đường dẫn dự thừa được chọn để tiếp cận máy chủ. Điều này cung cấp đường dẫn thay thế cho liên kết không thành công.

Và khi ta có nhiều hơn 2 đường dẫn dự phòng thì hiệu suất mạng sẽ bị giảm, Độ trễ khi truyền các gói tin sẽ bị tăng, tắc nghẽn mạng dẫn đến mất mát cái gói tin. Vì vậy ta cần một thuật toán để controller có thể tìm được một đường đi ngắn nhất và truyền bản tin qua. Thuật toán mà bài báo cáo đưa ra là thuật toán Dijkstra.

2.3.2: Cơ sở lý thuyết

Giao thức spanning tree: Lựa chọn đường dẫn thay thế có thể được xem thông qua giao thức spanning tree. Giao thức spanning tree thường được sử dụng để ngăn chặn các vòng lặp trong mạng. Đường dẫn dự phòng được coi là một liên kết dự phòng thông qua đó các vòng lặp được loại bỏ

Thuật toán dijkstra: là một trong những thuật toán cổ điển để giải quyết bài toán tìm đường đi ngắn nhất từ một điểm cho trước tới tất cả các điểm còn lại trong đồ thị có trọng số.

Ý tưởng cơ bản của thuật toán như sau:

- Bước 1: Từ đỉnh gốc, khởi tạo khoảng cách tới chính nó là 00, khởi tạo khoảng cách nhỏ nhất ban đầu tới các đỉnh khác là $+\infty$. Ta được danh sách các khoảng cách tới các đỉnh.
- Bước 2: Chọn đỉnh a có khoảng cách nhỏ nhất trong danh sách này và ghi nhận. Các lần sau sẽ không xét tới đỉnh này nữa.
- Bước 3: Lần lượt xét các đỉnh kề b của đỉnh a. Nếu *khoảng cách từ đỉnh gốc tới đỉnh b* nhỏ hơn khoảng cách hiện tại đang được ghi nhận thì cập nhật giá trị và đỉnh kề a vào khoảng cách hiện tại của b.

- Bước 4: Sau khi xét tất cả đỉnh kề b của đỉnh a. Lúc này ta được danh sách khoảng cách tới các điểm đã được cập nhật. Quay lại Bước 2 với danh sách này. Thuật toán kết thúc khi chọn được khoảng cách nhỏ nhất từ tất cả các điểm.

Vì pox là một controller có hỗ trợ python nên trong quá trình mô phỏng sẽ không cần thực hiện từng bước như trên mà sẽ sử dụng một hàm có sẵn trên thư viện networkx của python, đó là “shortest_path”:

shortest_path(G, source=None, target=None, weight=None, method='dijkstra')

Trong đó:

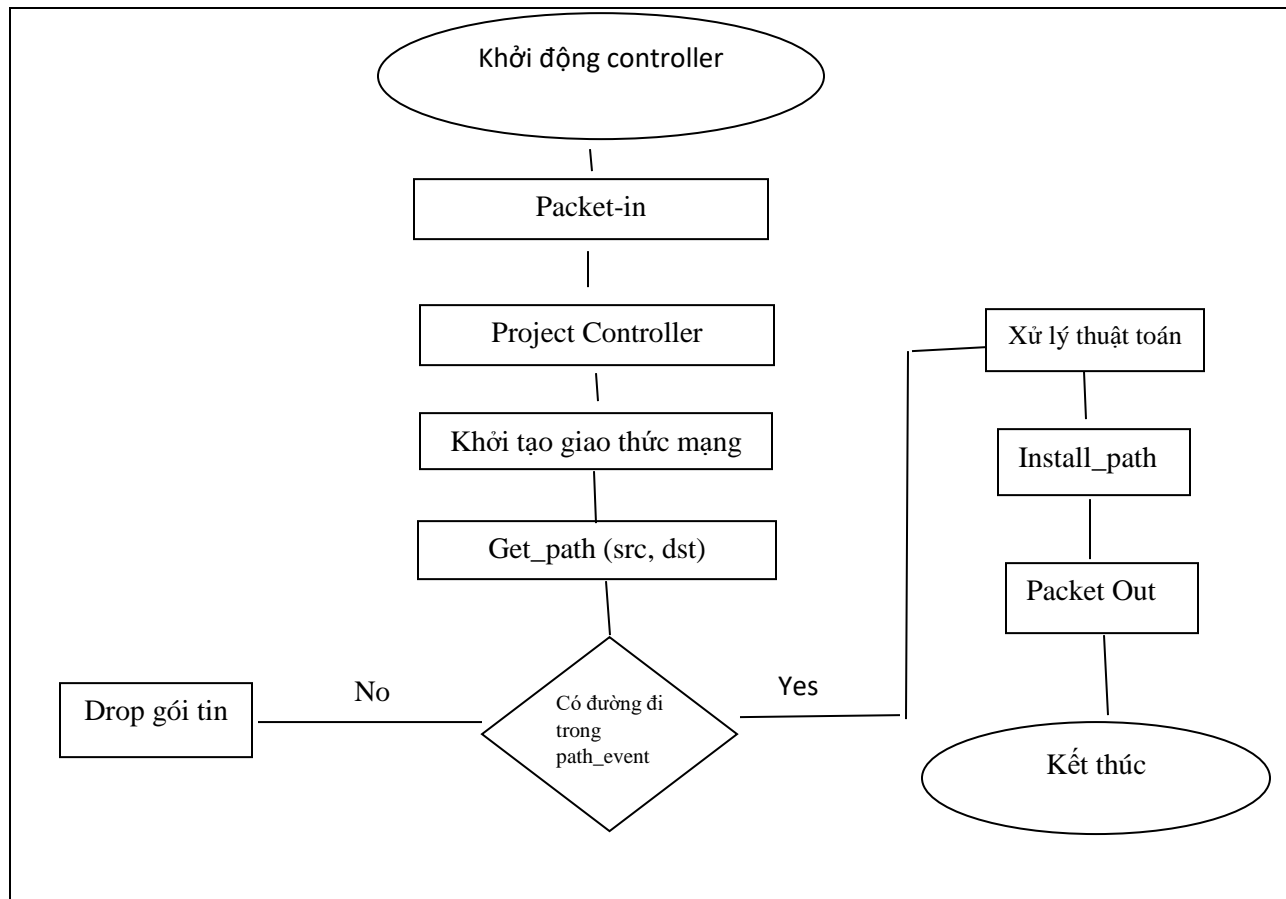
G: giao thức mạng cần triển khai

Source: điểm xuất phát (bên gửi)

Target: bên nhận

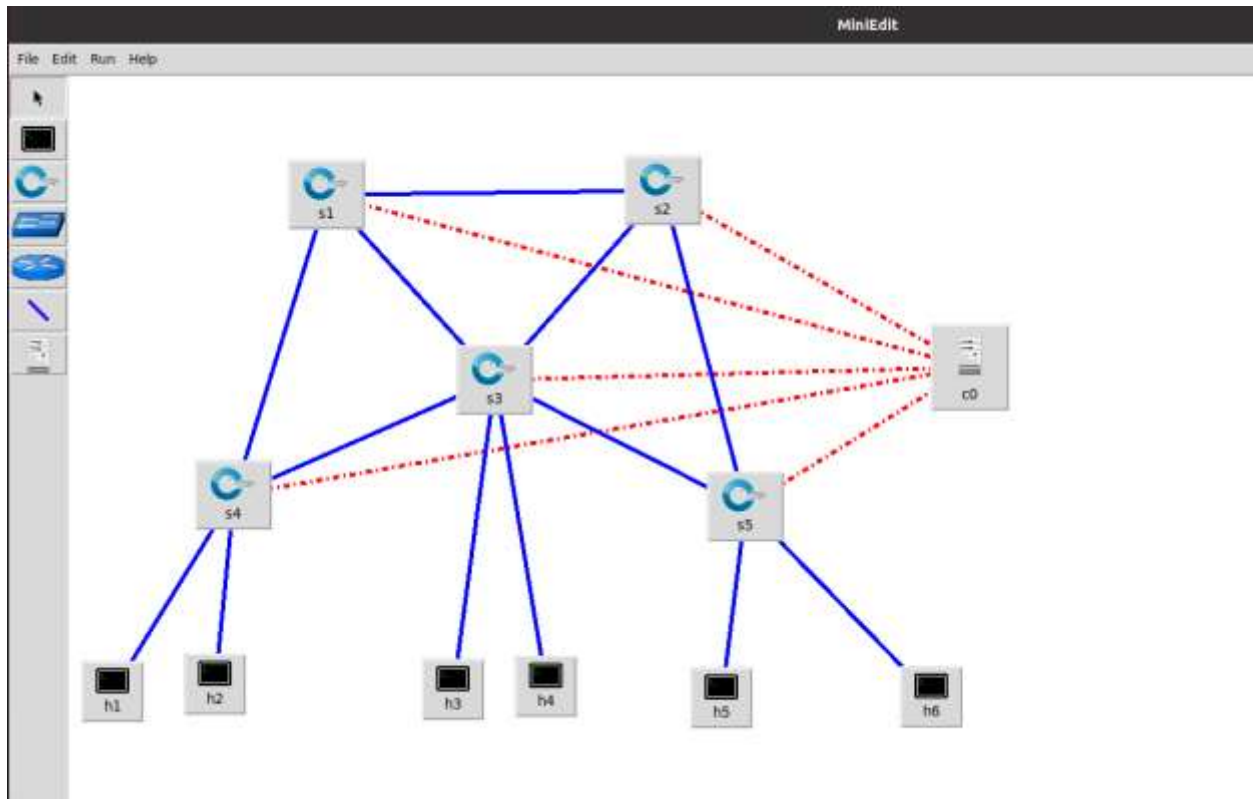
2.3.3: Mô hình, thuật toán, mô phỏng

Khi thực hiện bài toán định tuyến trên SDN, do các chức năng định tuyến đã được tập trung vào controller nên ta không cần thực hiện trên từng thiết bị mà chỉ cần thực hiện trong controller. Khi đó, ta sẽ cài các thuật toán xử lý vào trong controller cho project và chạy cùng với controller ban đầu, sau đó đánh giá kết quả. Mô hình của bài toán được đề xuất trong hình sau:



Hình 3: Sơ đồ xử lý bài toán

Thử nghiệm được thực hiện trên máy chủ ubuntu 20.04 nơi cài đặt công cụ ảo hóa mininet. Để triển khai cấu trúc liên kết, trong terminal chạy miniedit.py, một chương trình python bao gồm thanh công cụ ở một bên. Thanh công cụ bao gồm hosts, switches, Controller, netlinks, v.v. Trong terminal khác, mininet được thực thi. Topology được tạo trong miniedit được hiển thị trong hình 3. Controller bên phải được cấu hình làm bộ điều khiển từ xa và trong menu con tùy chọn của phần chỉnh sửa, hãy đảm bảo chọn hộp CLI bắt đầu. Thao tác này sẽ kích hoạt các chức năng trong cửa sổ bảng điều khiển mininet. Địa chỉ IP của máy chủ h1, h2, h3, h4, h5 và h6 được cấu hình lần lượt là 10.0.0.1, 10.0.0.2, 10.0.0.3, 10.0.0.4, 10.0.0.5 và 10.0.0.6.



Hình 4: Giao thức được tạo bởi Miniedit

Chức năng của Pox controller có thể được thực hiện với các chương trình python được gọi đơn giản là các thành phần của pox. Mở một terminal mới và chạy bộ điều khiển pox cùng với các thành phần của nó như trong hình 6. Để xem các sự kiện mạng, cần thực thi bộ điều khiển ở chế độ gỡ lỗi.

Thành phần `forwarding.l2_learning1` (được phát triển từ `forwarding.l2_learning`), `forwarding.l2_learning` là một module được sử dụng để xử lý các gói tin Ethernet và học cách chuyển tiếp chúng trong mạng. Khi một gói tin Ethernet được nhận bởi switch, nó sẽ được chuyển đến bộ điều khiển SDN để xử lý và xác định cách chuyển tiếp gói tin. Module `forwarding.l2_learning` sẽ học các địa chỉ MAC của các thiết bị mạng được kết nối với switch và xây dựng bảng chuyển tiếp (`forwarding table`) để quản lý việc chuyển tiếp gói tin. Khi có gói tin mới đến, `forwarding.l2_learning` sẽ kiểm tra địa chỉ MAC nguồn và đích của gói tin để xác định cách chuyển tiếp. Nếu địa chỉ MAC đích đã được học trước đó, bảng chuyển tiếp sẽ chỉ định switch nào để chuyển tiếp gói tin đến. Nếu địa chỉ MAC đích chưa được học trước đó, module sẽ gửi yêu cầu đến bộ điều khiển SDN để xác định cách chuyển tiếp. Module `forwarding.l2_learning1` được phát triển từ module ban đầu, Đầu tiên, thuật toán sẽ kiểm tra địa chỉ MAC đích của gói tin, nếu không biết địa chỉ đó thì gói tin sẽ được flood ra tất cả các cổng của switch (`self._flood_packet(event)`) để đảm bảo gói tin đến được đích. Nếu địa chỉ MAC đích đã biết, thuật toán sẽ tính toán đường đi ngắn nhất giữa các switch trong mạng bằng thuật toán Dijkstra và cài đặt luật

chuyển tiếp (flow entry) trên switch để chuyển tiếp gói tin đến cổng thích hợp. Cuối cùng, code gửi gói tin đến cổng thích hợp (self.send_packet(event, out_port)) để chuyển tiếp gói tin đến switch tiếp theo trên đường đi ngắn nhất.

```
# Compute the shortest path between switches using Dijkstra algorithm
src_switch = self.connection.dpid
dst_switch = self.topology[self.macToPort[dst]][ 'dpid' ]

if src_switch not in self.path:
    self.path[src_switch] = nx.shortest_path(self.topology, src_switch, dst_switch)
    log.info("Shortest path from switch %s to switch %s: %s", dpid_to_str(src_switch), dpid_to_str(dst_switch), self.path[src_switch])
out_port = self.macToPort[dst]
if in_port == out_port:
    # Drop the packet if it comes from and goes to the same port
    return
# Install a flow entry on the switch to forward the packet to the appropriate port
self._install_flow_rule(packet, in_port, out_port)
# Forward the packet to the appropriate port
self._send_packet(event, out_port)
```

Hình 5: Dijitra Algorithm

Thành phần openflow.Discovery có tác dụng tìm kiếm các switch OpenFlow được kết nối với mạng SDN bằng cách gửi các yêu cầu gửi gói tin hello đến các địa chỉ MAC của switch. Khi switch nhận được gói tin hello, nó sẽ trả lời lại bằng một gói tin hello_reply kèm theo thông tin về nó (như ID của switch, số phiên bản OpenFlow được hỗ trợ và các tính năng của switch). Đối với mỗi switch được phát hiện, openflow.Discovery sẽ gửi thông tin của switch đó đến các đối tác (partner) trong mạng SDN thông qua OpenFlow Controller. Nó sẽ tiếp tục tìm kiếm các switch khác và thông tin của chúng sẽ được gửi đến tất cả các đối tác trong mạng. Quá trình này sẽ giúp tạo ra một bản đồ toàn bộ mạng SDN và cung cấp cho Controller thông tin về topology của mạng. Controller sẽ sử dụng thông tin này để lập kế hoạch cho việc chuyển tiếp gói tin giữa các switch trong mạng và giải quyết các vấn đề khác liên quan đến chuyển mạch trong mạng SDN.

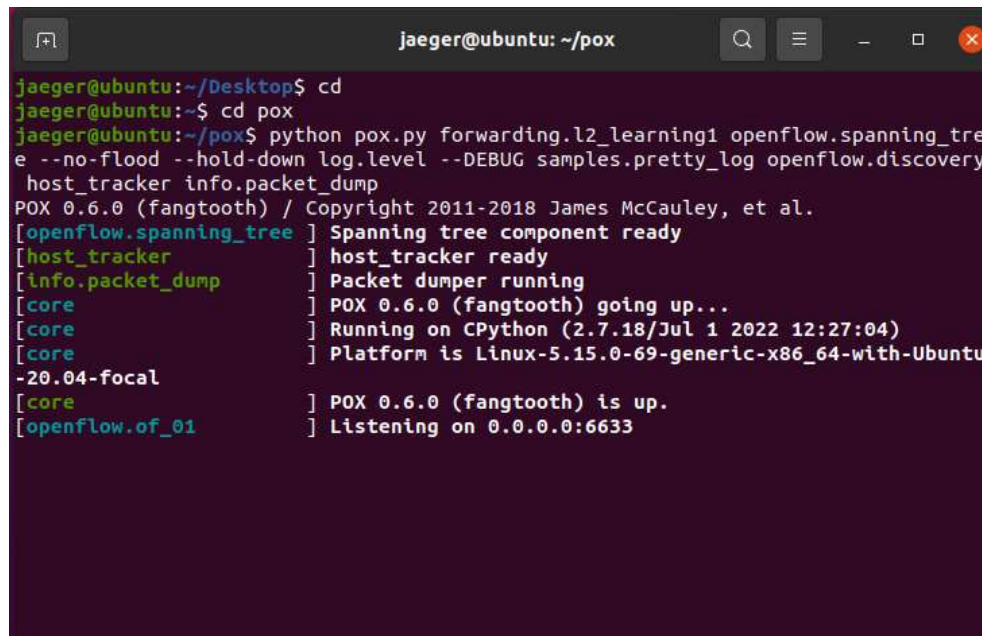
Thành phần openflow.spanning_tree --no-flood --hold-down là một thành phần có chức năng giúp cải thiện hiệu suất và độ ổn định của mạng bằng cách xác định các kết nối không cần thiết giữa các switch và loại bỏ chúng. Cụ thể, các tùy chọn "--no-flood" và "--hold-down" được sử dụng để ngăn chặn các gói tin broadcast và multicast khỏi bị phát lại nhiều lần trên các kết nối không cần thiết giữa các switch, giúp giảm tải cho mạng và tránh các vấn đề về độ trễ và xung đột trong mạng SDN. "--no-flood" có nghĩa là không truyền tải các gói tin broadcast hoặc multicast qua các cổng không cần thiết trong mạng, trong khi "--hold-down" cho phép các cổng mới được bật lại sau một khoảng thời gian giữa các lần truyền gói tin broadcast hoặc multicast, giúp tránh các vấn đề xảy ra khi các cổng mới được bật một cách đột ngột. Nhờ vào openflow.spanning_tree, mạng SDN có thể tránh được các vấn đề liên quan đến loop, giúp tăng hiệu suất và độ ổn định của mạng.

Thành phần log.level – DEBUG có chức năng giúp hiển thị thông tin chi tiết về quá trình hoạt động của controller, bao gồm các thông tin về gói tin nhận được và các tác vụ xử lý gói tin. Khi sử dụng log.level -- DEBUG, POX sẽ ghi lại tất cả các thông tin chi tiết về

hoạt động của controller trong quá trình chạy, cho phép người dùng có thể xem xét các lỗi xảy ra và tìm hiểu chi tiết về cách hoạt động của hệ thống. Tuy nhiên, việc sử dụng `log.level -- DEBUG` có thể làm giảm hiệu suất của hệ thống và tạo ra nhiều thông tin không cần thiết, do đó nên chỉ sử dụng khi cần thiết để tìm ra các vấn đề hoạt động của hệ thống.

“pretty_log” là một thành phần có chức năng giúp hiển thị các thông tin trên console dễ đọc và dễ hiểu hơn. Khi kích hoạt pretty_log, các thông tin sẽ được định dạng và đánh dấu màu cho từng thành phần khác nhau, giúp người dùng dễ dàng nhận biết và phân tích.

Thành phần “host_tracker”, y như tên gọi thì nó có chức năng theo dõi các host trong cấu trúc mạng



```
jaeger@ubuntu: ~/pox
jaeger@ubuntu:~/Desktop$ cd
jaeger@ubuntu:~$ cd pox
jaeger@ubuntu:~/pox$ python pox.py forwarding.l2_learning1 openflow.spanning_tree --no-flood --hold-down log.level --DEBUG samples.pretty_log openflow.discovery host_tracker info.packet_dump
POX 0.6.0 (fangtooth) / Copyright 2011-2018 James McCauley, et al.
[openflow.spanning_tree] Spanning tree component ready
[host_tracker] host_tracker ready
[info.packet_dump] Packet dumper running
[core] POX 0.6.0 (fangtooth) going up...
[core] Running on CPython (2.7.18/Jul 1 2022 12:27:04)
[core] Platform is Linux-5.15.0-69-generic-x86_64-with-Ubuntu
-20.04-focal
[core] POX 0.6.0 (fangtooth) is up.
[openflow.of_01] Listening on 0.0.0.0:6633
```

Hình 6: Chạy Pox controller cùng với những thành phần

Tại bảng điều khiển của bộ điều khiển pox và các sự kiện luồng mở đang được bộ điều khiển pox xử lý được hiển thị, được hiển thị dưới dạng thông báo nhật ký pox. Bộ điều khiển pox SDN kết nối với các Switches và thiết lập spanning tree kết hợp cùng với thuật toán Dijkstra. Với việc nhấp vào tùy chọn Run trong miniedit, toàn bộ kiến trúc sẽ trở nên rõ ràng. Output có thể được xem trong cửa sổ bảng điều khiển pox.

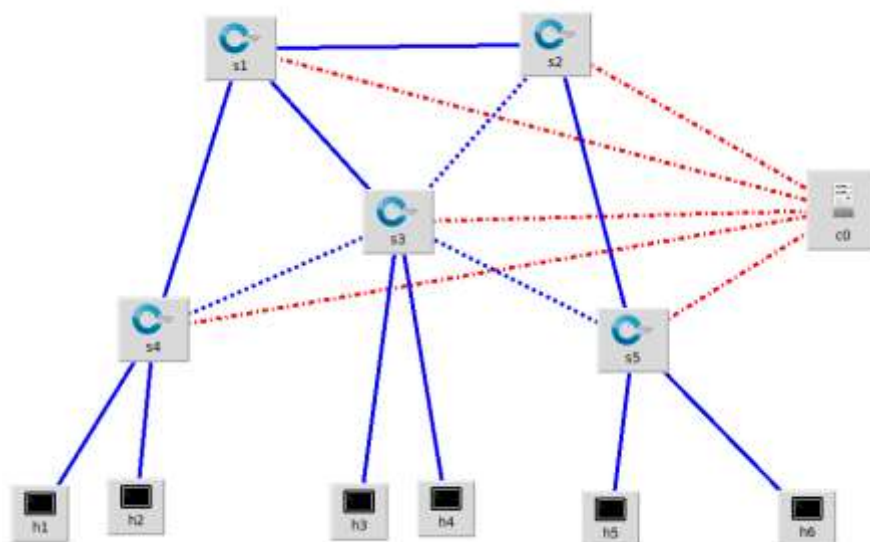
2.3.4: Kết quả, đánh giá

Trong terminal của mininet, xterm h1, sau đó ping host 6 bằng cách nhập ping 10.0.0.6 như trong hình 6, đây là địa chỉ IP tương ứng của host h6. Kết quả của lệnh ping được hiển thị trong hình. Bây giờ, các liên kết giữa các công tắc được thực hiện xuống bằng cách nhấp chuột phải vào liên kết và chọn tùy chọn liên kết xuống (đường chấm chấm

trong cấu trúc liên kết biểu thị liên kết xuống và đường liền nét biểu thị liên kết lên được hiển thị trong hình 8). Người ta quan sát thấy các gói openflow_Discovery trong cửa sổ bảng điều khiển pox đã hiển thị nhưng lệnh ping trong thiết bị đầu cuối máy chủ vẫn còn hoạt động.

```
"Node: h1"
root@ubuntu:/home/jaeger/mininet/examples# ping 10.0.0.6
PING 10.0.0.6 (10.0.0.6) 56(84) bytes of data:
64 bytes from 10.0.0.6: icmp_seq=1 ttl=64 time=50.3 ms
64 bytes from 10.0.0.6: icmp_seq=2 ttl=64 time=1.03 ms
64 bytes from 10.0.0.6: icmp_seq=3 ttl=64 time=0.128 ms
64 bytes from 10.0.0.6: icmp_seq=4 ttl=64 time=0.118 ms
64 bytes from 10.0.0.6: icmp_seq=5 ttl=64 time=0.057 ms
64 bytes from 10.0.0.6: icmp_seq=6 ttl=64 time=0.113 ms
64 bytes from 10.0.0.6: icmp_seq=7 ttl=64 time=0.043 ms
64 bytes from 10.0.0.6: icmp_seq=8 ttl=64 time=0.077 ms
64 bytes from 10.0.0.6: icmp_seq=9 ttl=64 time=0.043 ms
64 bytes from 10.0.0.6: icmp_seq=10 ttl=64 time=0.036 ms
64 bytes from 10.0.0.6: icmp_seq=11 ttl=64 time=0.040 ms
64 bytes from 10.0.0.6: icmp_seq=12 ttl=64 time=0.048 ms
64 bytes from 10.0.0.6: icmp_seq=13 ttl=64 time=0.123 ms
64 bytes from 10.0.0.6: icmp_seq=14 ttl=64 time=0.042 ms
64 bytes from 10.0.0.6: icmp_seq=15 ttl=64 time=0.051 ms
64 bytes from 10.0.0.6: icmp_seq=16 ttl=64 time=0.047 ms
```

Hình 7: Ping từ host 1 đến host 6



Hình 8: Nét đứt màu xanh đại diện cho những link bị down


```

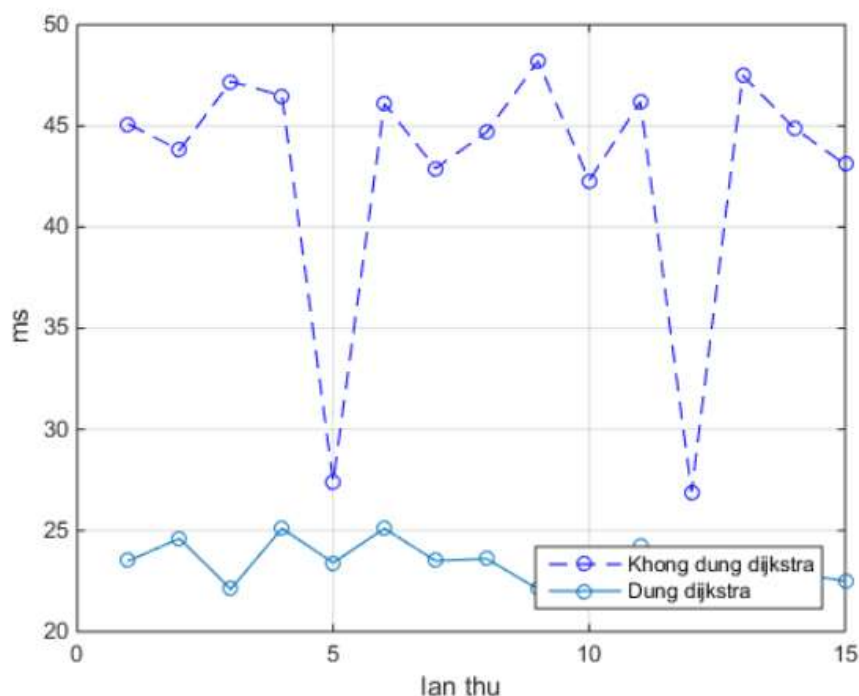
[dump:00-00-00-00-00-04] [ethernet][ipv4][icmp][echo][56 bytes]
[forwarding.l2_learning] installing flow for 2e:8f:8b:fa:1d:67.1 -> b2:39:aa:02:b9:89.3
[openflow.discovery] link timeout: 00-00-00-00-00-03.3 -> 00-00-00-00-00-04.2
[dump:00-00-00-00-00-04] [ethernet][arp]
[forwarding.l2_learning] installing flow for b2:39:aa:02:b9:89.3 -> 2e:8f:8b:fa:1d:67.1
[dump:00-00-00-00-00-01] [ethernet][arp]

```

Hình 9: Nhật ký của openflow.Discovery và forwarding.l2._learning

Ngay cả sau khi link down, host vẫn có thể truyền được các gói tin thông qua các link dự phòng. Điều này là do thành phần cây bao trùm có đường dẫn dự phòng để đến đích. Do đó, hosts đang tìm một cách khác để đến đích thông qua thành phần openflow.Discovery và thành phần Forwarding.l2_learning có nhật ký được hiển thị trong terminal như trong hình 8. Cứ sau 10 giây (theo mặc định), các luồng trong công tắc sẽ được cập nhật. Vì đường dẫn được giữ như một bản sao lưu hoạt động như một đường dẫn thay thế để đến đích. SDN đóng một vai trò quan trọng với tính linh hoạt, tính sẵn sàng và khả năng phục hồi nhanh chóng sau sự cố liên kết.

Sau khi thay module forwarding_learning bằng forwarding_learning1. Dưới đây là so sánh thời gian truyền package khi sử dụng và không sử dụng thuật toán Dijitra (hình 10).



Hình 10: Đồ thị so sánh giữa thời gian truyền package khi sử dụng và không sử dụng thuật toán Dijitra

Sau khi áp dụng thuật toán Dijkstra vào module forwarding.l2_learning, ta thấy thời gian truyền của package được giảm đáng kể. Sau khi thử nhiều lần, ta có thể thấy thuật toán tìm đường đi ngắn nhất Dijkstra được áp dụng vào controller đã có tác dụng. Từ đó có thể thấy thuật toán này có thể áp dụng vào SDN để thời gian truyền tin được nhanh hơn. Dù sau khi có kết nối các switch thì thời gian truyền tin sẽ được cải thiện, trở lên rất là ngắn kể cả khi áp dụng hay không áp dụng thuật toán tìm đường đi ngắn nhất. nhưng khi áp dụng thuật toán Dijkstra vào thì ta cũng có thể coi như một nâng cấp cho mô hình SDN ban đầu.

2.4: Bài toán truyền tin khi có host không đáng tin

2.4.1: Giới thiệu bài toán

Dù trong hệ thống mạng nào thì luôn có những lỗ hổng bảo mật, và mạng SDN cũng không ngoại lệ. Việc truyền và nhận các bản tin đến từ các máy chủ không đáng tin trong mạng SDN vẫn tồn tại một số nguy cơ bị tấn công hệ thống. Những máy chủ không đáng tin này có thể là những thiết bị được cài vào với mục đích dùng để ăn cắp thông tin khách hàng và đưa ra bên ngoài mạng, hoặc có thể gửi các bản tin một cách liên tục nhằm ảnh hưởng tới controller, làm cho nó quá tải và gây ra sự cố về hoạt động mạng. (DDOS attack). Kẻ tấn công có thể giả mạo địa chỉ MAC hoặc địa chỉ IP của một thiết bị trong mạng để gửi các gói tin, làm cho bộ điều khiển SDN định tuyến sai hoặc cho phép kẻ tấn công tiếp cận các tài nguyên mạng trái phép. Trong một số trường hợp, kẻ tấn công có thể tấn công các thiết bị trong mạng SDN từ xa thông qua các lỗ hổng bảo mật hoặc các phần mềm độc hại, và lấy được quyền điều khiển các thiết bị trong mạng. Để giảm thiểu các nguy cơ này, các nhà quản trị mạng có thể sử dụng các giải pháp như đưa hệ thống vào chế độ bảo vệ, giới hạn số lượng kết nối tối đa một thiết bị có thể tạo ra, chặn các yêu cầu trùng lặp, hay sử dụng các dịch vụ bảo mật DDOS. Sau khi phát hiện các máy chủ giả mạo (không đáng tin cậy) bằng các phương pháp kể trên, bài báo cáo đề xuất hệ thống tường lửa ảo và bộ định tuyến để điều khiển và trao đổi lưu lượng giữa các máy chủ, bao gồm các quy tắc liên quan đến việc các máy chủ nào có thể giao tiếp với nhau cũng như các loại gói tin mà chúng được phép truyền qua mạng.

2.4.2: Cơ sở lý thuyết

Bài toán nhằm đến mục tiêu triển khai một Firewall trên các ‘switch’ trong một mạng SDN để kiểm soát và điều phối lưu lượng mạng. Firewall có thể được triển khai bằng cách sử dụng giao thức OpenFlow để cài đặt các quy tắc trên ‘switch’, qua đó có thể kiểm soát lưu lượng mạng trên các cổng (ports) của switch.

Các quy tắc được thiết kế để xử lý các gói tin mạng khi chúng vào hoặc ra khỏi ‘switch’. Các quy tắc có thể được thiết lập để định tuyến gói tin đến các cổng (port) tương ứng với địa chỉ đích, chặn các gói tin dựa trên địa chỉ nguồn và đích, hoặc phân loại các gói tin và áp dụng các quy tắc khác nhau tùy thuộc vào loại gói tin.

Ngoài ra, bài toán còn triển khai một số quy tắc đơn giản để xử lý các gói tin ICMP, ARP và TCP trên các ‘switch’. Khi một gói tin tới ‘switch’, các luật được áp dụng để xử lý gói tin đó và xác định cổng đích phù hợp để chuyển tiếp gói tin tới switch tiếp theo hoặc host đích.

Các quy tắc được triển khai có thể được tùy chỉnh để đáp ứng nhu cầu của mỗi hệ thống mạng cụ thể. Bằng cách sử dụng Firewall, ta có thể kiểm soát và quản lý lưu lượng mạng trên mạng SDN, giảm thiểu các rủi ro bảo mật và nâng cao hiệu suất mạng. Firewall có thể được coi là phương tiện hiệu quả để bảo vệ hệ thống hoặc mạng lưới hệ thống cục bộ khỏi các mối đe dọa bảo mật dựa trên mạng trong khi đồng thời truy cập vào hệ thống bên ngoài thông qua mạng diện rộng và Internet

src ip	dst ip	protocol	action
any ipv4	any ipv4	icmp	accept
any	any	arp	accept
any ipv4	any ipv4	-	drop

Bảng 1: Xử lý gói tin qua SDN

ICMP (Internet Control Message Protocol) là một giao thức báo cáo lỗi, thông báo cho sender biết việc gửi data đi có vấn đề, cũng giống như bộ định tuyến sử dụng để tạo thông báo lỗi đến địa chỉ IP nguồn khi các sự cố mạng ngăn chặn việc phân phối các IP packages. ICMP tạo và gửi thư đến địa chỉ IP nguồn, cho biết rằng một gateway vào Internet mà không thể truy cập được. Mọi thiết bị mạng IP đều có khả năng gửi, nhận hoặc xử lý tin nhắn ICMP. ICMP không phải là giao thức truyền tải gửi dữ liệu giữa các hệ thống. Đây là một giao thức lớp mạng được sử dụng để truyền các thông điệp điều khiển giữa máy chủ và bộ định tuyến. Để đảm bảo an ninh và an toàn của mạng, việc duy trì giao tiếp thành công giữa các thiết bị là điều cần thiết. Các thông điệp ICMP được truyền dưới dạng các datagrams, bao gồm một IP header đóng gói dữ liệu ICMP. CMP packets là IP packets với ICMP trong phần dữ liệu IP. Các tin nhắn ICMP cũng chứa toàn bộ tiêu đề IP từ tin nhắn gốc, vì vậy end system sẽ biết được packet nào đang có vấn đề.

ARP (viết tắt của cụm từ Address Resolution Protocol) là giao thức mạng được dùng để tìm ra địa chỉ phần cứng (địa chỉ MAC) của thiết bị từ một địa chỉ IP nguồn. Thiết bị gửi sử dụng ARP để có thể dịch địa chỉ IP sang địa chỉ MAC. Thiết bị sẽ gửi một request ARP đã chứa địa chỉ IP của thiết bị nhận. Tất cả thiết bị trên đoạn local network sẽ nhìn thấy thông điệp này. Tuy nhiên, chỉ thiết bị có địa chỉ IP chứa trong request mới có thể phản hồi lại với thông điệp mà chứa địa chỉ MAC của nó. Thiết bị gửi khi đó sẽ có đầy đủ các thông tin để gửi packet tới thiết bị nhận.

2.4.3: Mô hình, thuật toán, mô phỏng

Mô hình:

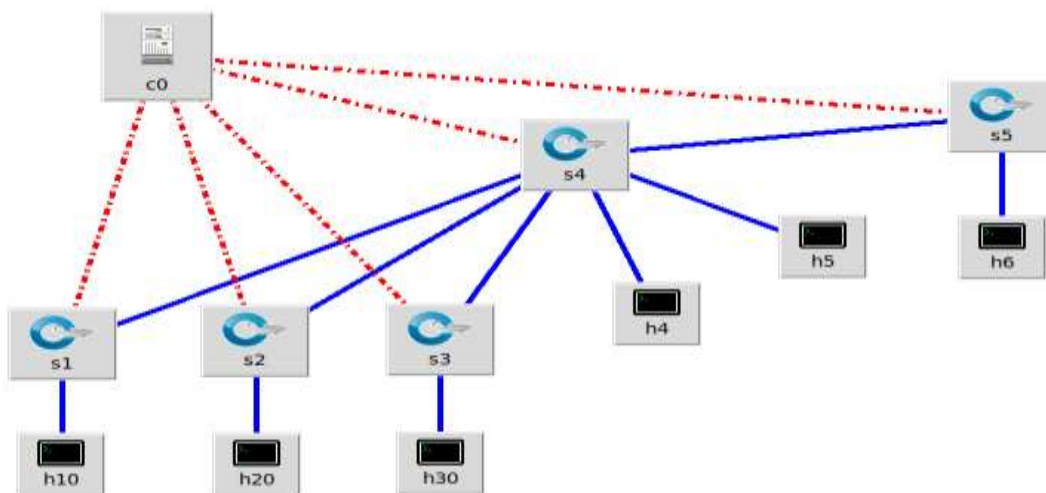
Máy chủ h1 (cổng 0) kết nối với ‘switch’ s1 tại tầng thứ nhất (trên cổng 1), s1 kết nối với ‘switch’ lõi s4 (cổng 11 để kết nối với cổng 7 trên switch lõi). Máy chủ h2 và máy chủ h3 kết nối với switch lõi tương tự như cách máy chủ h1 kết nối với switch lõi, trên các cổng khác nhau. Các máy chủ đáng tin cậy và không đáng tin cậy (tương ứng là h4 và h5) được liên kết trực tiếp với ‘switch’ lõi, cho phép kết nối với các máy chủ khác (do các hạn chế của giao thức ICMP và TCP).

Máy chủ (h6) kết nối với bộ chuyển mạch trung tâm dữ liệu, bộ chuyển mạch này kết nối với bộ chuyển mạch lõi, do đó cho phép kết nối với phần còn lại của máy chủ.

Ý tưởng cơ bản của thuật toán như sau:

1. Tạo ra một bộ điều khiển để thực thi các quy tắc trong giao thức ICMP: máy chủ đáng tin cậy có thể giao tiếp với bất kỳ máy chủ nào khác, máy chủ không tin cậy chỉ có thể giao tiếp với máy chủ đáng tin cậy. Cụ thể, các máy chủ h10/h20/h30 và ‘server’ có thể giao tiếp với tất cả các máy chủ khác ngoại trừ máy chủ không tin cậy.
2. Bộ điều khiển còn thực hiện các quy tắc sau cho giao thức TCP: tất cả các máy chủ có thể trao đổi các gói tin TCP với nhau; ngoại lệ duy nhất là máy chủ không đáng tin cậy không thể giao tiếp với máy chủ. Đối với các gói tin ARP, không có giới hạn nào (sử dụng flooding).

Mô phỏng trên Mininet:



Hình 11: Giao thức xử lý bài toán untrusted host

2.4.4: Kết quả, đánh giá

Kết quả ping giữa các host

```
mininet> pingall
*** Ping: testing ping reachability
h4 -> h5 h6 h10 h20 h30
h5 -> h4 X X X X
h6 -> h4 X h10 h20 h30
h10 -> h4 X h6 h20 h30
h20 -> h4 X h6 h10 h30
h30 -> h4 X h6 h10 h20
*** Results: 26% dropped (22/30 received)
mininet> █
```

Có thể thấy các gói tin gửi đến và đi (tổng cộng 8) của máy chủ h5 đều bị “dropped”. Kết quả là 22/30 gói tin gửi thành công, tỉ lệ là 26% dropped.

```
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
src ip not untrusted
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
src ip not untrusted
dstip is h10
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
src ip not untrusted
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
src ip not untrusted
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
packet is ICMP :)
```

Tương ứng thông báo trả về trên bộ điều khiển.

Kiểm tra bảng lưu lượng

```
mininet> dctl dump-flows
*** s1 ***
cookie=000, duration=596.810s, table=0, n_packets=2, n_bytes=84, arp,vlan_tci=0x0000,d_l_src=00:00:00:00:04,d_l_dst=ff:ff:ff:ff:ff:ff,arp_spa=104.82.214.112,arp_tpa=156.134.2.12,arp_op=1 actions=FL000
cookie=000, duration=596.880s, table=0, n_packets=1, n_bytes=42, arp,vlan_tci=0x0000,d_l_src=00:00:00:00:05,d_l_dst=00:00:00:00:04,arp_spa=156.134.2.12,arp_tpa=104.82.214.112,arp_op=2 actions=FL000
cookie=000, duration=596.799s, table=0, n_packets=1, n_bytes=42, arp,vlan_tci=0x0000,d_l_src=00:00:00:00:04,d_l_dst=ff:ff:ff:ff:ff:ff,arp_spa=104.82.214.112,arp_tpa=10.0.4.10,arp_op=1 actions=FL000
cookie=000, duration=596.792s, table=0, n_packets=2, n_bytes=84, arp,vlan_tci=0x0000,d_l_src=00:00:00:00:06,d_l_dst=00:00:00:00:04,arp_spa=10.0.4.10,arp_tpa=104.82.214.112,arp_op=2 actions=FL000
cookie=000, duration=596.725s, table=0, n_packets=1, n_bytes=42, arp,vlan_tci=0x0000,d_l_src=00:00:00:00:04,d_l_dst=ff:ff:ff:ff:ff:ff,arp_spa=104.82.214.112,arp_tpa=10.0.1.10,arp_op=1 actions=FL000
cookie=000, duration=596.723s, table=0, n_packets=2, n_bytes=84, arp,vlan_tci=0x0000,d_l_src=00:00:00:00:01,d_l_dst=00:00:00:00:04,arp_spa=10.0.1.10,arp_tpa=104.82.214.112,arp_op=2 actions=FL000
cookie=000, duration=596.664s, table=0, n_packets=1, n_bytes=42, arp,vlan_tci=0x0000,d_l_src=00:00:00:00:04,d_l_dst=ff:ff:ff:ff:ff:ff,arp_spa=104.82.214.112,arp_tpa=10.0.2.20,arp_op=1 actions=FL000
cookie=000, duration=596.637s, table=0, n_packets=2, n_bytes=84, arp,vlan_tci=0x0000,d_l_src=00:00:00:00:02,d_l_dst=00:00:00:00:04,arp_spa=10.0.2.20,arp_tpa=104.82.214.112,arp_op=2 actions=FL000
```

```
cookie=000, duration=596.874s, table=0, n_packets=1, n_bytes=98, icmp,vlan_tci=0x0000,d_l_src=00:00:00:00:01,d_l_dst=00:00:00:00:04,nw_src=10.0.1.10,nw_dst=104.82.214.112,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth11"
cookie=000, duration=546.413s, table=0, n_packets=1, n_bytes=98, icmp,vlan_tci=0x0000,d_l_src=00:00:00:00:06,d_l_dst=00:00:00:00:01,nw_src=10.0.4.10,nw_dst=10.0.1.10,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth1"
cookie=000, duration=546.410s, table=0, n_packets=1, n_bytes=98, icmp,vlan_tci=0x0000,d_l_src=00:00:00:00:01,d_l_dst=00:00:00:00:06,nw_src=10.0.1.10,nw_dst=10.0.4.10,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth11"
cookie=000, duration=546.104s, table=0, n_packets=1, n_bytes=98, icmp,vlan_tci=0x0000,d_l_src=00:00:00:00:01,d_l_dst=00:00:00:00:04,nw_src=10.0.1.10,nw_dst=104.82.214.112,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth11"
cookie=000, duration=546.100s, table=0, n_packets=1, n_bytes=98, icmp,vlan_tci=0x0000,d_l_src=00:00:00:00:04,d_l_dst=00:00:00:00:01,nw_src=104.82.214.112,nw_dst=10.0.1.10,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth1"
cookie=000, duration=546.056s, table=0, n_packets=1, n_bytes=98, icmp,vlan_tci=0x0000,d_l_src=00:00:00:00:01,d_l_dst=00:00:00:00:05,nw_src=10.0.1.10,nw_dst=156.134.2.12,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth11"
cookie=000, duration=536.836s, table=0, n_packets=1, n_bytes=98, icmp,vlan_tci=0x0000,d_l_src=00:00:00:00:01,d_l_dst=00:00:00:00:06,nw_src=10.0.1.10,nw_dst=10.0.4.10,nw_tos=0,icmp_type=8,icmp_code=0 actions=output:"s1-eth11"
```

Hình 12: Bảng lưu lượng bài toán untrusted host

Từ kết quả mô phỏng, ta nhận thấy các gói tin đã không thể được đến các host không đáng tin, từ đó có thể nâng cao tính bảo mật của mạng SDN.

3: Đánh giá

Bài báo cáo đã xử lý thành công 2 bài toán tìm đường đi ngắn nhất và bài toán gửi gói tin khi xuất hiện các host không đáng tin. Với bài toán tìm đường đi ngắn nhất thì việc dùng thuật toán Dijkstra đã thành công làm giảm thời gian gửi và nhận gói tin trong mạng SDN; phát hiện được những đường dẫn bị sập. Với bài toán gửi gói tin khi xuất hiện các host không đáng tin thì đã thành công ngăn chặn việc gửi và nhận các bản tin từ các host không đáng tin, từ đó nâng cao tính bảo mật của mạng SDN

Tuy nhiên, các bài toán vẫn còn những hạn chế, yêu cầu nhanh chóng xử lý trong tương lai:

- Với bài toán tìm đường đi ngắn nhất: Dù đã tìm được đường đi ngắn nhất nhưng chưa thể xuất ra được các switch mà gói tin đi qua, từ đó chưa thể đưa ra được sự so sánh, sự khác biệt cụ thể về đường đi của bản tin so với sử dụng controller pox sẵn có.

- Với bài toán gửi gói tin khi xuất hiện các host đáng nghi thì vẫn chưa thể có một thuật toán để dự đoán chính xác các host đáng nghi mà chỉ dựa vào việc giả thiết khi xử lý bài toán.

4: Kết luận

Bài toán định tuyến là một bài toán lớn và quan trọng trong bất cứ mô hình mạng nào. Khi giải quyết thành công và tối ưu được bài toán này thì việc gửi gói tin sẽ trở lên nhanh hơn và các hệ thống mạng sẽ được đánh giá cao hơn.

Trong SDN, bài toán định tuyến được thực hiện tại bộ điều khiển và điều này giúp ta không mất nhiều thời gian, công sức khi thiết lập, sửa chữa. Dù trong bài báo cáo chỉ xử lý một phần rất nhỏ của bài toán định tuyến như tìm đường đi ngắn nhất, phát hiện đường dẫn bị sập và gửi tin khi xuất hiện các host không đáng tin nhưng điều đó cũng góp phần nâng cao hệ thống định tuyến của mạng SDN. Từ các kết quả, đánh giá ta có thể đưa ra kết luận rằng: Một hệ thống mạng SDN có thể tối ưu hóa các thông số khi gửi tin và nâng cao tính bảo mật khi ta áp dụng được các thuật toán trong controller. Từ đó ta đã có thể đánh giá được tầm quan trọng của controller trong mạng điều khiển mềm, khi muốn tối ưu hóa mạng SDN thì một trong những cách thuận tiện nhất là tác động trực tiếp vào controller.

Mặc dù các bài toán đã được chứng minh, mô phỏng rằng chúng có tác dụng tốt trong bộ điều khiển của SDN nhưng khi đi sâu vào giải quyết các bài toán thì nhận thấy trong đó vẫn còn tồn tại những nhược điểm và nhiều vấn đề làm cho các bài toán chưa thể tối ưu. Từ đó, trong tương lai, nhóm hi vọng có thể tiếp tục nghiên cứu, phát triển các thuật toán để có thể giải quyết bài toán một cách triệt để nhất.

5: Tài liệu tham khảo

- [1] Hailong Zhang; Jinyao Yan, "Performance of SDN Routing in Comparison with Legacy Routing Protocols," 29 October 2015.
- [2] Abeer A. Z. Ibrahim; Fazirulhisyam Hashim; Aduwati Sali; Nor K. Noordin, "A Multi-Objective Routing Mechanism for Energy Management Optimization in SDN Multi-Control Architecture," 7 February 2022.
- [3] V.Muthumanikandan & C. Valliyammai, "Link Failure Recovery Using Shortest Path Fast Rerouting Technique in SDN," *Link Failure Recovery Using Shortest Path Fast Rerouting Technique in SDN*, 29 June 2017.
- [4] Yi-Ren Chen & Wen-Guey Tzeng, "RL-Routing: An SDN Routing Algorithm Based

on Deep Reinforcement Learning," 19 August 2020.

- [5] Abbas Yazdinejad & Reza M. Parizi, "Cost optimization of secure routing with untrusted devices in software defined networking," 21 March 2020.
- [6] Syed Waleed, Muhammad Faizan, Maheen Iqbal, Muhammad Irfan Anis, "Demonstration of Single Link Failure Recovery using Bellman Ford and Dijkstra Algorithm in SDN," 2017.
- [7] Jue Chen, Jinbang Chen, "Link Failure Recovery in SDN: High Efficiency, Strong Scalability and Wide Applicability," 2018.