ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG
CHUYÊN NGÀNH: AN TOÀN THÔNG TIN

BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH

IPSDN: HỆ THỐNG PHÒNG CHỐNG XÂM NHẬP THÔNG MINH TRONG CÁC MẠNG ĐIỀU KHIỂN BẰNG PHẦN MỀM

GIẢNG VIÊN HƯỚNG DẪN:

ThS. Nguyễn Duy

NHÓM SINH VIÊN THỰC HIỆN:

Họ và tên: Lê Kim Tuấn
MSSV: 19520333
Họ và tên: Nguyễn Ngọc Diễm Quỳnh
MSSV: 19520242
Lớp: NT114.M21.ANTN

Tháng 07/2022

MỤC LỤC

MỤC LỤC HÌNH ẢNH

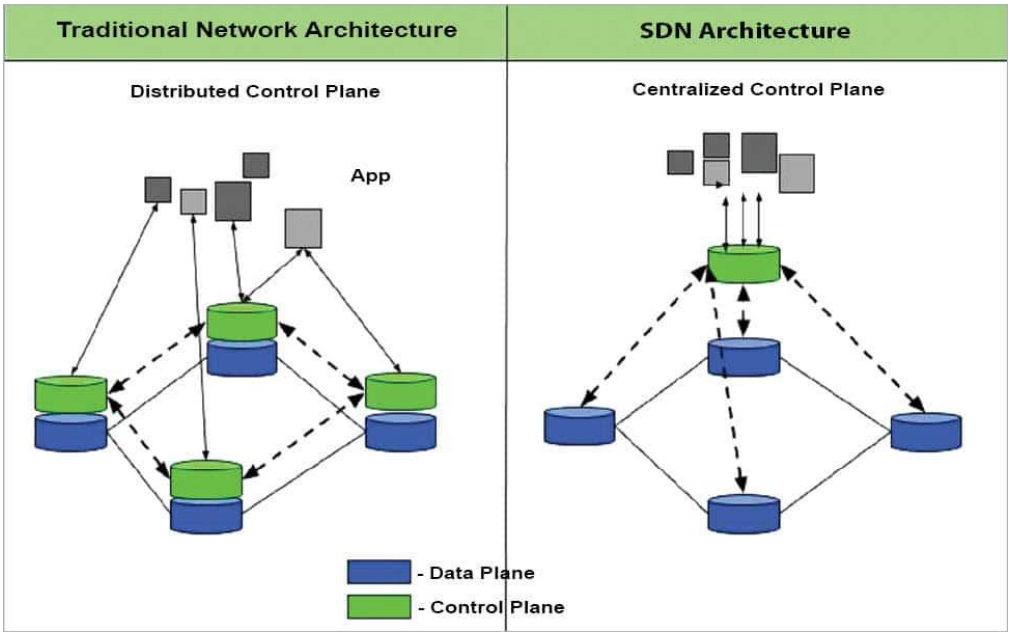
MỤC LỤC BẢNG

BÁO CÁO CHI TIẾT

I. Sơ lược về ngữ cảnh và nội dung đề tài

1. Ngữ cảnh:

SDN (Software-Defined Networking): Mạng điều khiển bằng phần mềm được sử dụng rộng rãi trong những năm gần đây nhờ những ưu điểm vượt trội so với mô hình mạng truyền thống như dễ triển khai, linh hoạt, hỗ trợ lập trình và có khả năng mở rộng. Trong SDN, hoạt động của mạng được điều khiển và quản lý thông qua các bộ điều khiển trung tâm (controller) giúp quản lý lưu lượng, cấu hình định tuyến, tích hợp các giải pháp bảo mật cho mạng và cung cấp tầm nhìn toàn cục về cơ sở hạ tầng đã triển khai.



Hình 1 Mạng truyền thống và mạng SDN

Khi nhu cầu sử dụng dữ liệu và lưu lượng mạng tiếp tục phát triển trong các mạng thực tế, rủi ro đối với bảo mật thông tin cũng tăng cao, ảnh hưởng đến nhiều cá nhân và tổ chức. Mạng SDN thường tồn tại lỗ hổng tại bộ điều khiển trung tâm. Một khi khai thác được bộ điều khiển, kẻ tấn công có thể chiếm quyền điều khiển toàn mạng.

Đồ án này đề xuất việc kết hợp Snort và Deep Learning vào SDN, để tạo thành hệ thống phòng chống xâm nhập thông minh trong các mạng điều khiển bằng phần mềm, được đặt tên là IPSDN.

2. Nội dung đề tài:

IPSDN sử dụng Mininet để mô phỏng kết nối giữa Open vSwitch và các host trong SDN. Controller được sử dụng là Ryu.

Trong hệ thống này Snort đóng vai trò là signature-based IDS giám sát lưu lượng đi qua Open vSwitch (OVS) và phát hiện các hành động độc hại dựa trên dấu hiệu

được định nghĩa trong các rule, sau đó gửi các cảnh báo tới controller thông qua Unix socket. Nhờ các cảnh báo này, controller sẽ thêm một flow entry trên OVS để chặn các traffic đến từ địa chỉ MAC của kẻ tấn công.

Ngoài ra, controller sử dụng Deep Learning để phát hiện các hành động bất thường dựa trên các flow của OVS (flow-based anomaly detection) nhằm cải thiện hạn chế của signature-based IPS.

IPSDN là sự kết hợp của Snort IDS, Deep Learning và sự can thiệp để sửa đổi các flow entry trên OVS của Ryu controller.

Để đánh giá hiệu quả của mô hình này, một mạng SDN đã được xây dựng trên hệ điều hành Ubuntu 18.04 LTS. Từ mô phỏng kết quả, IPSDN có khả năng phát hiện và ngăn chặn các cuộc tấn công một cách hiệu quả, nhanh chóng mà không tiêu tốn nhiều tài nguyên phần cứng (không quá 3% CPU usage).

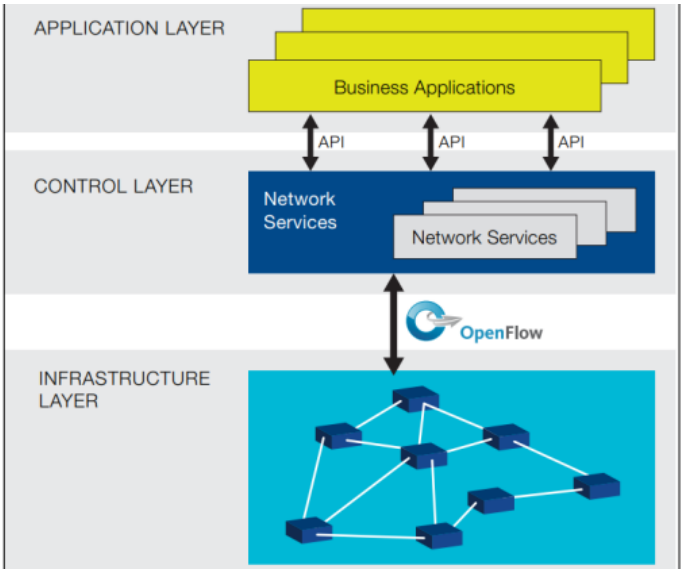
II. Nội dung chi tiết và mô hình triển khai đề tài:

1. Mạng điều khiển bằng phần mềm SDN:

Software-Defined Networks (SDN): Mạng điều khiển bằng phần mềm là một kiểu kiến trúc mạng mới tách biệt chức năng điều khiển (Control Plane) và chuyển tiếp dữ liệu (Forwarding Plane/Data Plane), cung cấp khả năng ảo hóa mạng và tự động hóa thông qua khả năng lập trình.

Trong SDN, quản trị viên có thể định hình lưu lượng mạng từ một console điều khiển tập trung mà không cần phải xử lý từng thiết bị chuyển mạch (switch) ở trong mạng. SDN controller sẽ định hướng cho các switch xử lý traffic đi qua chúng trong trường hợp switch không thể tự xử lý được.

Một kiến trúc SDN điển hình bao gồm 3 lớp: lớp ứng dụng, lớp điều khiển và lớp cơ sở hạ tầng.



Hình 2 Kiến trúc SDN

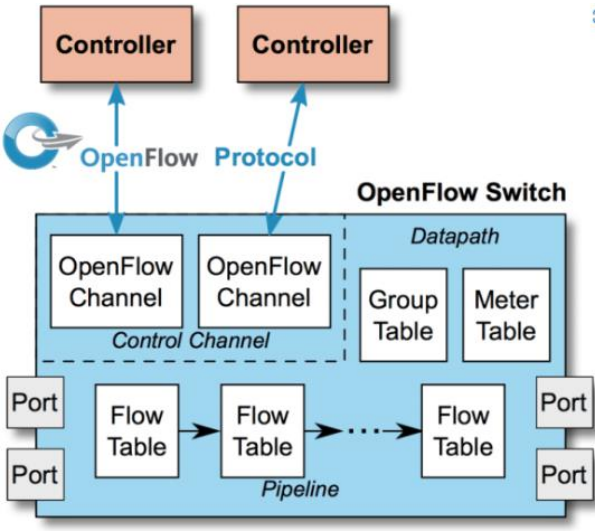
Lớp ứng dụng: Là các ứng dụng được triển khai trên mạng, được kết nối tới lớp điều khiển thông qua các API, cung cấp khả năng cho phép lớp ứng dụng lập trình lại (cấu hình lại) mạng (điều chỉnh các tham số trễ, băng thông, định tuyến, ...) thông qua lớp điều khiển.

Lớp điều khiển: tập trung các controller thực hiện việc điều khiển cấu hình mạng theo các yêu cầu từ lớp ứng dụng dựa trên khả năng của mạng. Các bộ điều khiển này có thể là các phần mềm được lập trình.

Lớp cơ sở hạ tầng: Là các thiết bị mạng thực tế (vật lý hay ảo hóa) thực hiện việc chuyển tiếp gói tin theo hướng dẫn của lớp điều khiển. Một thiết bị mạng có thể hoạt động theo hướng dẫn của nhiều bộ điều khiển khác nhau, điều này giúp tăng cường khả năng ảo hóa của mạng.

2. OpenFlow (version 1.3):

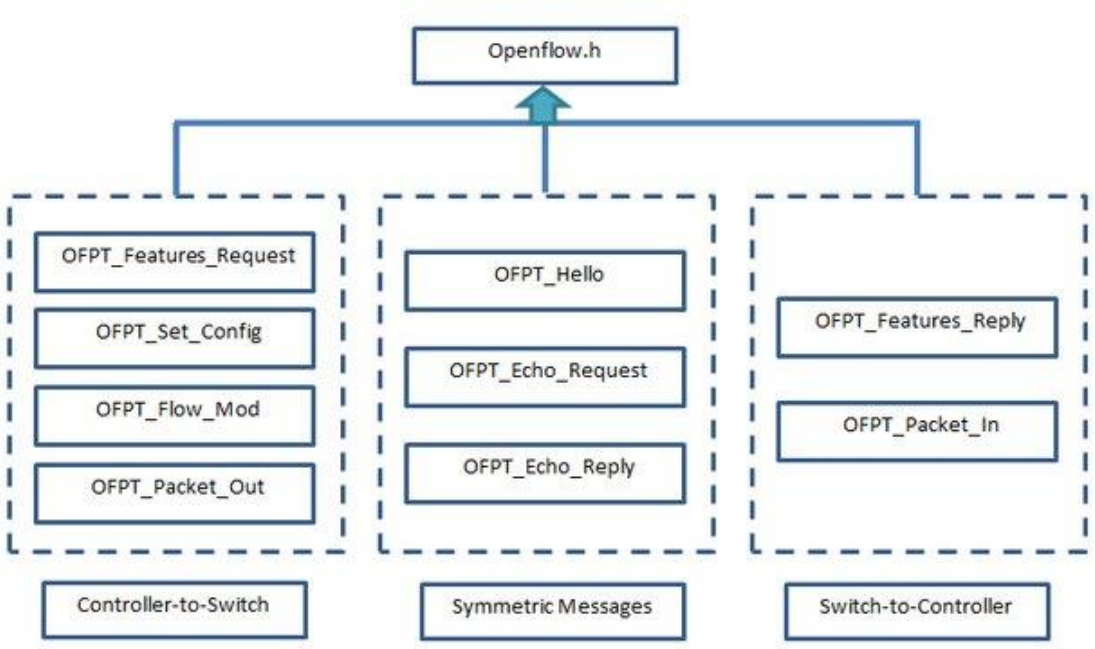
OpenFlow là tiêu chuẩn đầu tiên, cung cấp khả năng truyền thông giữa các giao diện của Control Plane và Forwarding Plane trong kiến trúc SDN. OpenFlow cho phép truy cập trực tiếp và điều khiển Forwarding Plane của các thiết bị mạng như switch, router, cả thiết bị vật lý và thiết bị ảo, do đó giúp di chuyển phần điều khiển mạng ra khỏi các thiết bị chuyển mạch thực tế tới phần mềm điều khiển trung tâm. Các quyết định về các luồng traffic sẽ được xử lý tập trung tại OpenFlow controller giúp đơn giản trong việc quản trị cấu hình trong toàn hệ thống. Một thiết bị OpenFlow bao gồm ít nhất 3 thành phần:



Hình 3 OpenFlow

- **OpenFlow Secure Channel**: kênh kết nối giữa các thiết bị như switch/router và bộ điều khiển, cho phép các lệnh và các gói tin được gửi giữa bộ điều khiển và thiết bị. Kênh này có thể được mã hóa bằng TLS hoặc trực tiếp bằng TCP, với tuỳ chọn mặc định là TLS.

- **OpenFlow Protocol**: giao thức tiêu chuẩn cho việc truyền thông giữa bộ điều khiển và thiết bị. OpenFlow protocol hỗ trợ 3 loại tin nhắn: controller-to-switch, asynchronous, và symmetric.
  - Tin nhắn **controller-to-switch:** được controller khởi tạo và dùng quản lý trực tiếp hoặc xem xét trạng thái của switch.
  - Tin nhắn **asynchronous:** được switch khởi tạo và dùng để cập nhật cho controller các sự kiện trong mạng và các thay đổi trên trạng thái của switch.
  - Tin nhắn **symmetric:** có thể được controller hoặc switch khởi tạo và được gửi không cần yêu cầu. Là tin nhắn Hello được trao đổi giữa controller và switch để khởi tạo kết nối.



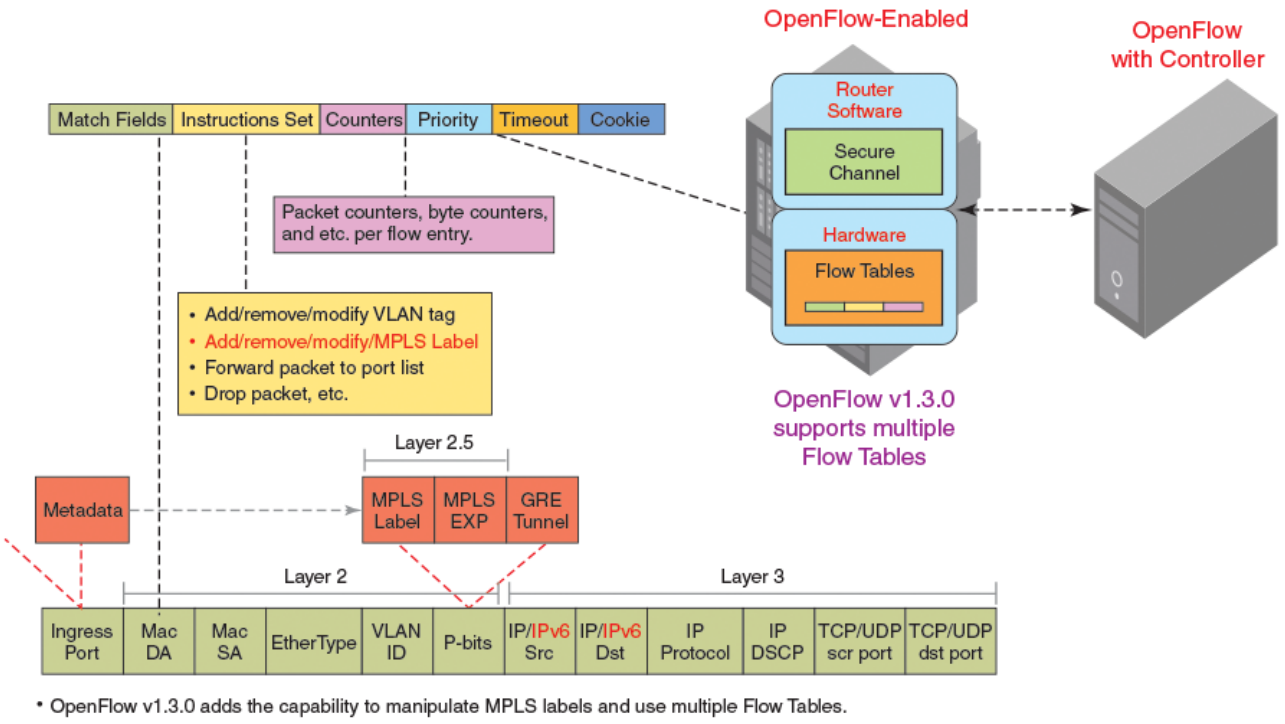
Hình 4 OpenFlow Messages

- **Flow table**: liên kết các hành động với các luồng (flows), giúp thiết bị xử lý các traffic trong mạng; mỗi mục (flow entry) trong flow table chứa một tập các trường phù hợp trong gói tin (match) và một hành động (như gửi ra port, sửa đổi trường hoặc hủy). Khi một OpenFlow switch nhận được một gói tin nó chưa bao giờ thấy trước đây mà không tìm được flow tương ứng trong flow table, nó sẽ gửi gói tin này đến bộ điều khiển. Controller sau đó đưa ra quyết định về cách xử lý gói tin này. Nó có thể hủy (drop) gói tin, hoặc thêm một flow entry hướng dẫn việc chuyển tiếp gói tin đến các port chỉ định.

Trong mỗi OpenFlow switch sẽ chứa một hoặc nhiều flow table. Trong mỗi flow table chứa một tập các flow entry. Các flow entry này được thêm, sửa đổi hoặc xóa bởi SDN controller. Mỗi flow entry sẽ bao gồm các thành phần sau:

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Bảng 1 Cấu trúc flow entry



Hình 5 Cấu trúc flow entry

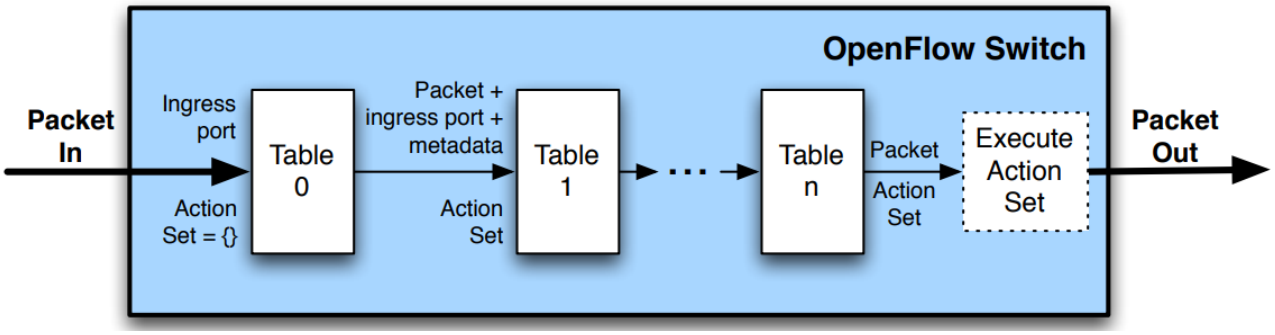
- Match Fields: Gồm các thông tin về header của gói tin và ingress port dùng để so khớp các gói tin. Khi một gói tin match với một flow entry thì gói tin đó sẽ được xử lý bằng các hành động đã được định nghĩa trong flow entry đó.
- Priority: Mức độ ưu tiên của flow entry đó. Mỗi gói tin có thể match với nhiều flow entry, khi đó gói tin sẽ được xử lý bởi flow entry có priority cao hơn.
- Counters: Cập nhật bộ đếm mỗi khi gói tin được xử lý bởi flow entry đó.
- Timeout: Bao gồm hard_timeout (0) - thời gian tối đa mà flow entry có thể tồn tại và idle_timeout (0) - Thời gian còn hiệu lực từ lúc flow entry được truy cập lần cuối.
- Instructions: Định nghĩa các hành động để xử lý gói tin.
- Cookie: Giá trị được chọn bởi controller được sử dụng để lọc số liệu thống kê của flow entry, sửa đổi flow entry hoặc xóa flow entry.
- Flags (0): có thể dùng kết hợp các giá trị dưới đây:

Giá trị	Giải thích
OFPFF_SEND_FLOW_REM	Gửi tin ngắn Flow Removed đến controller khi entry này bị xóa.
OFPFF_CHECK_OVERLAP	Khi command có giá trị OFPFC_ADD, kiểm tra các entry trùng nhau. Nếu có, Flow Mod không thành công và trả về 1 lỗi.
OFPFF_RESET_COUNTS	Đặt lại bộ đếm gói tin và bộ đếm byte của entry liên quan.
OFPFF_NO_PKT_COUNTS	Vô hiệu hóa bộ đếm gói tin của entry.
OFPFF_NO_BYT_COUNTS	Vô hiệu hóa bộ đếm byte của entry.

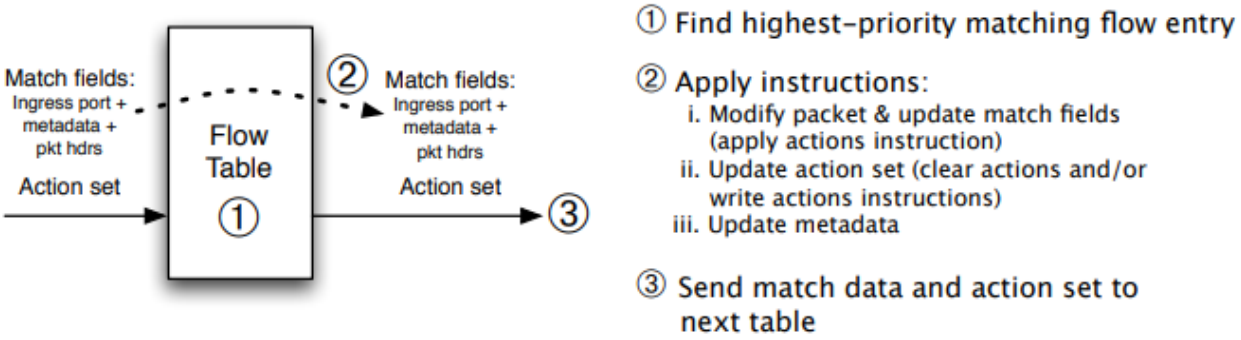
Bảng 2 Giá trị của flags

Cơ chế hoạt động của flow table trên OpenFlow switch:

- Ngoài các flow entry thông thường, mỗi flow table còn chứa một table-miss flow entry được sử dụng để xử lý các gói tin khi không có flow entry nào trong flow table khớp với gói tin đó. Table-miss flow entry có thể sẽ chuyển tiếp gói tin đó đến controller, gửi gói tin tới flow table tiếp theo hoặc loại bỏ gói tin.

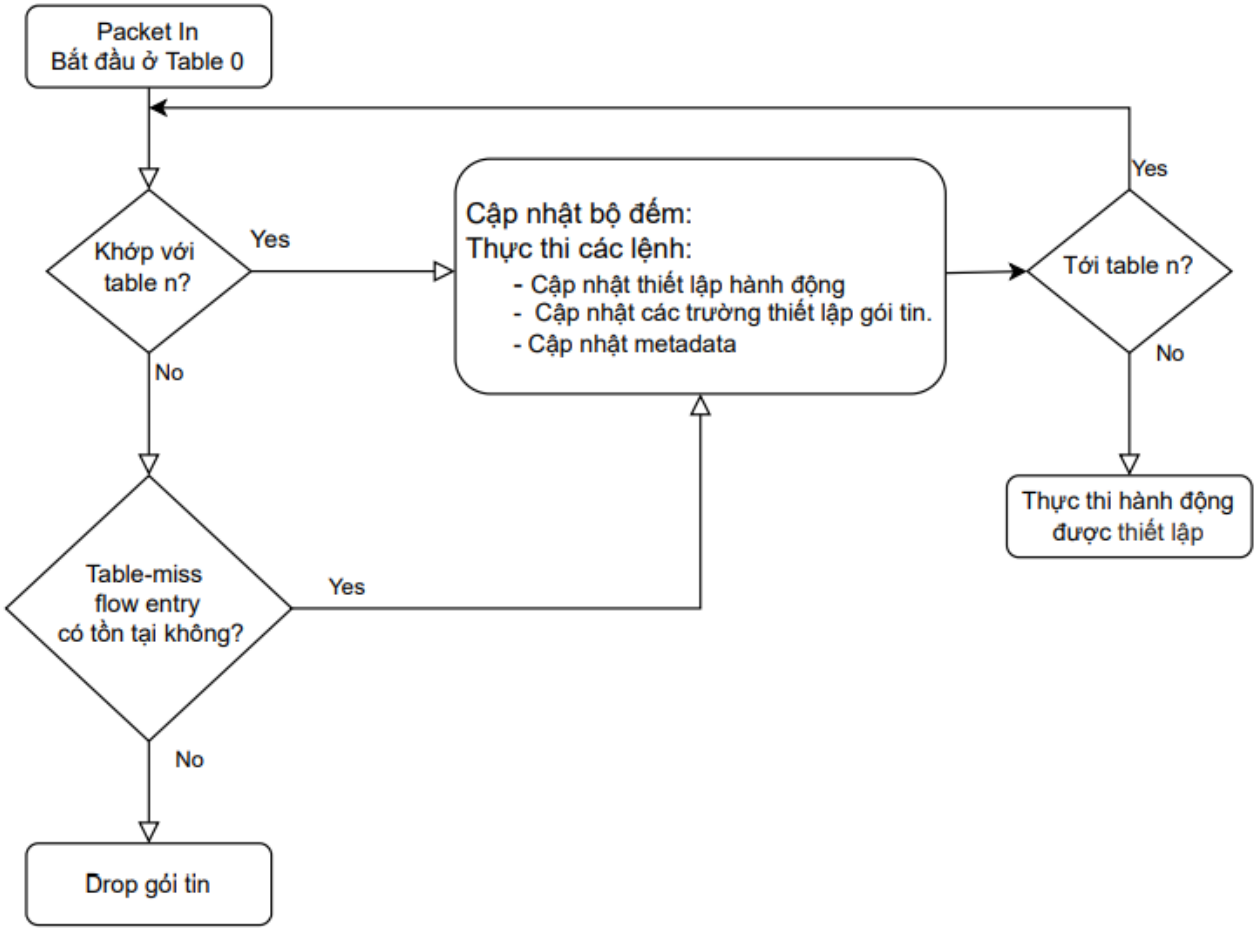


Hình 6 Quy trình pipeline xử lý gói tin khớp nhiều bảng



Hình 7 Chi tiết quy trình xử lý gói tin của mỗi bảng

- Với các OpenFlow switch chứa nhiều flow table, các flow table sẽ được đánh số bắt đầu từ table 0. Việc xử lý pipeline sẽ luôn được bắt đầu tại table 0. Các table khác có thể sẽ được sử dụng phụ thuộc vào đầu ra của match entry trong table đầu tiên. Việc chia ra nhiều flow table thay vì chỉ sử dụng một flow table sẽ tiết kiệm thời gian xử lý các gói tin hơn.
- Khi flow table xử lý một gói tin, nó sẽ so khớp với các flow entry để chọn ra một. Nếu một flow entry được tìm thấy, tập lệnh trong flow entry được thực thi. Những lệnh này có thể chuyển gói tin trực tiếp đến một flow table khác (Goto-Table), tại đó quá trình tương tự được lặp lại. flow entry chỉ có thể chuyển gói tin đến flow table có số thứ tự lớn hơn flow table của chính nó. Các flow entry của bảng cuối cùng không có lệnh Goto-Table. Nếu flow entry không chuyển gói tin đến flow table khác, quá trình xử lý pipeline dừng lại ở bảng này, gói tin được xử lý với tập các hành động và thường được chuyển tiếp.
- Nếu một gói tin không khớp với flow entry trong flow table thì đây là table miss. Hành vi trên table miss phụ thuộc vào cấu hình bảng. Các lệnh trong table-miss flow entry trong flow table chỉ định cách xử lý các gói không khớp một cách linh hoạt, bao gồm drop, chuyển sang một bảng khác hoặc gửi đến controller qua Control Channel bằng Packet-In.
- Trong một số trường hợp, khi một flow entry không xử lý được hoàn toàn gói tin và quá trình xử lý pipeline dừng lại, nếu không có table-miss thì gói tin bị loại bỏ.
- Trong một số trường hợp, flow table có thể chuyển hướng của một flow (các gói tin cùng loại) tới một group table để kích hoạt một hoặc nhiều hành động để xử lý flow. Meter table có thể được kích hoạt để tác động đến hiệu suất xử lý của một flow.
- Group table cho phép nhóm một số lượng port thành một output port thiết lập hỗ trợ multicasting, multipath, indirection và fast-failover.
- Meter table cho phép OpenFlow switch thực hiện các hành động QoS (Quality of Service). Meter table sẽ thực hiện tính toán tốc độ của gói tin và kích hoạt điều khiển tốc độ gói tin đó. Meter được gắn trực tiếp vào các flow entry, bất kỳ một flow entry nào cùng có thể chỉ định meter. Meter đo và kiểm soát tốc độ tổng hợp của tất cả các flow entry mà nó được gắn vào.

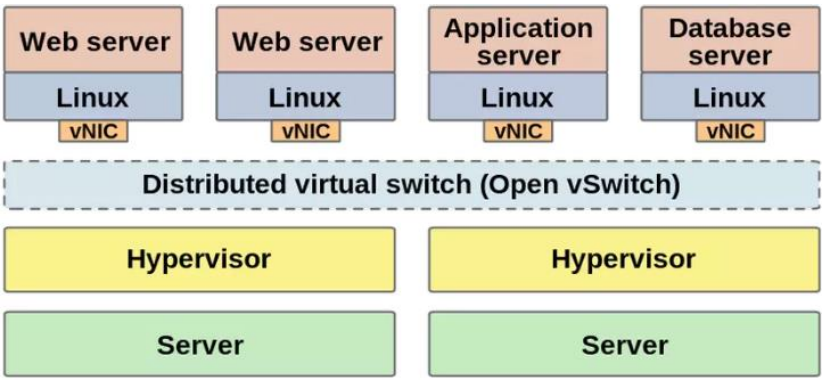


Hình 8 Lưu đồ thuật toán quy trình xử lý gói tin

Khi một gói tin từ ngoài mạng được gửi tới switch, quá trình xử lý gói tin được bắt đầu từ table 0. Gói tin sẽ được kiểm tra xem có một flow entry nào match với gói tin, thực hiện chuỗi hành động hoặc trỏ gói tin tới table thứ n nào không? Nếu không có chúng sẽ được tìm kiếm trong table-miss flow entry. Nếu tiếp tục không tìm thấy thì gói tin lập tức bị drop. Ngược lại nếu tồn tại một flow entry nào đó trong flow table match với gói tin hoặc tồn tại table-miss flow entry thì counter sẽ được update và thực thi các câu lệnh: Cập nhật thiết lập hành động, cập nhật các trường thiết lập gói tin, cập nhật metadata. Trong trường hợp gói tin được gửi đến table thứ n, tại đây nó sẽ tiếp tục được kiểm tra xem có cần phải gửi đến table nào khác không, nếu có thì gói tin tiếp tục được gửi đến table đó. Nếu không thì thực thi hành động đã được thiết lập.

3. Open vSwitch:

Open vSwitch (OVS) là một phần mềm chuyển mạch ảo đa lớp (multilayer software switch). Mục đích chính là hỗ trợ lớp chuyển mạch cho môi trường ảo hóa phần cứng, hỗ trợ nhiều giao thức và tiêu chuẩn được sử dụng trong hệ thống chuyển mạch thông thường. Open vSwitch hỗ trợ nhiều công nghệ ảo hóa dựa trên nền tảng Linux như Xen/XenServer, KVM, và VirtualBox.



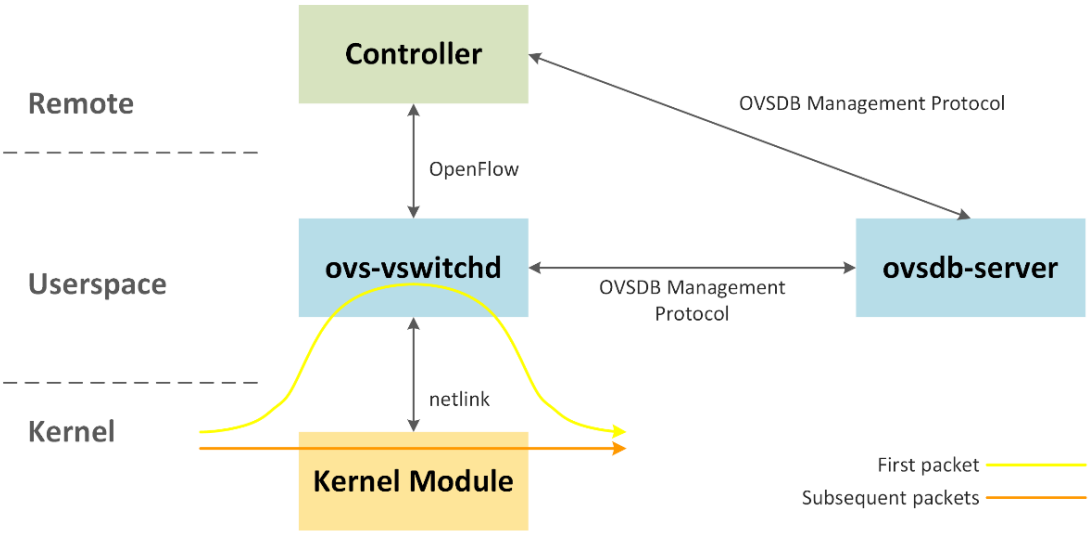
Hình 9 Vị trí Open vSwitch trong SDN

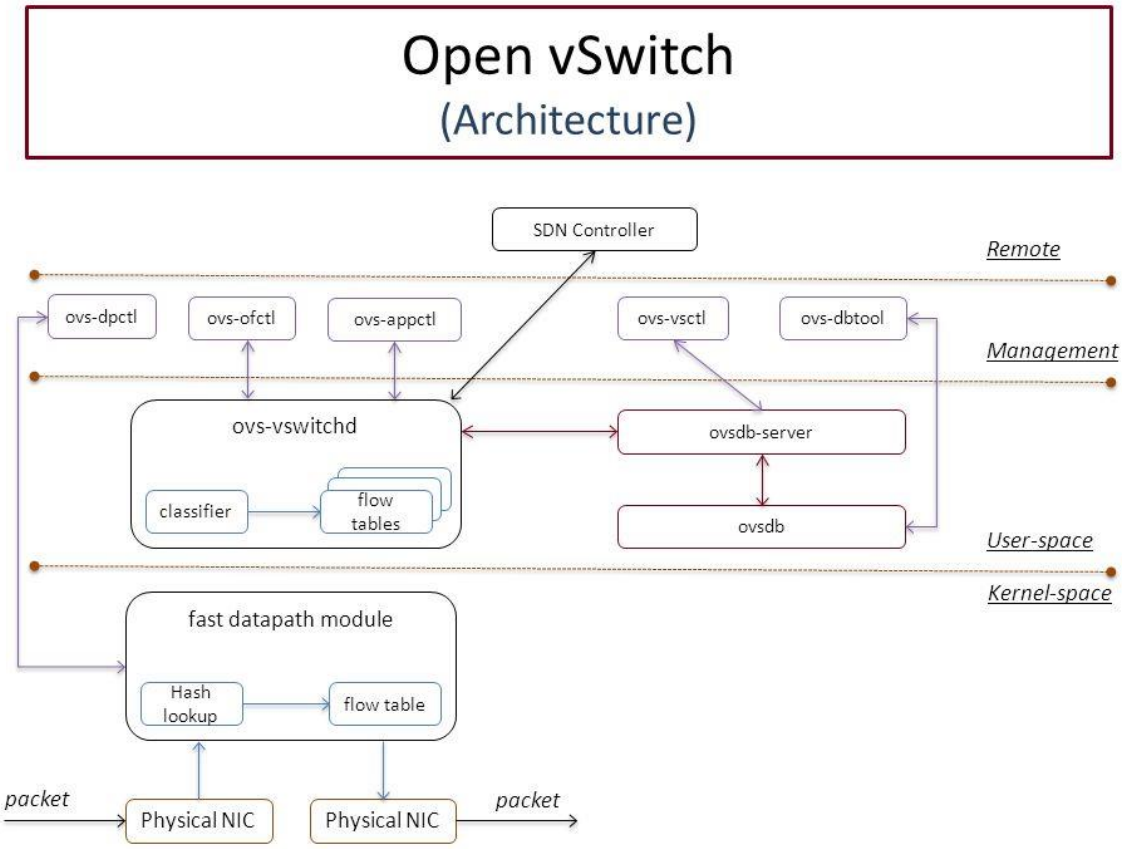
Open vSwitch hỗ trợ các tính năng:

- VLAN tagging & 802.1q trunking
- Giao thức Spanning Tree tiêu chuẩn (802.1D)
- LACP
- Ánh xạ port (SPAN/RSPAN)
- Giao thức Tunneling
- QoS

Các thành phần chính của Open vSwitch:

- ovs-vswitchd: thực hiện chuyển đổi các luồng chuyển mạch.
- ovsdb-server: là một hệ quản trị cơ sở dữ liệu nhỏ gọn, cho phép ovs-vswitchd thực hiện các truy vấn đến cấu hình.
- ovs-dpctl: công cụ để cấu hình các thiết bị chuyển mạch kernel module.
- ovs-vsctl: tiện ích để truy vấn và cập nhật cấu hình ovs-switchd.
- ovs-appctl: tiện ích gửi command để chạy Open vSwitch.





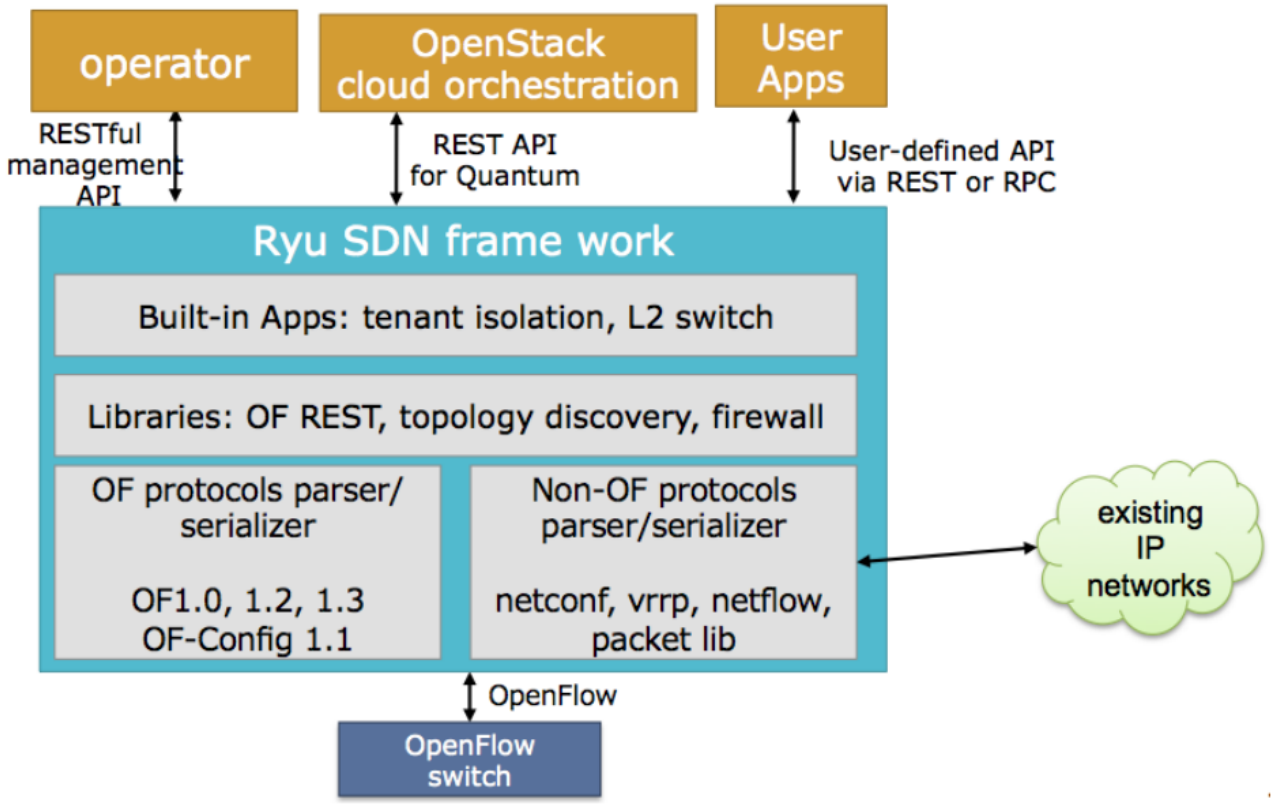
Hình 10 Các thành phần chính trong OVS

Open vSwitch có thể thực hiện các chức năng sau:

- Thay đổi địa chỉ của gói tin đã nhận hoặc gửi gói tin đến port chỉ định.
- Gửi gói tin đến controller (Packet-In).
- Gửi gói tin được chuyển tiếp từ controller đến port chỉ định (Packet-Out).

4. SDN controller (sử dụng OpenFlow 1.3):

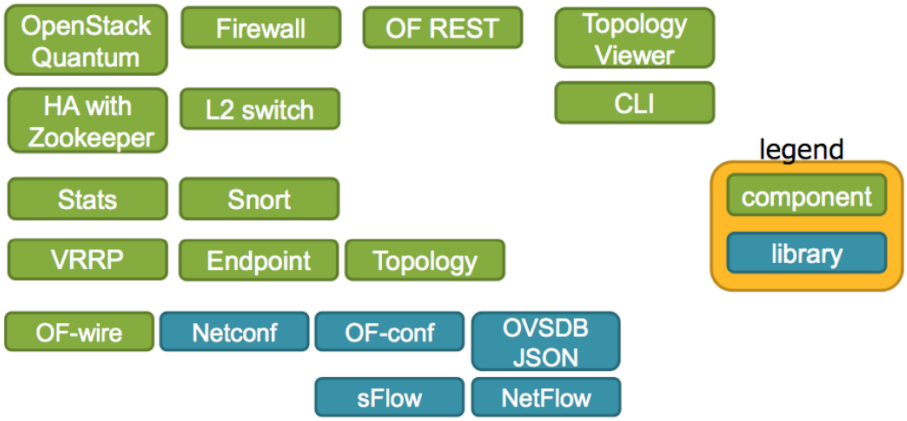
Trong SDN, controller là bộ não, đảm nhận việc truyền nhận thông tin với các switch và router thông qua các API hướng Nam (southbound APIs), và lên các ứng dụng thông qua các API hướng Bắc (northbound APIs).



Hình 11 Ryu controller

Controller được sử dụng trong hệ thống này là Ryu. Ryu là controller mã nguồn mở được viết bằng Python, được thiết kế để tăng tính linh hoạt của mạng nhờ quản lý và điều chỉnh lưu lượng truy cập.

Ryu bao gồm các yếu tố phần mềm và API để lập trình viên dễ dàng thiết lập, nhiều tính năng mới và các tài liệu hỗ trợ. Ryu có thể được triển khai kết hợp Mininet (cung cấp môi trường thử nghiệm và phát triển ảo cho SDNs) và cho phép tích hợp Snort.



Hình 12 Các thành phần và thư viện trong Ryu controller

Ryu controller thực hiện một số công việc chính:

- Học địa chỉ MAC của host đã kết nối tới port và lưu lại trong bảng địa chỉ MAC (MAC address table).
- Khi nhận gói tin có địa chỉ là host đã học, gửi gói tin này đến port tương ứng với host đã lưu trong MAC address table.
- Khi nhận gói tin có địa chỉ là host chưa xác định, gửi gói tin này cho tất cả các host trong mạng (flooding).

Luồng hoạt động của Ryu controller:

- Trong giao thức OpenFlow, thủ tục đầu tiên cần thực hiện là bắt tay để đảm bảo quá trình giao tiếp giữa OpenFlow switch và controller. Tuy nhiên, ở đây không đề cập đến vì Ryu's framework đảm bảo thủ tục này được hoàn tất.
- Khi controller bắt đầu hoạt động, MAC address table được khởi tạo ở dạng 1 dictionary lồng {Key: Datapath_ ID, value: {key: địa chỉ MAC, value: port}} với Datapath ID dùng để xác định OpenFlow switch.
- Controller sử dụng một số event handler để xử lý các gói tin nhận được, tên lớp sự kiện này là ryu.controller.ofp_event.EventOFP + <OpenFlow message name>. Ví dụ: đối với Packet-In message thì event sẽ là EventOFPPacketIn.
- Mỗi event sẽ được gắn với một trạng thái (state) trên controller với các giá trị như sau:

Định nghĩa	Giải thích
ryu.controller.handler.HANDSHAKE_DISPATCHER	Trao đổi tin nhắn HELLO
ryu.controller.handler.CONFIG_DISPATCHER	Đợi tin nhắn từ switch
ryu.controller.handler.MAIN_DISPATCHER	Trạng thái bình thường
ryu.controller.handler.DEAD_DISPATCHER	Ngắt kết nối

Bảng 3 Các trạng thái của event trên Ryu controller

- Sau khi nhận gói Packet-In từ Open vSwitch gửi đến để xin chỉ dẫn xử lý gói tin, Ryu kiểm tra địa chỉ MAC đích và port tương ứng trong gói tin có trong MAC address table hay chưa.
  - Nếu địa chỉ MAC đích được tìm thấy, 1 flow entry được thêm vào flow table trên Open vSwitch.
  - Ngược lại, flooding gói tin ra cho tất cả các host.
- Song song với việc xử lý chuyển mạch, Ryu controller tạo một luồng (thread) định kỳ gửi yêu cầu tới OpenFlow switch để thu thập thông tin về các flow entry trên các flow table.

5. Giao tiếp giữa Ryu controller và Open vSwitch:

Đầu tiên, controller sẽ lắng nghe các sự kiện Packet-In (ryu.controller.ofp_event. EventOFPPacketIn) để nhận các gói tin từ Open vSwitch. Các Packet-In được xử lý bởi class OFPPacketIn trong Ryu controller. OFPPacketIn class bao gồm các thuộc tính sau:

Tên thuộc tính	Giải thích
match	Thông tin metadata của gói tin đã nhận được thiết lập trong thực thể lớp ryu.ofproto.ofproto_v1_3_parser.OFPMatch.
data	Dữ liệu nhị phân của các gói tin đã nhận.
total_len	Độ dài dữ liệu của gói tin nhận được.
buffer_id	Chỉ ra ID của gói tin được lưu trong bộ đệm của OpenFlow switch, nếu không có thì dùng ryu.ofproto.ofproto_v1_3.OFP_NO_BUFFER

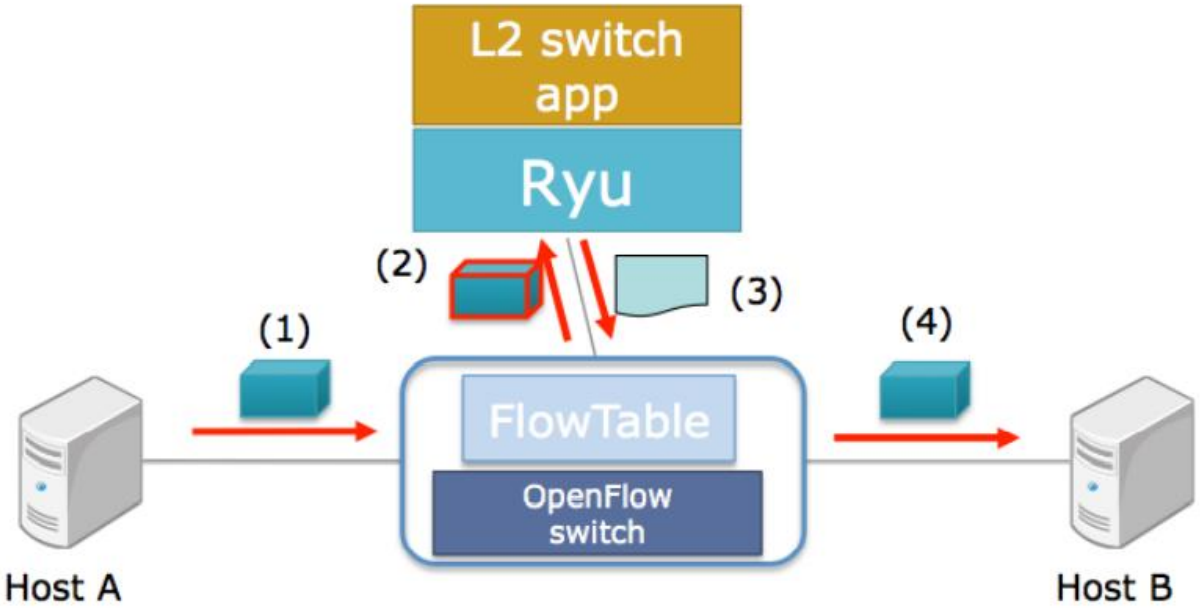
Bảng 4 Thuộc tính OFPPacketIn

Controller thực hiện học địa chỉ MAC nguồn của gói tin và thông tin về port kết nối tương ứng. Sau đó xem xét địa chỉ MAC đích của gói tin nhận được có trong MAC address table hay không.

- Nếu địa chỉ MAC đích đã được biết thì tạo một Packet-Out (OFPPacketOut) gửi gói tin đến port tương ứng, đồng thời sử dụng tin nhắn OFPFlowMod để tạo một flow entry trên switch. Việc add flow trên switch sẽ giảm thiểu việc phải xử lý các Packet-In cho những lần tiếp theo, vì switch đã biết cách xử lý các gói tin cùng loại.
- Nếu host không có trong MAC address table thì sử dụng Packet-Out để flooding gói tin. Packet-Out được định nghĩa trong OFPPacketOut với các thuộc tính:

Tên thuộc tính	Giải thích
datapath	Thông tin meta của gói tin đã nhận được thiết lập trong thực thể lớp ryu.ofproto.ofproto_v1_3_parser.OFPMatch.
buffer_id	Chỉ ra ID của gói tin được lưu trong bộ đệm của OpenFlow switch, nếu không có thì dùng ryu.ofproto.ofproto_v1_3.OFP_NO_BUFFER
data	Chỉ định dữ liệu nhị phân của gói tin. Được dùng khi OFP_NO_BUFFER được gán cho buffer_id. Khi bộ đệm OpenFlow switch được sử dụng, điều này sẽ bị bỏ qua.
in_port	Chỉ định port nhận gói tin, nếu không có thì chỉ định OFPP_CONTROLLER
actions	Chỉ định danh sách các hành động

Bảng 5 Thuộc tính OFPPacketOut



Hình 13 Quy trình xử lý gói tin

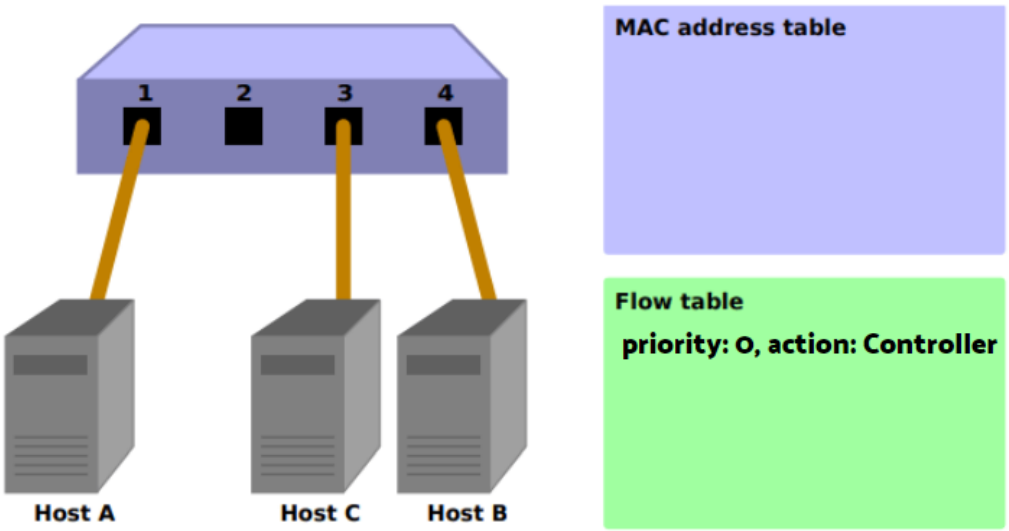
Chi tiết cách hoạt động:

- Khởi tạo:

Lúc này flow table trong Open vSwitch và MAC address table trong Ryu controller đều trống.

Sau quá trình bắt tay giữa Open vSwitch và Ryu, một table-miss flow entry được thêm vào flow table trong Open vSwitch. Ryu ở trạng thái đợi Packet-In.

Giả sử host A kết nối với port 1, host B với port 4 và host C với port 3.



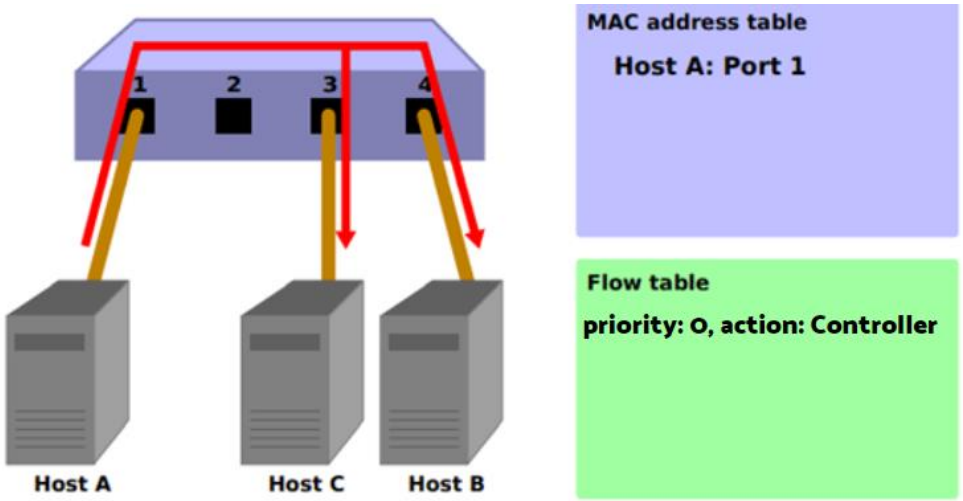
Hình 14 Hoạt động khởi tạo giữa Ryu controller và Open vSwitch

- Host A ping Host B: ARP request

Khi host A gửi gói tin đến host B, Open vSwitch kiểm tra flow table nhận thấy chỉ chứa một entry là table-miss flow entry, nên gói tin được gửi tới Ryu controller bằng hàm Packet-In.

Controller nhận được gói tin, kiểm tra MAC host A - port 1 chưa có trong MAC address table nên thêm vào cặp giá trị này. Vì port tương ứng với host B không được tìm thấy trong MAC address table nên controller sử dụng Packet-Out để flooding gói tin ra toàn mạng. Cả host B và host C đều nhận được gói tin này.

Trước khi gửi gói ICMP echo request, host A không biết địa chỉ MAC của host B, nên gói ARP được gửi đi đầu tiên.



Hình 15 Host A gửi đến host B gói ARP request

Packet-In:

```
in-port: 1
eth-dst: Host B
eth-src: Host A
```

Packet-Out:

```
action: OUTPUT:Flooding
```

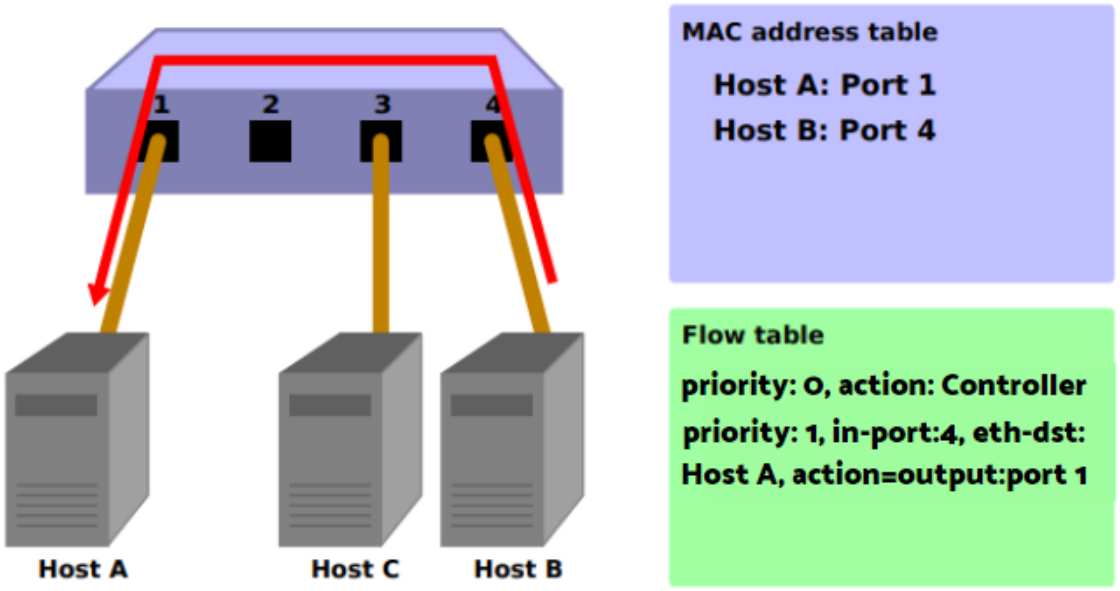
Hình 16 Packet-In và Packet-Out khi host A gửi đến host B gói ARP request

- Host B -> Host A: ARP reply

Khi host B và host C nhận được gói tin flooding, host B nhận ra IP của mình và phản hồi lại Open vSwitch, vì chưa có thêm flow entry nào ngoài table-miss flow entry nên Open vSwitch tiếp tục dùng Packet-In gửi gói tin đến Ryu controller, port 4 tương ứng với MAC của host B được thêm vào MAC address table.

Lúc này trong controller đã lưu địa chỉ MAC của host A và port tương ứng, nên một flow entry từ host B -> host A được thêm vào flow table trong Open

vSwitch và có độ ưu tiên là 1. Khi xem xét các flow entry, flow nào có độ ưu tiên lớn hơn thì được áp dụng trước.



Hình 17 Host B phản hồi host A bằng ARP reply

Packet-In:

```
in-port: 4
eth-dst: Host A
eth-src: Host B
```

Packet-Out:

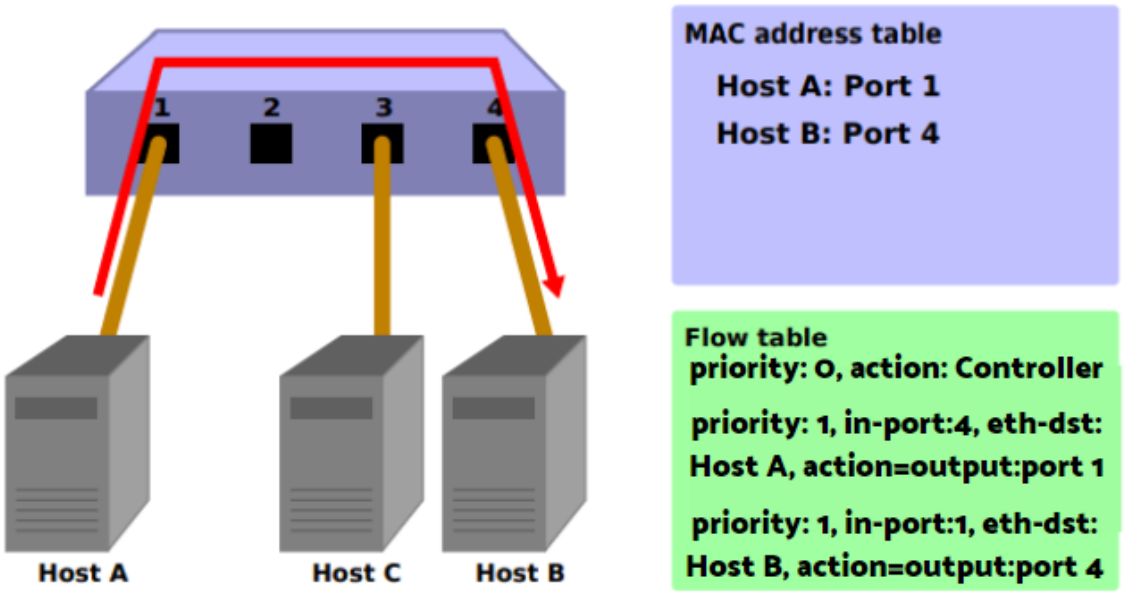
```
action: OUTPUT:Port 1
```

Hình 18 Packet-In và Packet-Out host B phản hồi host A bằng ARP reply

- Host A -> Host B: ICMP echo request

Sau khi biết địa chỉ MAC của host B, host A gửi gói tin ICMP echo request tới host B. Khi gói tin đến Open vSwitch, nó sẽ khớp với table-miss flow entry nên được gửi đến controller.

Khi controller nhận được Packet-In và kiểm tra thấy port tương ứng với MAC của host B nằm trong MAC address table, nó gửi gói tin đến thẳng port 4 (không flooding) và thêm 1 flow entry từ host A -> host B vào flow table trong Open vSwitch với độ ưu tiên là 1.



Hình 19 host A ping host B bằng gói ICMP request

Packet-In:

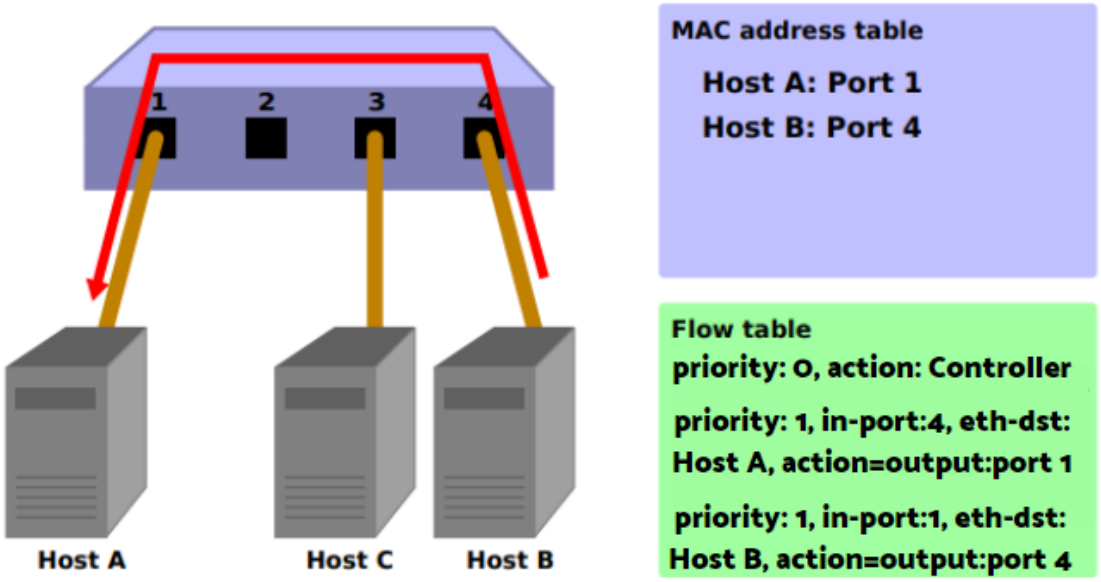
```
in-port: 1
eth-dst: Host B
eth-src: Host A
```

Packet-Out:

```
action: OUTPUT:Port 4
```

Hình 20 Packet-In và Packet-Out khi host A ping host B bằng gói ICMP request

- Host B -> Host A: ICMP echo reply



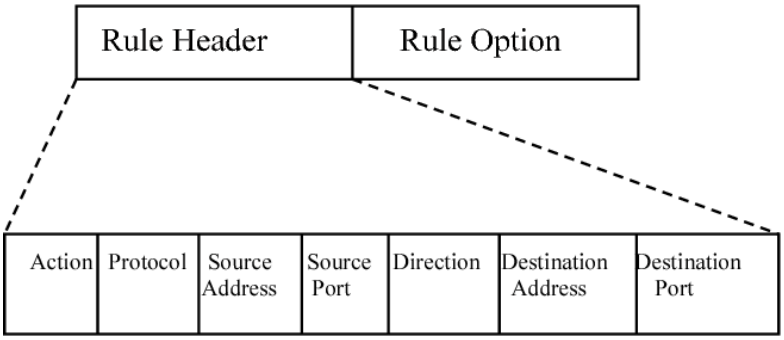
Hình 21 host B trả lời host A bằng gói ICMP reply

Khi host B gửi gói tin ICMP echo reply cho host A, trong Open vSwitch đã có flow entry tương ứng từ host B -> host A nên gói tin được gửi thẳng đến host A mà không cần xin chỉ dẫn từ Ryu controller.

6. Snort:

Snort là giải pháp phát hiện và ngăn chặn xâm nhập mạng mã nguồn mở. Trong đồ án này, Snort đóng vai trò là hệ thống phát hiện xâm nhập mạng (network-based IDS).

Snort dựa trên quy tắc do người dùng viết hay sử dụng từ cộng đồng Snort và của nhà cung cấp.



Hình 22 Cấu trúc các quy tắc của Snort

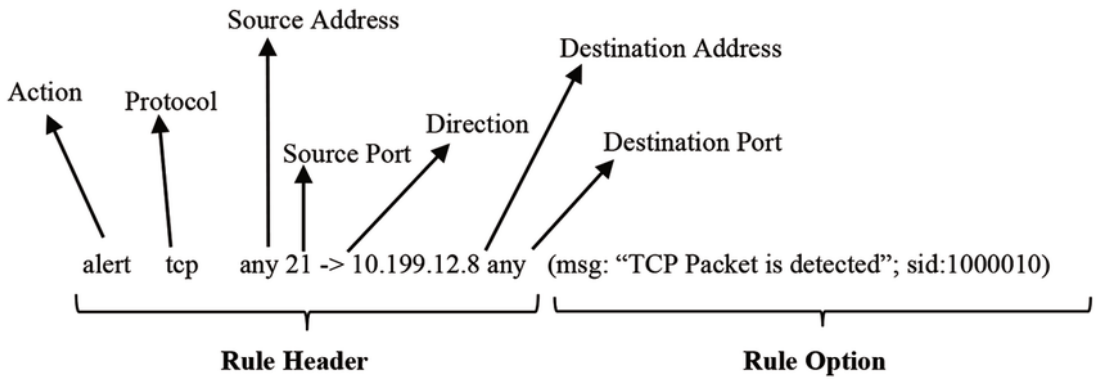
Phần đầu quy tắc (Rule Header) bao gồm:

- Action:
  - alert – tạo cảnh báo bằng các phương thức được chọn, và ghi log gói tin vào file.
  - log – ghi log gói tin vào file.
  - pass – bỏ qua gói tin.
  - activate – cảnh báo và kích hoạt một quy tắc động khác.
  - dynamic – duy trì chế độ chờ đến khi được kích hoạt bởi một quy tắc khác và hoạt động như quy tắc ghi log.
- Protocol bao gồm:
  - TCP
  - UDP
  - ICMP
- Source Address, Destination Address: có thể là một dãy IP hoặc một IP cụ thể.
- Direction có thể là -> (hướng từ source address đến destination address) hoặc <> (từ source address đến destination address và ngược lại).

Phần tùy chọn (Rule Option): có nhiều tùy chọn tuy nhiên phần này chỉ trình bày những tùy chọn được sử dụng trong đồ án.

- msg: Tin nhắn sẽ được in ra log hoặc cảnh báo cho người quản trị.
- dsize: Kiểm tra kích thước gói tin, tùy chọn này có thể kiểm tra lỗi tràn bộ đệm (buffer overflow) nhanh hơn kiểm tra cả nội dung của gói tin.
- flags: kiểm tra các cờ TCP, bao gồm:
  - F - FIN (LSB trong byte của cờ TCP)
  - S – SYN
  - R – RST
  - P – PSH
  - A – ACK
  - U – URG
  - 2 – Lưu trữ bit 2
  - 1 - Lưu trữ bit 1 (MSB trong byte của cờ TCP)
- flow: Được sử dụng trong các luồng TCP, chỉ định hướng cho các luồng hoạt động. Bao gồm:
  - to_client: Kích hoạt trên phản hồi từ máy chủ.
  - to_server: Kích hoạt trên yêu cầu từ máy khách.
  - from_client: Kích hoạt trên yêu cầu từ máy khách.
  - from_server: Kích hoạt trên phản hồi từ máy chủ.
  - established: Kích hoạt trên các kết nối TCP đã thiết lập. Snort không quan tâm đến các gói tin bắt tay 3 bước (thông thường không có dữ liệu bên trong các gói tin này).
  - not_established: Kích hoạt khi không có kết nối TCP nào được thiết lập.
  - stateless: Kích hoạt bất kể trạng thái của bộ xử lý luồng (hữu ích cho các gói được thiết kế để gây ra sự cố máy móc)
  - no_stream: Không kích hoạt trên các gói xây dựng lại luồng (hữu ích cho dsize và stream5)
  - only_stream: Chỉ kích hoạt trên các gói xây dựng lại luồng.
  - no_frag: Không kích hoạt trên các gói xây dựng lại các phân mảnh.
  - only_frag: Chỉ kích hoạt trên các gói xây dựng lại các phân mảnh.
- itype: Snort nhận ra các lưu lượng sử dụng giá trị ICMP này. Các giá trị nằm ngoài phạm vi cũng có thể được thiết lập để phát hiện lưu lượng truy cập đáng ngờ.

- detection_filter: tùy chọn mới thay thế từ khóa threshold. Nó xác định ngưỡng vượt qua của một máy chủ nguồn hoặc đích, trước khi một quy tắc tạo ra sự kiện. Detection_filter có định dạng như sau:
  - detection_filter: track <by_src|by_dst>, count <value>, seconds <value>.
  - Trong đó:
    - track by_src|by_dst: chỉ định địa chỉ ip nguồn hoặc đích cần được theo dõi.
    - count: số lượng gói tin tối đa đến trong 1 đơn vị thời gian trước khi vượt ngưỡng của bộ lọc.
    - seconds: đơn vị thời gian tính bằng giây. (giá trị khác 0).



Hình 23 Ví dụ một quy tắc trong Snort

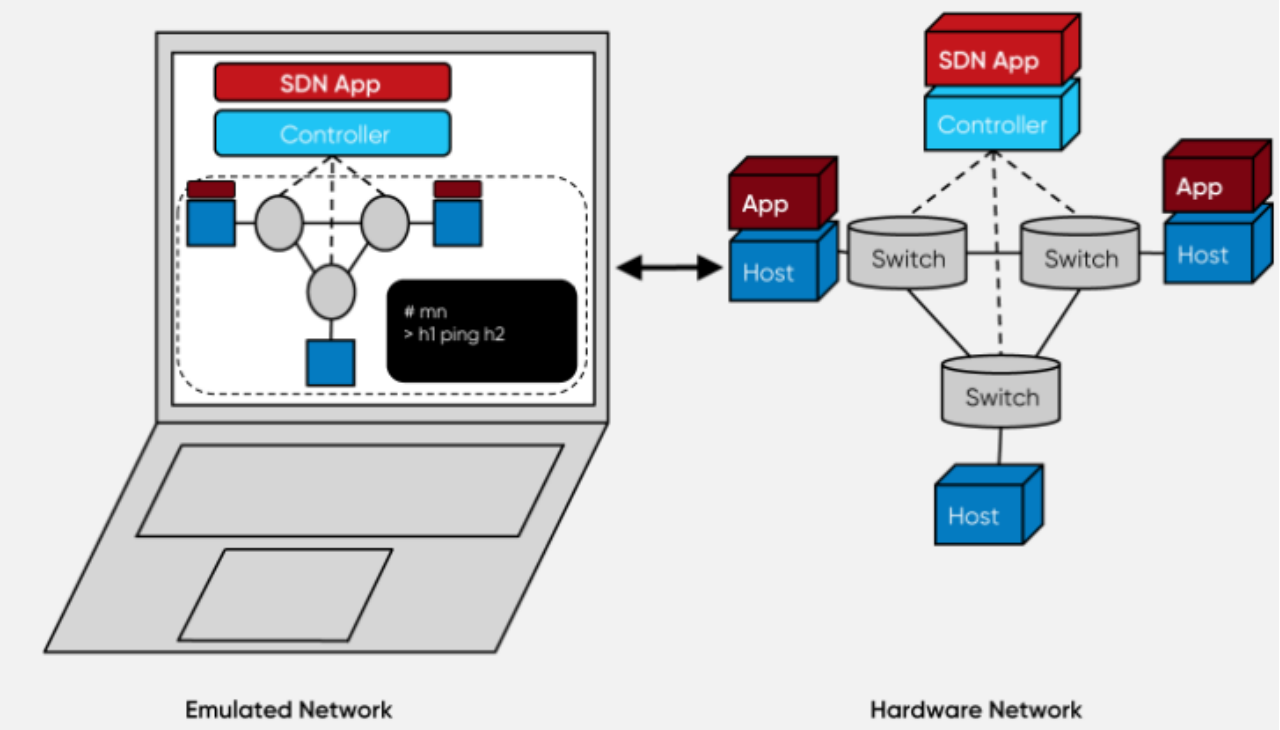
Snort thực hiện phân tích giao thức, tìm kiếm kết hợp nội dung và có thể được sử dụng để phát hiện nhiều loại tấn công và thăm dò khác nhau như buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts…

Snort hoạt động ở 3 chế độ:

- Package sniffer: hiển thị thông tin header các gói tin
- Package log: ghi lại các thông tin vào file log để xử lý sau này
- IDS: phân tích các gói tin hoặc các luồng TCP, thực hiện các chức năng IDS theo cơ chế dựa trên dấu hiệu (signature-based).

7. Mininet:

Mininet là một công cụ giả lập mạng, bao gồm tập hợp các máy chủ đầu cuối, các thiết bị chuyển mạch, thiết bị định tuyến và các liên kết trên một nhân Linux. Các máy chủ Mininet chạy phần mềm mạng Linux tiêu chuẩn, các thiết bị chuyển mạch hỗ trợ OpenFlow để định tuyến tùy chỉnh linh hoạt và SDN. Mininet sử dụng công nghệ ảo hóa (ở mức đơn giản) để tạo nên hệ thống mạng hoàn chỉnh, chạy chung trên cùng một nhân, hệ thống và mã nguồn của người dùng.



Hình 24 Mininet

Các máy chủ ảo, thiết bị chuyển mạch, liên kết và các bộ điều khiển trên Mininet là các thực thể thực sự, được giả lập dưới dạng phần mềm thay vì phần cứng. Một máy chủ Mininet có thể thực hiện SSH vào đó, chạy bất kì phần mềm nào đã cài trên hệ thống Linux (môi trường mà Mininet đang chạy). Các phần mềm này có thể gửi gói tin thông qua các ethernet interface của Mininet với tốc độ liên kết và độ trễ đặt trước.

Mininet cho phép tạo và tùy chỉnh cấu trúc (topology) mạng nhanh chóng, chạy được các phần mềm thực sự như web servers, TCP monitoring, Wireshark; tùy chỉnh được việc chuyển tiếp gói tin. Mininet cũng dễ dàng sử dụng và không yêu cầu cấu hình đặc biệt gì về phần cứng để chạy: Mininet có thể cài trên laptop, server, VM, cloud (Linux).

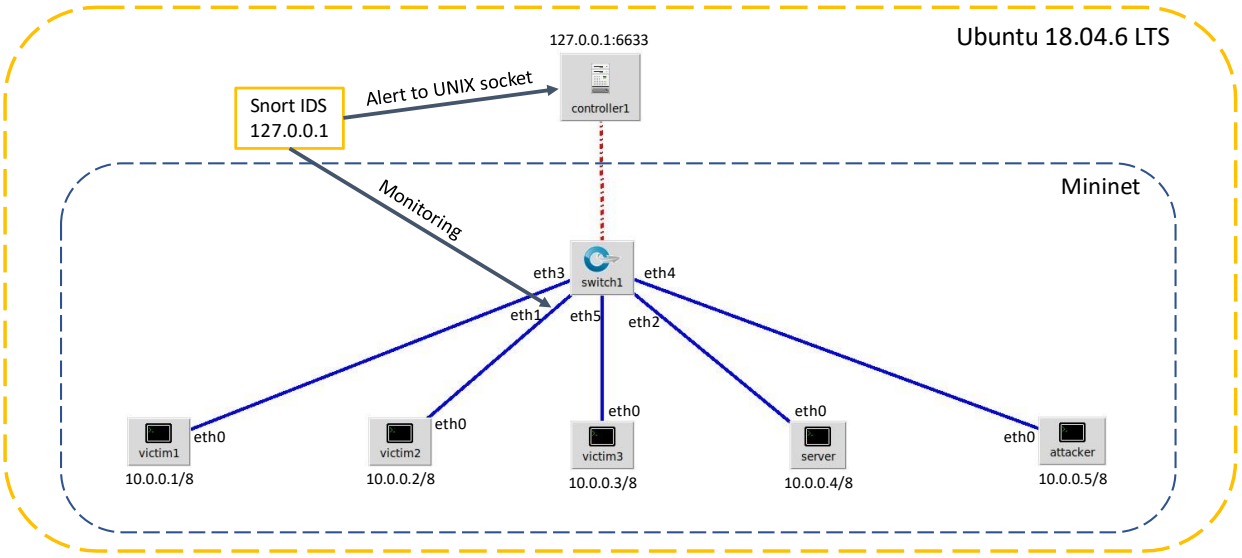
Gần như mọi hệ điều hành đều ảo hóa các tài nguyên máy tính bằng cách sử dụng một tiến trình trừu tượng hóa. Mininet sử dụng ảo hóa dựa trên tiến trình để chạy nhiều máy chủ và chuyển đổi trên một nhân hệ điều hành duy nhất. Kể từ phiên bản 2.2.26, Linux đã hỗ trợ network namespaces, một tính năng ảo hóa nhẹ cung cấp các tiến trình riêng lẻ với giao diện mạng, bảng định tuyến và bảng ARP riêng biệt. Kiến trúc vùng chứa (container architecture) Linux thêm chroot () jails, user namespaces và tiến trình cũng như giới hạn CPU và bộ nhớ để cung cấp ảo hóa cấp hệ điều hành đầy đủ, nhưng Mininet không yêu cầu các tính năng bổ sung này. Mininet có thể tạo kernel hoặc các user-space OpenFlow switch, controller để điều khiển các switch và các Host để giao tiếp qua mạng mô phỏng. Mininet kết nối các switch và các Host bằng các cặp ethernet ảo (veth).

8. Mô hình hoàn chỉnh:

IPSDN được xây dựng nhờ Mininet. Mininet chủ yếu được sử dụng để tạo ra mạng ảo, có thể thiết lập các kết nối mạng theo ý muốn nhờ GUI và CLI. Mininet GUI được dùng để thiết kế các topology của mạng với số lượng host và các switch theo yêu cầu.

Sau khi kết nối các host trong mạng tới switch, cần phải thêm controller vào để tạo thành SDN hoàn chỉnh. Controller có nhiệm vụ giám sát và điều tiết traffic trong mạng. Nhờ Mininet ta có thể mô phỏng mạng vậy lý và đánh giá hiệu quả hoạt động của chúng thay vì phải xây dựng một hệ thống vật lý có thể có nhiều khó khăn trong thời gian đầu khởi tạo.

OpenFlow là giao thức được sử dụng trong hệ thống này, giúp controller giao tiếp với Forwarding plane. Đây là một giao thức tiêu chuẩn trong mạng SDN. Thông qua OpenFlow, SDN controller ghi các thay đổi đã xảy ra với Forwarding plane. Nhờ tính năng, phân vùng lưu lượng, kiểm soát luồng để có hiệu suất lý tưởng, thử nghiệm các ứng dụng mới và cấu hình, ...có thể được thực hiện bởi quản trị viên mạng.



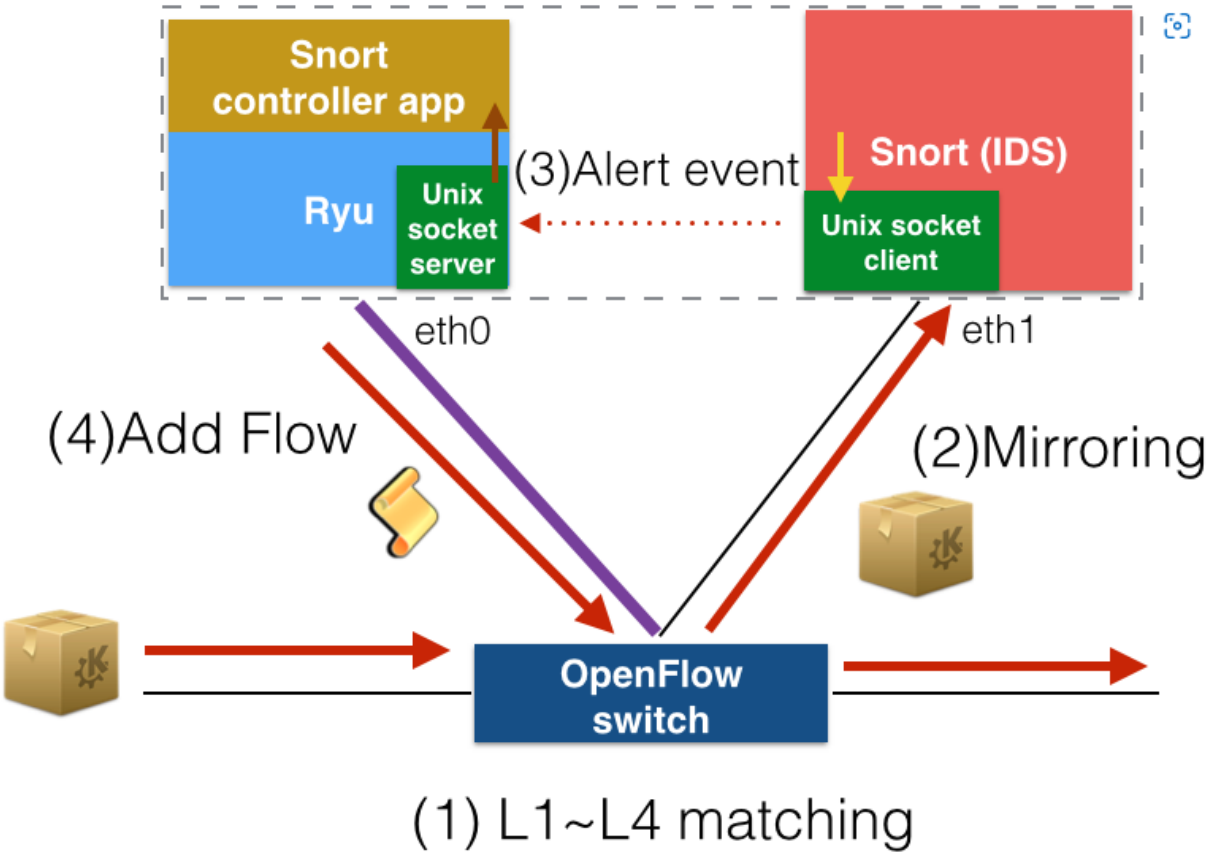
Hình 25 Kiến trúc IPSDN

Trong hệ thống này, controller kết nối với Open vSwitch thông qua giao thức OpenFlow đã trình bày ở trên.

Snort đóng vai trò là IDS, có nhiệm vụ monitor traffic trên 1 ethernet interface cụ thể của Open vSwitch, do đó luôn có thêm một output port được thêm vào các flow entry trên Open vSwitch để mirror (copy) traffic đến interface mà Snort đang chạy. Snort dựa vào các rule được định nghĩa để phát hiện các hành động độc hại dựa trên dấu hiệu (signature-based), sau đó đưa ra cảnh báo đến Ryu thông qua kết nối Unix socket. Khi Ryu nhận được cảnh báo từ Snort, nó sẽ add flow trên Open vSwitch để chặn các traffic độc hại này với độ ưu tiên cao hơn các flow entry thông thường.

Controller đóng vai trò là IPS, định kỳ lấy các flow có trong Open vSwitch và phân tích dựa trên Deep Learning để phát hiện các hành động bất thường (anomaly-based). Khi phát hiện các flow độc hại, controller sẽ add flow Open vSwitch để chặn

các traffic độc hại đến từ địa chỉ MAC của attacker với độ ưu tiên cao hơn các flow entry thông thường.



Hình 26 Snort signature-based IDS kết hợp Ryu controller

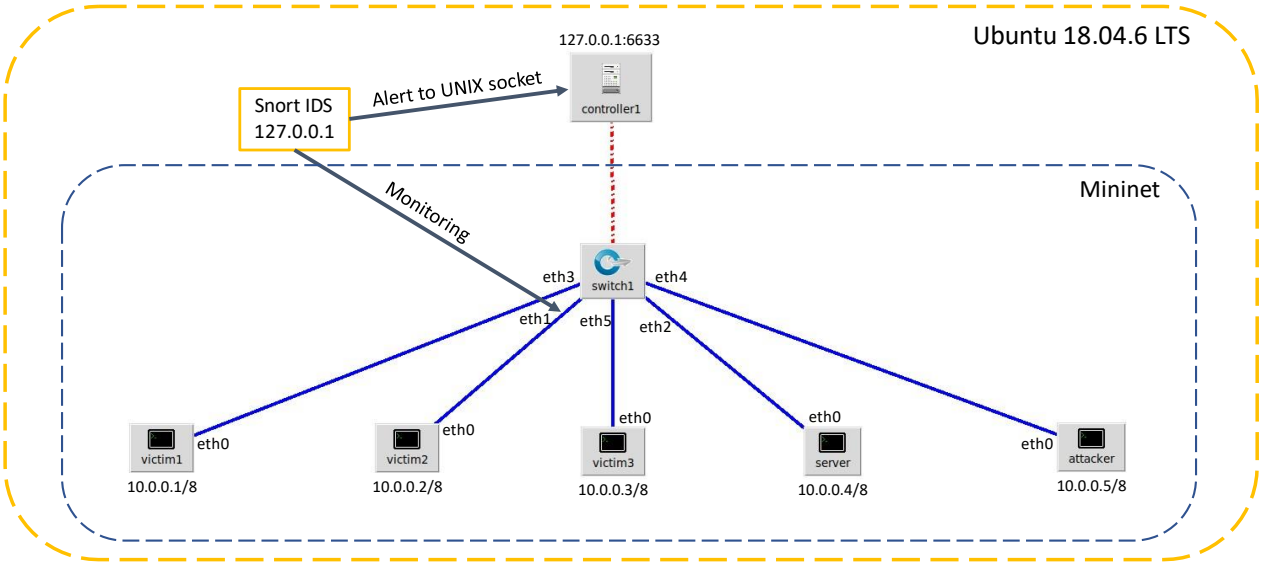
Các flow entry được chọn áp dụng lên gói tin nhờ độ ưu tiên, flow nào có độ ưu tiên càng cao thì được áp dụng trước nhất. Trong đồ án này, các flow entry có độ ưu tiên từ thấp đến cao lần lượt là:

Flow entry	Độ ưu tiên	Idle_timeout
Table-miss	0	
Cho phép	1	15
Chặn traffic bất thường	2	15
Chặn traffic theo dấu hiệu	3	15

Bảng 6 Flow entry, Độ ưu tiên và idle_timeout tương ứng

III. Thực nghiệm và kết quả:

IPSDN được triển khai trên một máy ảo Ubuntu 18.04 LTS (VMware) trên máy thật chạy Windows 11 với 16GB bộ nhớ RAM và CPU AMD Ryzen 5 3550H (4 nhân 8 luồng, xung nhịp 2.1 – 3.7 GHz). Máy ảo được cấp 8GB RAM và 2 nhân 4 luồng CPU.



1. Các bước thực hiện:

Bước 1: Khởi tạo các thiết bị mạng và hệ thống mạng.

Bước 2: Khởi tạo Unix socket giao tiếp giữa controller và Snort IDS.

Bước 3: Ánh xạ gói tin từ switch đến Snort IDS.

Bước 4: Phát hiện tấn công dựa trên các rule được định nghĩa trong Snort IDS.

Bước 5: Nếu khớp các rule, gửi sự kiện liên quan (alert) đến SDN controller.

Bước 6: Controller nhận sự kiện và ra lệnh cho switch drop các gói tin đến từ các host tấn công hoặc độc hại.

2. Mô phỏng các cuộc tấn công:

Loại hình tấn công được mô phỏng là DoS (Denial of Service) và công cụ thực hiện tấn công là Scapy. Các cuộc tấn công DoS được thực hiện như sau:

SYN Flooding: khai thác quá trình bắt tay 3 bước của TCP. Attacker gửi một khối lượng lớn các gói tin tin SYN đến máy nạn nhân và giả mạo địa chỉ IP nguồn. Máy nạn nhân gửi lại gói tin SYN/ACK, đợi attacker trả lời và để lại một cổng mở. Attacker không gửi gói ACK hoàn tất quá trình bắt tay mà tiếp tục gửi đi các gói SYN khác để thiết lập kết nối mới. Việc nhận quá nhiều gói SYN và mở nhiều cổng đợi phản hồi làm máy nạn nhân tiêu tốn tài nguyên và từ chối dịch vụ.

UDP Flooding: gửi một số lượng lớn các gói tin UDP tới cổng ngẫu nhiên trên máy nạn nhân và máy này sẽ:

- Kiểm tra các ứng dụng với cổng.
- Thấy rằng không có ứng dụng nghe ở cổng.
- Trả lời với một gói ICMP Destination Unreachable.

Khi số lượng request vượt ngưỡng này, máy nạn nhân sẽ mất khả năng xử lý các yêu cầu thông thường dẫn đến tình trạng từ chối dịch vụ. Lợi dụng đặc tính không duy trì trạng thái của UDP, attacker sử dụng IP giả mạo. Trường hợp này sẽ rất khó để xác định được nguồn tấn công thực sự.

Smurf: Attacker gửi gói tin ICMP echo request với địa chỉ đích là địa chỉ broadcast và địa chỉ nguồn là địa chỉ của máy nạn nhân. Theo đó tất cả các máy nằm trong mạng broadcast sẽ đồng loạt trả lời về địa chỉ của máy nạn nhân bằng gói tin ICMP echo reply. Kết quả là máy tính này sẽ không thể xử lý kịp thời một lượng lớn thông tin và dẫn tới bị treo máy. Chỉ cần một lượng nhỏ các gói tin ICMP đi đến mạng broadcast sẽ khuếch đại lượng gói tin lên gấp bội. Tỷ lệ khuếch đại phụ thuộc vào số máy tính trong mạng broadcast. Vì thế attacker chiếm được càng nhiều hệ thống mạng hoặc router cho phép chuyển trực tiếp các gói tin đến địa chỉ broadcast không qua chỗ lọc địa chỉ nguồn ở các đầu ra của gói tin thì cuộc tấn công càng dễ dàng hơn.

LAND: Attacker gửi một gói SYN được chỉnh sửa để địa chỉ IP nguồn và đích giống nhau. Khi máy nạn nhân cố gắng trả lời tin nhắn này, nó sẽ vào một vòng lặp bằng cách tạo lại các câu trả lời cho chính mình dẫn đến một kịch bản lỗi và cuối cùng có thể bị sập.

Multi SYN Flooding: tương tự SYN Flooding nhưng nhắm vào nhiều hơn 1 máy mục tiêu.

Multi UDP Flooding: tương tự UDP Flooding nhưng nhắm vào nhiều hơn 1 máy mục tiêu.

3. Signature-based IPS:

Trong thí nghiệm hoạt động của IPS dựa trên dấu hiệu. Các cuộc tấn công SYN flooding và UDP flooding được thực hiện từ máy attacker (10.0.0.5) nhắm vào máy server (10.0.0.4).

Controller thực hiện lắng nghe các Snort alert từ Unix socket (từ network socket trong trường hợp Snort và Ryu nằm trên 2 máy khác nhau) thông qua sự kiện EventAlert trong thư viện snortlib.

Công cụ Scapy được sử dụng để gửi tổng cộng 1000 packet (TCP SYN/UDP) từ máy attacker đến server với tốc độ 20 packet/giây.

Snort được cấu hình để alert với giá trị threshold là 10 packet/giây:

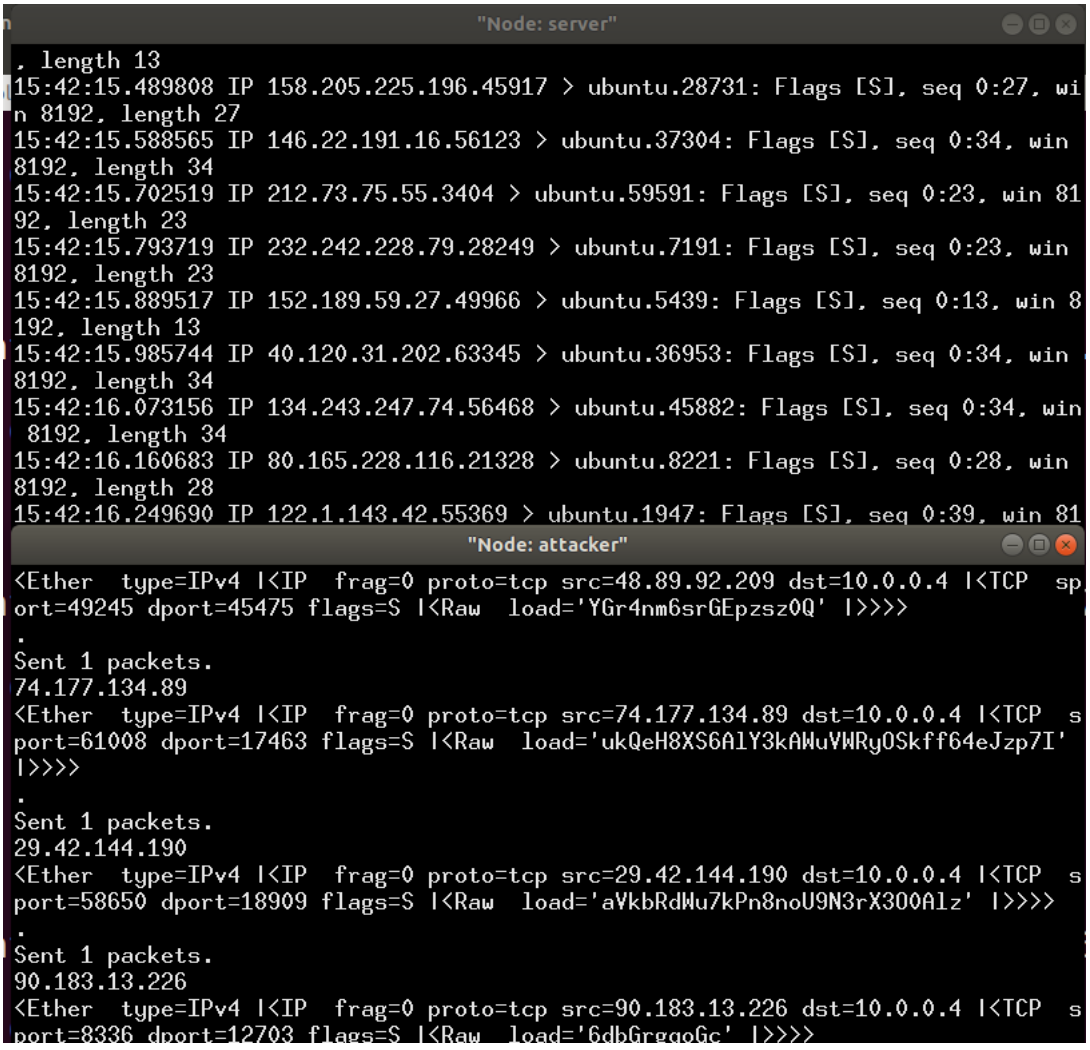
```
#SYN flooding
alert tcp any any -> any any (sid:1000001; rev:1; flags:S; msg:"TCP SYN flooding attack"; flow:not_established; detection_filter: track by_dst, count 10, seconds 1;)
```

```
#UDP flooding
alert udp any any -> any any (sid:1000004; rev:1; msg:"UDP flooding attack"; detection_filter: track by_dst, count 10, seconds 1;)
```

Thực hiện khởi chạy controller và Mininet, sau đó thực hiện câu lệnh pingall. Các flow trên switch trước khi thực hiện attack:

```
mininet> dpctl dump-flows -O OpenFlow13
*** switch1 ------------------------------------------------------------------------
 cookie=0x0, duration=13.396s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth2",dl_src=fa:46:af:1a:b5:3b,
dl_dst=2a:e8:ee:d1:24:df,nw_src=10.0.0.4,nw_dst=10.0.0.2 actions=output:"switch1-eth1",output:"switch1-eth1"
 cookie=0x0, duration=13.393s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth1",dl_src=2a:e8:ee:d1:24:df,
dl_dst=fa:46:af:1a:b5:3b,nw_src=10.0.0.2,nw_dst=10.0.0.4 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=13.383s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth2",dl_src=fa:46:af:1a:b5:3b,
dl_dst=fe:2f:6f:8f:1e:67,nw_src=10.0.0.4,nw_dst=10.0.0.5 actions=output:"switch1-eth4",output:"switch1-eth1"
 cookie=0x0, duration=13.381s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,
dl_dst=fa:46:af:1a:b5:3b,nw_src=10.0.0.5,nw_dst=10.0.0.4 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=13.375s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth2",dl_src=fa:46:af:1a:b5:3b,
dl_dst=52:62:9c:28:1e:a4,nw_src=10.0.0.4,nw_dst=10.0.0.3 actions=output:"switch1-eth5",output:"switch1-eth1"
 cookie=0x0, duration=13.373s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth5",dl_src=52:62:9c:28:1e:a4,
dl_dst=fa:46:af:1a:b5:3b,nw_src=10.0.0.3,nw_dst=10.0.0.4 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=13.364s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth2",dl_src=fa:46:af:1a:b5:3b,
dl_dst=76:7f:87:78:a7:cd,nw_src=10.0.0.4,nw_dst=10.0.0.1 actions=output:"switch1-eth3",output:"switch1-eth1"
 cookie=0x0, duration=13.362s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth3",dl_src=76:7f:87:78:a7:cd,
dl_dst=fa:46:af:1a:b5:3b,nw_src=10.0.0.1,nw_dst=10.0.0.4 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=13.353s, table=0, n_packets=2, n_bytes=196, idle_timeout=15, priority=1,icmp,in_port="switch1-eth1",dl_src=2a:e8:ee:d1:24:df
,dl_dst=fe:2f:6f:8f:1e:67,nw_src=10.0.0.2,nw_dst=10.0.0.5 actions=output:"switch1-eth4",output:"switch1-eth1"
 cookie=0x0, duration=13.351s, table=0, n_packets=1, n_bytes=98, idle_timeout=15, priority=1,icmp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,
dl_dst=2a:e8:ee:d1:24:df,nw_src=10.0.0.5,nw_dst=10.0.0.2 actions=output:"switch1-eth1",output:"switch1-eth1"
```

Thực hiện TCP SYN flooding từ attacker đến server:



Quan sát các flow trên switch, ta nhận thấy rất nhiều flow lạ vì số lượng IP được giả mạo là rất nhiều và các flow được match dựa trên thuộc tính thành phần là IPv4 và TCP/UDP port (layer 3,4):

```
 cookie=0x0, duration=9.693s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=67.76.69.127,nw_dst=10.0.0.4,tp_src=2022,tp_dst=46916 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=9.597s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=23.213.174.135,nw_dst=10.0.0.4,tp_src=35247,tp_dst=40280 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=9.513s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=82.176.57.148,nw_dst=10.0.0.4,tp_src=34958,tp_dst=22634 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=9.409s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=28.95.115.210,nw_dst=10.0.0.4,tp_src=36248,tp_dst=41354 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=9.302s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=48.79.212.188,nw_dst=10.0.0.4,tp_src=24016,tp_dst=26662 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=9.193s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=184.29.217.88,nw_dst=10.0.0.4,tp_src=419,tp_dst=39465 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=9.105s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=177.240.170.17,nw_dst=10.0.0.4,tp_src=9107,tp_dst=29168 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=9.001s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=68.130.123.192,nw_dst=10.0.0.4,tp_src=25118,tp_dst=19017 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=8.894s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=43.29.18.165,nw_dst=10.0.0.4,tp_src=37227,tp_dst=8515 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=8.803s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=157.42.198.162,nw_dst=10.0.0.4,tp_src=23187,tp_dst=27581 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=8.690s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=31.238.161.99,nw_dst=10.0.0.4,tp_src=11423,tp_dst=6531 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=8.594s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=198.161.134.115,nw_dst=10.0.0.4,tp_src=33420,tp_dst=32113 actions=output:"switch1-eth2",output:"switch1-eth1"
 cookie=0x0, duration=8.499s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
```

Snort phát hiện cuộc tấn công và gửi các cảnh báo (alert) tới Ryu controller:

```
Alertmsg: TCP SYN flooding attack
ipv4(csum=10344,dst='10.0.0.4',flags=0,header_length=5,identification=1,offset=0,option=None,proto=6,src='40.120.31.202',tos=0,total_length=74,ttl
=64,version=4)
ethernet(dst='fa:46:af:1a:b5:3b',ethertype=2048,src='fe:2f:6f:8f:1e:67')
Block all traffic from fe:2f:6f:8f:1e:67
Alert number 1 takes 0.00977802276611 seconds to process
Alertmsg: TCP SYN flooding attack
ipv4(csum=62059,dst='10.0.0.4',flags=0,header_length=5,identification=1,offset=0,option=None,proto=6,src='134.243.247.74',tos=0,total_length=74,tt
l=64,version=4)
ethernet(dst='fa:46:af:1a:b5:3b',ethertype=2048,src='fe:2f:6f:8f:1e:67')
Block all traffic from fe:2f:6f:8f:1e:67
Alert number 2 takes 0.00984382629395 seconds to process
Alertmsg: TCP SYN flooding attack
ipv4(csum=15254,dst='10.0.0.4',flags=0,header_length=5,identification=1,offset=0,option=None,proto=6,src='80.165.228.116',tos=0,total_length=68,tt
l=64,version=4)
ethernet(dst='fa:46:af:1a:b5:3b',ethertype=2048,src='fe:2f:6f:8f:1e:67')
Block all traffic from fe:2f:6f:8f:1e:67
Alert number 3 takes 0.008624792099 seconds to process
Alertmsg: TCP SYN flooding attack
ipv4(csum=26489,dst='10.0.0.4',flags=0,header_length=5,identification=1,offset=0,option=None,proto=6,src='122.1.143.42',tos=0,total_length=79,ttl=
64,version=4)
ethernet(dst='fa:46:af:1a:b5:3b',ethertype=2048,src='fe:2f:6f:8f:1e:67')
Block all traffic from fe:2f:6f:8f:1e:67
Alert number 4 takes 0.0103299617767 seconds to process
Alertmsg: TCP SYN flooding attack
```

Sau khi nhận được các cảnh báo, controller thực hiện add flow trên switch để chặn đứng cuộc tấn công:

```
mininet> dpctl dump-flows -O OpenFlow13
*** switch1 ------------------------------------------------------------------------
 cookie=0x0, duration=3.903s, table=0, n_packets=46, n_bytes=3637, idle_timeout=15, priority=2,dl_src=fe:2f:6f:8f:1e:67 actions=drop
 cookie=0x0, duration=9.798s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=fe:2f:6f:8f:1e:67,dl_
dst=fa:46:af:1a:b5:3b,nw_src=80.194.191.235,nw_dst=10.0.0.4,tp_src=23882,tp_dst=18106 actions=output:"switch1-eth2",output:"switch1-eth1"
```

Từ thời điểm này, switch sẽ loại bỏ các traffic đến từ MAC của attacker, ngăn không cho chúng đến với server.

Đối với tấn công UDP flooding, quy trình hoạt động của IPSDN diễn ra theo cơ chế tương tự.

Thời gian trung bình để Snort xử lý một alert (thời gian từ lúc tiếp nhận alert đến khi hoàn thành add flow trên switch) là 0.009547551s = 9,5ms.

4. Anomaly-based IPS:

Chức năng phát hiện xâm nhập dựa trên các flow bất thường (flow-based anomaly detection hay anomaly-based IPS) là chức năng phụ hỗ trợ cho signature-based IPS trong IPSDN. Song song với chức năng signature-based IPS, controller sẽ tạo một thread riêng để định kỳ thu thập các flow từ switch và sử dụng mô hình học sâu (Deep Learning) để phân loại flow và ra lệnh cho switch block traffic nếu phát hiện flow bất thường. Cơ chế block tương tự như signature-based IPS.

Cứ sau một khoảng nghỉ nhất định, controller sẽ gửi tin nhắn OFPFlowStatsRequest và chờ đợi sự kiện EventOFPFlowStatsReply chứa đựng thông tin về các Flow trên switch. Sau đó một mô hình học sâu sẽ được dùng để dự đoán xem các flow đó có độc hại hay là không.

Trong trường hợp Snort bỏ qua các traffic độc hại thì IPSDN vẫn còn một lớp bảo vệ nữa là mô hình học sâu, cung cấp khả năng phòng chống tấn công toàn diện hơn.

Các trường dữ liệu trong flow entry được sử dụng để huấn luyện mô hình bao gồm: ip-protocol (UDP/TCP/ICMP), duration (thời gian tồn tại của flow entry), byte-count (tổng dung lượng của các packet đã match với flow entry) và packet-count (tổng số lượng packet được match bởi flow entry). Trên lý thuyết, các flow có lưu lượng dữ liệu lớn trong một khoảng thời gian ngắn là các flow độc hại (DoS).

Dữ liệu được dùng để huấn luyện mô hình được thu thập thủ công thông qua thao tác tạo ra các traffic lành tính (được gán nhãn cho flow là 0) cũng như traffic độc hại và gán nhãn 1 cho các flow tương ứng.

Hình ảnh dữ liệu thu thập được dùng để train model:

Duration	Ip-protocol	Byte-count	Packet-count	Class
4	17	240	4	0
3	17	693	9	0
4	6	702	9	0
0	6	156	2	0
5	6	384	6	0
0	17	0	0	1
5	17	462	6	0
15	6	0	0	1
2	6	0	0	1
13	6	460	5	0
14	6	0	0	1
6	17	0	0	1
5	17	438	6	0

Bằng phương pháp học sâu sử dụng thuật toán Multi-layer Perceptron (MLP classifier), tỉ lệ dự đoán chính xác (accuracy score) dựa trên tập test gồm gần 1000 flow độc hại và lành tính đạt được là 92.7725674091442%.

Hiện thực lại ngữ cảnh tấn công TCP SYN flooding từ attacker đến server và sử dụng cơ thế anomaly-based IPS (không khởi chạy Snort). Ta thấy flow độc hại đã bị phát hiện bởi controller và một drop flow đã được thêm trên switch:

```
CPU usage in percent: 1.5
Sending flow stats request for datapath 1
Receiving flow stats from datapath 1

Anomaly detection of flow entry 1 6 0 0
Anomaly detected, configuring datapath 1 to block traffic from ea:70:e9:c8:b0:c9

Anomaly detection of flow entry 1 6 0 0
Anomaly detected, configuring datapath 1 to block traffic from ea:70:e9:c8:b0:c9

Anomaly detection of flow entry 1 6 0 0
Anomaly detected, configuring datapath 1 to block traffic from ea:70:e9:c8:b0:c9

Anomaly detection of flow entry 0 6 0 0
Anomaly detected, configuring datapath 1 to block traffic from ea:70:e9:c8:b0:c9

Anomaly detection of flow entry 0 6 0 0
Anomaly detected, configuring datapath 1 to block traffic from ea:70:e9:c8:b0:c9

Anomaly detection of flow entry 0 6 0 0
Anomaly detected, configuring datapath 1 to block traffic from ea:70:e9:c8:b0:c9

Anomaly detection of flow entry 0 6 0 0
Anomaly detected, configuring datapath 1 to block traffic from ea:70:e9:c8:b0:c9
```

```
mininet> dpctl dump-flows -O OpenFlow13
*** switch1 ------------------------------------------------------------------------
 cookie=0x0, duration=8.046s, table=0, n_packets=86, n_bytes=6739, idle_timeout=15, priority=2,dl_src=ea:70:e9:c8:b0:c9 actions=drop
 cookie=0x0, duration=14.853s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=ea:70:e9:c8:b0:c9,dl
_dst=a6:54:5c:a0:f7:b8,nw_src=140.67.206.32,nw_dst=10.0.0.4,tp_src=9999,tp_dst=2324 actions=output:"switch1-eth2"
 cookie=0x0, duration=14.756s, table=0, n_packets=0, n_bytes=0, idle_timeout=15, priority=1,tcp,in_port="switch1-eth4",dl_src=ea:70:e9:c8:b0:c9,dl
```

Thời gian trung bình để controller xử lý một flow stats reply và add block flow là 0.010s = 10ms.

IV. Kết luận:

Trong đồ án này, nhóm đã thực hiện triển khai hệ thống phòng chống xâm nhập thông minh trong các mạng điều khiển bằng phần mềm (IPSDN), thực hiện kiểm thử bằng cách tạo ra nhiều cuộc tấn công DoS khác nhau để quan sát khả năng hoạt động của mô hình.

Kết quả nhận được cho thấy IPSDN có khả năng phát hiện và ngăn chặn các cuộc tấn công kể trên một cách hiệu quả, nhanh chóng mà không tiêu tốn nhiều tài nguyên phần cứng (không quá 3% CPU usage). Bên cạnh đó, IPSDN còn một số nhược điểm

cần cải thiện như lượng dữ liệu training mô hình còn hạn chế và chưa thực sự sát với ngữ cảnh các cuộc tấn công trong thực tế, và số lượng tấn công mô phỏng chưa đủ độ phong phú.

V. Hướng phát triển trong tương lai:

IPSDN có thể được cải thiện theo các hướng chính sau:

- Phát triển từ cơ chế phát hiện các cuộc tấn công DoS/DDoS thành phát hiện các cuộc tấn công bất kỳ dựa vào sự kết hợp giữa phân tích các flow và phân tích nội dung gói tin (flow-based + packet-based).
- Mở rộng dataset từ việc thu thập các luồng hoạt động bình thường và bất thường lên sử dụng thêm KDD Cup.
- Sử dụng thêm công cụ tạo ra các traffic (từ scapy thành kết hợp scapy và hping3).
- Mở rộng việc chặn các traffic độc hại thành tùy chọn chặn hoặc chuyển hướng sang các honeypot.
- Nâng version của OpenFlow từ 1.3 lên 1.5, Python2 lên Python3, Snort2 lên Snort3 và Mininet2 lên Mininet3.

VI. Tài liệu tham khảo:

[1] Ryu controller, https://book.ryu-sdn.org/en/Ryubook.pdf
[2] Ryu controller, https://nsrc.org/workshops/2014/nznog-sdn/raw-attachment/wiki/WikiStart/Ryu.pdf
[3] OpenFlow version 1.3, https://opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.3.5.pdf
[4] Open vSwitch, https://docs.openvswitch.org/en/latest/intro/what-is-ovs/
[5] Snort rules, http://manual-snort-org.s3-website-us-east-1.amazonaws.com/node33.html#SECTION00469000000000000000
[6] Snort and snort rules, https://paginas.fe.up.pt/~mgi98020/pgr/writing_snort_rules.htm
[7] Snort, https://blog.snort.org/2011/09/flow-matters.html?fbclid=IwAR2H2xYoHz1CwVEgkOrMN3WAnSRgnk3bHIcXoyYeBl0rI0G5cLMOA6au6Ks
[8] Mininet, https://github.com/mininet/mininet/wiki/Documentation
[9] Fares, Awatef Ali Yousef R., et al. "DoS attack prevention on IPS SDN networks." 2019 Workshop on Communication Networks and Power Systems (WCNPS). IEEE, 2019.

[10] Nam, Kiho, and Keecheon Kim. "A study on sdn security enhancement using open source ids/ips suricata." 2018 International Conference on Information and Communication Technology Convergence (ICTC). IEEE, 2018.

[11] Ujjan, Raja Majid Ali, Zeeshan Pervez, and Keshav Dahal. "Suspicious traffic detection in SDN with collaborative techniques of snort and deep neural networks." 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS). IEEE, 2018.

[12] Manso, Pedro, José Moura, and Carlos Serrão. "SDN-based intrusion detection system for early detection and mitigation of DDoS attacks." Information 10.3 (2019): 106.

[13] Karan, B. V., D. G. Narayan, and P. S. Hiremath. "Detection of DDoS attacks in software defined networks." 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS). IEEE, 2018.

[14] Pratama, Rifqi Fauzan, Novian Anggis Suwastika, and Muhammad Arief Nugroho. "Design and implementation adaptive Intrusion Prevention System (IPS) for attack prevention in software-defined network (SDN) architecture." 2018 6th International Conference on Information and Communication Technology (ICoICT). IEEE, 2018.

HẾT