



دانشگاه صنعتی امیر کبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

پروژه کارشناسی

گرایش نرم افزار

طراحی و پیاده سازی vTAP برای مانیتورینگ ماشین های مجازی در
شبکه های مبتنی بر نرم افزار

نگارش

علیرضا آقایی

استاد راهنما

دکتر سیاوش خرسندی

اسفند 1399

با تشکر از دکتر خرسندی، استاد پروژه، که مرا در انجام این پروژه یاری نمودند.

چکیده

با تکامل شبکه‌های مبتنی بر نرم‌افزار و همچنین مجازی‌سازی کارکردهای شبکه، شرکت‌های ارتباطات راه دور و توسعه‌دهندگان سرویس‌های ابری به دنبال آن هستند تا کارکرد دستگاه‌های فیزیکی شبکه را در بستر مجازی‌شده ابری یا مبتنی بر نرم‌افزار پیاده‌سازی کنند.

هنگام بروز خطا در شبکه استفاده از مکانیزمی جهت بررسی وضعیت شبکه بصورت منظم، آن هم جهت شناسایی علل خرابی و در نتیجه بازسازی عملیات امری ضروری است. دستگاه فیزیکی TAP در شبکه، سخت‌افزاری است که چنین کاری را انجام میدهد. نسخه مجازی این دستگاه تحت عنوان virtual TAP یا همان vTAP شناخته می‌شود.

از جمله موارد استفاده vTAP می‌توان به رؤیت‌پذیری ترافیک محیط‌های مجازی، تعیین گلوگاه‌های سرعت، تشخیص ناهنجاری و تقویت دفاع در برابر حملات امنیتی اشاره کرد.

تاکنون نسخه‌های تجاری از محصول vTAP عرضه شده ولی بعلت در دست نبودن فریم‌ورک متن‌باز، در این پروژه به ایجاد و تست vTAP با استفاده از فریم‌ورک‌های متن‌باز مبادرت نموده‌ایم. کنترلر مورد استفاده در پروژه Ryu می‌باشد که مبتنی بر زبان برنامه‌نویسی پایتون است؛ نیز برای سوئیچ مجازی بین ماشین‌های مجازی از Open vSwitch استفاده شده که یک سوئیچ متن‌باز محسوب می‌گردد. برای این پروژه ابتدا یک سوئیچ یادگیرنده لایه 2 بررسی و با آن یک monitor ایجاد شده که توسط monitor عملکرد vTAP محقق شده است.

واژه‌های کلیدی: شبکه‌های مبتنی بر نرم‌افزار، مجازی‌سازی کارکردهای شبکه، vTAP، گلوگاه سرعت، Ryu، Open vSwitch

فهرست مطالب

صفحه

عنوان

ج	فهرست مطالب
ه	فهرست تصاویر و جداول
1	1 فصل اول: مقدمه
1	1.1 شبکه‌های نرم‌افزار محور
3	1.2 دستگاه فیزیکی TAP و مفهوم vTAP
4	1.3 هدف و اهمیت کار
4	1.4 ساختار پایان‌نامه
6	2 فصل دوم: مفاهیم پایه و ادبیات موضوع
6	2.1 مفاهیم اصلی در شبکه‌های نرم‌افزار محور
6	2.1.1 جداسازی صفحه داده از صفحه کنترل و مرکزیت کنترلر
7	2.1.2 محیط‌های مناسب برای بکارگیری شبکه‌های نرم‌افزار محور
8	2.1.3 زنجیره‌ای کردن سرویس‌های شبکه در شبکه‌های نرم‌افزار محور
9	2.1.4 تحولات از underlay های سنتی به underlay های شبکه نرم‌افزار محور
11	2.1.5 ریزدانگی در شبکه‌های نرم‌افزار محور
13	2.1.5.1 تعریف سرویس

14.....	2.1.5.2 پیکربندی سرویس
15.....	2.1.6 جمع‌بندی و نکات تکمیلی
18.....	2.2 پروتکل OpenFlow
19.....	2.2.1 تعریف تجرید در پروتکل OpenFlow
20.....	2.2.2 تجریدهای اصلی در پروتکل OpenFlow
20.....	Datapath 2.2.2.1
21.....	Port 2.2.2.2
23.....	Queues & Tables 2.2.2.3
24.....	Table 2.2.2.4
26.....	Flow 2.2.2.5
28.....	Match 2.2.2.6
34.....	Instruction 2.2.2.7
39	3 فصل سوم: معماری کلی سیستم
39	Hardware 3.1
40	Operating System (Ubuntu) 3.2
40	VirtualBox نرم‌افزار 3.3
40.....	3.3.1 انواع حالات شبکه کردن در VirtualBox
43.....	3.3.2 آماده‌سازی VirtualBox برای مانیتورینگ
50	3.4 معرفی OVS (Open vSwitch)
50.....	3.4.1 OVS درباره
50.....	3.4.2 OVS نصب
51.....	3.4.3 مؤلفه‌های کلیدی در OVS
53	3.5 mininet ابزار
55	3.6 معرفی کنترلر Ryu
55.....	3.6.1 نصب کنترلر Ryu
56	3.7 MySQL ، Web App و MySQL Workbench

4 فصل چهارم: بررسی سوئیچ یادگیرنده.....	57
4.1 بررسی و اجرای کد سوئیچ یادگیرنده.....	57
4.1.1 بررسی کد سوئیچ یادگیرنده	57
4.1.2 اجرای سوئیچ یادگیرنده در محیط mininet.....	65
5 فصل پنجم: پیاده‌سازی و اجرای MONITOR	70
5.1 بررسی و اجرای کد monitor.....	70
5.1.1 اجرا برای مانیتورینگ ترافیک ماشین‌های مجازی.....	74
5.1.2 اجرا در محیط mininet.....	80
5.2 جمع‌بندی.....	83
6 فصل ششم: کارهای آینده	84
7 منابع و مراجع	85

فهرست تصاویر و جداول

شکل 1 – شمای کلی شبکه نرم‌افزار محور	2
شکل 2 – TAP در شبکه	3
شکل 3 – مثالی از یک شبکه underlay.....	9
شکل 4 – مثالی از یک شبکه overlay.....	10
شکل 5 – لایه‌بندی صفحه کنترل در معماری شبکه‌های نرم‌افزار محور	12
شکل 6 – معماری شبکه نرم‌افزار محور در سطح تعریف سرویس	13
شکل 7 – معماری شبکه نرم‌افزار محور در سطح پیکربندی سرویس	14

- شکل 8 - شمای کلی از 4 ویژگی اصلی هر تجرید..... 19
- شکل 9 - طریقه کلی انتخاب جریان با توجه به بسته ورودی مطابقت شده..... 26
- شکل 10 - فرآیند کلی مطابقت 31
- شکل 11 - پشته طبقه‌بند برای OpenFlow 1.1..... 32
- شکل 12 - پشته طبقه‌بند برای OpenFlow 1.4..... 32
- شکل 13 - نمودار وابستگی طبقه‌بند برای OpenFlow 1.4..... 33
- شکل 14 - شکل پایلین صفحه داده جهت پردازش بسته‌ها..... 34
- شکل 15 - نحوه رمزگشایی و استخراج کلید 35
- شکل 16 - شکل کلی از مرحله اجرا 36
- شکل 17 - معماری کلی سیستم (بخش mininet برگرفته از [14]) 39
- شکل 18 - انواع حالات شبکه کردن در VirtualBox 43
- شکل 19 - نرم‌افزار VirtualBox پس از نصب ماشین‌های مجازی VM-A و VM-B 44
- شکل 20 - نسخه Ubuntu و کرنل لینوکس نصب شده در ماشین‌های مجازی VM-A و VM-B 45
- شکل 21 - پیکربندی اولیه سیستم (برگرفته از [18]) 46
- شکل 22 - عدم اتصال سوئیچ mybridge به اینترنت (برگرفته از [18]) 47
- شکل 23 - نحوه انتخاب واسط مجازی vport2 برای ماشین مجازی VM-B 48
- شکل 24 - پیکربندی سیستم پس از آماده‌سازی شبکه مد نظر برای انجام پروژه (برگرفته از [18]) 49
- شکل 25 - معماری کلی OVS 51
- شکل 26 - ماشین حالت سوئیچ در حالت اتصال اصلی 61
- شکل 27 - ماشین حالت کنترلر در حالت اتصال اصلی..... 62
- شکل 28 - نمودار وابستگی متغیرها در برنامه سوئیچ یادگیرنده 63
- شکل 29 - خروجی دستوری ovs-ofctl جهت مشاهده جدول جریان سوئیچ s1 66
- شکل 30 - جدول جریان سوئیچ s1 پس از اتصال موفقیت‌آمیز به کنترلر 66
- شکل 31 - جدول جریان سوئیچ s1 پس از انجام ping 67
- شکل 32 - خروجی دستور ifconfig برای استخراج آدرس IP ماشین مجازی VM-B 76
- شکل 33 - خروجی دستور ping و خاتمه آن توسط وقفه 77
- شکل 34 - صفحه وب مربوط به اطلاعات آماری جریان‌ها در محیط ماشین مجازی 78

- شکل 35 - صفحه وب مربوط به اطلاعات آماری پورت‌ها در محیط ماشین مجازی 79
- شکل 36 - جدول جریان سوئیچ mybridge 80
- شکل 37 - صفحه وب مربوط به اطلاعات آماری جریان‌ها در محیط mininet 81
- شکل 38 - صفحه وب مربوط به اطلاعات آماری پورت‌ها در محیط mininet 82

1 فصل اول: مقدمه

1.1 شبکه‌های نرم‌افزار محور

شبکه سنتی به شبکه‌ای اطلاق می‌گردد که در آن برای کنترل ترافیک از دستگاه‌های خاص و ثابتی همچون روتر و سوئیچ استفاده می‌شود. در شبکه‌های سنتی، صفحه کنترل (بخش مربوط به هندل کردن ترافیک عبوری از بخش‌های مختلف شبکه) و صفحه داده (بخش مربوط به ارسال داده مطابق سیاست‌گذاری‌های بخش کنترل) هر دو در یک دستگاه جمع شده بودند که معمولاً این دستگاه محصولی انحصاری بود که به شرکتی خاص (مثل HP، CISCO و غیره) تعلق داشت. مطلبی که معمولاً درباره شبکه‌های نرم‌افزار محور (معادلاً همان شبکه‌های مبتنی بر نرم‌افزار) گفته می‌شود این است که در آنها، صفحه کنترل از داده مجزا شده است ولی جداسازی صفحه داده از صفحه کنترل، آنچنان هم جدید نیست؛ چرا که مفهوم این جداسازی، پیش‌تر در شبکه تلفن نیز بعنوان راهی برای تأمین¹ و مدیریت استفاده می‌شد و سازمان IETF نیز یک واسط پیشنهادی تحت عنوان ForCES² برای جداسازی صفحات داده و کنترل پیشنهاد داده بود [1].

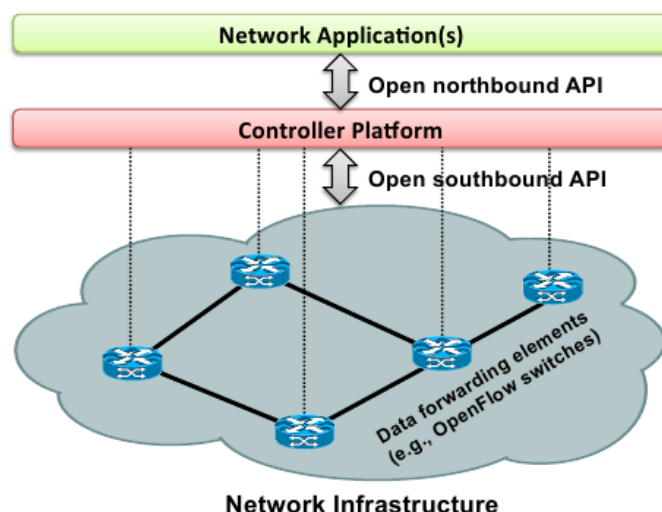
پس بهتر است شبکه نرم‌افزار محور را نه بعنوان راه‌حل جدید بلکه بعنوان یک نوع ساده‌سازی در راه‌حل‌های قبلی در نظر گرفت که با تعریف api دقیق‌تر و بهینه‌تر نسبت به api های پیشین، به ما امکان استفاده از سوئیچ‌های whitebox را می‌دهد. سوئیچ‌های whitebox ابزارهای جدیدی از شبکه هستند که دیگر حالت ویژگی³ و وابستگی به شرکت یا شخص خاصی ندارند. پس در شبکه‌های نرم‌افزار محور بعلت بهتر و دقیق‌تر بودن واسطه‌ها⁴، وظیفه ساختن واسط برای تجهیزات شبکه از انسان سلب شده و با ارائه دستگاه به همراه واسط شبکه نرم‌افزار محور خطای انسانی کاهش قابل ملاحظه‌ای می‌یابد. با توجه به نکات گفته شده، شبکه نرم‌افزار محور بیشتر یک اصطلاح در دنیای مارکتینگ است تا در دنیای مهندسی. تصویر زیر، شمای کلی از یک شبکه نرم‌افزار محور را نشان می‌دهد:

¹ Provisioning

² Forwarding and Control Element Separation

³ Peculiarity

⁴ Interfaces



شکل 1 - شمای کلی شبکه نرم‌افزار محور [11]

همانطور که در شکل نیز پیداست کنترلر به نوعی نقش مبدأ و مرکز را دارد و ستون اصلی ارتباطات در شبکه نرم‌افزار محور تلقی می‌گردد. ترافیک‌های عبوری از کنترلر به 3 دسته عمده تقسیم می‌شوند که 2 دسته آن در شکل 1 نیز دیده می‌شود.

- 1- ترافیک Northbound که بین کنترلر و لایه بالاتر یعنی لایه کاربرد وجود دارد و api آن، توابع سیستمی را در بر دارد که جهت ارتباط کنترلر با برنامه‌های کاربردی در لایه کاربرد استفاده می‌شوند.
- 2- ترافیک Southbound که بین کنترلر و لایه پایین‌تر شامل ادوات زیرساختی شبکه وجود دارد.
- 3- ترافیک East-West Bound (در شکل نشان داده نشده است) که در سناریوهای حاوی چند کنترلر برای ارتباط میان کنترلرها استفاده می‌شود.

در شبکه‌های نرم‌افزار محور دیگر نیازی به مدل‌های مختلف شرکت‌های متفاوت تأمین کننده سرویس‌های شبکه وجود ندارد چون با جداسازی صفحه کنترل از داده، به نوعی شبکه را قابل برنامه‌نویسی⁵ می‌کنیم که باعث می‌شود برنامه‌ها یک مدل مرجع داشته باشند که مستقل از شرکت (های) سازنده سوئیچ whitebox، منطق حاکم بر شبکه را تعیین می‌کند.

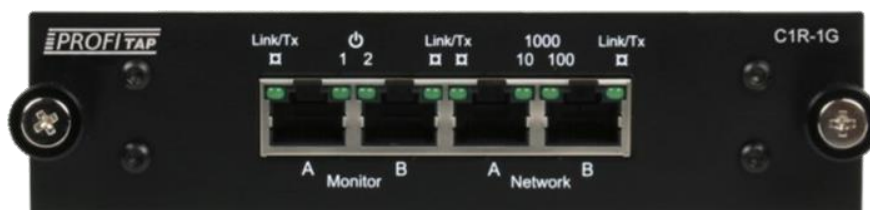
⁵ Programmable

برای شبکه‌های نرم‌افزار محور، پروتکل‌های مختلفی وجود دارند که از جمله آن‌ها می‌توان به ⁶NETCONF، ⁷OVSDB، ⁸OF-CONFIG، OpenFlow و ForCES اشاره کرد. در این پروژه، از پروتکل OpenFlow و واسط‌های مربوط به آن استفاده کرده‌ایم.

در فصل بعد، به تفصیل درباره مفاهیم شبکه‌های نرم‌افزار محور صحبت به میان رفته‌است. بنابراین در اینجا به همین شرح مختصر بسنده می‌کنیم.

1.2 دستگاه فیزیکی TAP و مفهوم vTAP

عبارت TAP مخفف Traffic Access Point یا Test Access Point می‌باشد و به سخت‌افزاری اطلاق می‌شود که در نقطه بخصوصی از شبکه مستقر می‌گردد تا ترافیک عبوری از آن نقطه قابل دسترسی به منظور تست و یا عیب‌یابی باشد. TAP در شبکه عمدتاً به منظور نظارت⁹ بر ترافیک شبکه استفاده می‌شود. شکل زیر یک دستگاه TAP را نشان می‌دهد:



شکل 2 – TAP در شبکه [2]

دستگاه TAP شکل بالا دو پورت مربوط به Network دارد که به دو نقطه در شبکه مانند A و B متصل می‌شوند که در اینجا قصد دسترسی و نظارت بر ترافیک عبوری بین آن دو نقطه را داریم. نیز دستگاه vTAP دو پورت مربوط به Monitor دارد که در آن، ترافیک جمع‌آوری شده (یک پورت برای سمت A به B و یک پورت دیگر برای سمت B به A) به دستگاه سومی تحت عنوان monitor فرستاده می‌شود که وظیفه نظارت بر شبکه را بر عهده دارد. توجه داشته باشید TAP و monitor دو دستگاه جدا هستند که در این میان، عملکرد TAP صرفاً فرستادن یک نسخه از ترافیک اصلی به monitor جهت نظارت بوده و عملکرد monitor، نظارت بر شبکه می‌باشد. این نظارت، در ساده‌ترین حالت (در پروژه همین حالت از monitor پیاده‌سازی شده است) صرفاً

⁶ Network Configuration Protocol

⁷ Open vSwitch Database Management

⁸ OpenFlow Configuration and Management Protocol

⁹ Monitor

گزارشی از وضعیت شبکه است که بصورت متناوب تولید می‌گردد. با اضافه کردن عملکردهای دیگری همچون هشدار، تشخیص فعالیت مخرب¹⁰ و تهیه نقشه توپولوژی شبکه می‌توان به حالت‌های پیچیده‌تری از monitor دست یافت.

در شبکه مجازی، به جای دستگاه‌های فیزیکی از ماشین‌های مجازی، کانتینرها و مانند آن‌ها استفاده می‌شود. بنابراین، در شبکه مجازی دیگر دستگاه فیزیکی TAP نمی‌تواند پاسخگوی نیاز شبکه به عملکرد TAP باشد چرا که TAP با اتصالات فیزیکی به رسانه انتقال اطلاعات کار می‌کند حال آنکه در محیط‌های مجازی دیگر اتصالات فیزیکی بدین شکل وجود ندارند. بدین منظور، از معادل مجازی آن با نام virtual TAP یا همان vTAP استفاده می‌شود که بنوعی پورت مجازی قابل شنود است. در شبکه مجازی، لزومی ندارد monitor یک دستگاه فیزیکی باشد. یک نرم‌افزار جمع‌آوری اطلاعات از vTAP که متناوباً گزارش شبکه را تولید می‌نماید نیز برای عملکرد monitor در شبکه مجازی کفایت می‌کند. در این پروژه، چنین نرم‌افزاری را توسط کنترلر Ryu و با زبان پایتون ایجاد کرده‌ایم. راهکار vTAP در دو سناریوی مختلف مورد آزمایش قرار گرفته است: یکی نرم‌افزار VirtualBox و دیگر نرم‌افزار mininet. بخش اصلی پروژه مانیتورینگ ترافیک ماشین‌های مجازی است که در VirtualBox و به کمک ماشین‌های مجازی صورت گرفته است. نرم‌افزار mininet صرفاً برای بیان نکات دقیق‌تر درباره صحت عملکرد vTAP بوده و به نوعی جنبه تکمیلی دارد.

1.3 هدف و اهمیت کار

مقوله شبکه‌های نرم‌افزار محور و همچنین مجازی‌سازی کارکردهای شبکه از طرفی مباحثی هستند که در چند سال گذشته مورد توجه روزافزون قرار گرفته‌اند. از طرفی دیگر راهکارهای vTAP ارائه شده تاکنون محصولات تجاری می‌باشند. هدف از پروژه، ایجاد یک راهکار vTAP به‌مراه نرم‌افزار جمع‌آوری اطلاعات با استفاده از فریم‌ورک‌های غیرتجاری و متن‌باز است که بتواند مانیتورینگ بین ماشین‌های مجازی را انجام دهد.

1.4 ساختار پایان‌نامه

در فصل دوم به طور مفصل به بحث درباره مفاهیم پایه می‌پردازیم. در فصل سوم معماری کلی سیستمی که monitor در آن اجرا شده و راهکار vTAP مورد بررسی قرار گرفته است. در فصل چهارم یک سوئیچ یادگیرنده

¹⁰ Malicious Activity

را بررسی می‌کنیم که کد پایتون آن از قبل بصورت آماده در source tree مربوط به محل نصب Ryu وجود دارد. فصل پنجم حاوی نکات و جزئیات پیاده‌سازی و اجرای monitor می‌باشد که در آن با استفاده از سوئیچ یادگیرنده مطرح شده در فصل چهارم، کد پایتون monitor مورد نظر در پروژه را ایجاد می‌کنیم و درستی آن را مورد آزمایش قرار می‌دهیم. نهایتاً در فصل ششم با برشمردن شماری از کارهای آینده بحث را به پایان می‌بریم.

2 فصل دوم: مفاهیم پایه و ادبیات موضوع

در این فصل قصد داریم بطور مفصل درباره مفاهیم اصلی در شبکه‌های نرم‌افزار محور صحبت کرده و وارد جزئیات مربوط به پروتکل OpenFlow شویم.

2.1 مفاهیم اصلی در شبکه‌های نرم‌افزار محور

2.1.1 جداسازی صفحه داده از صفحه کنترل و مرکزیت کنترلر

جداسازی صفحه داده از صفحه کنترل در شبکه‌های نرم‌افزار محور بدین معنا نیست که صفحه کنترل، به تمامی از سوئیچ‌های whitebox برداشته شود. هر قدر هم که کنترل ترافیک، از سوئیچ‌های whitebox به یک کنترلر مرکزی محول گردد باز هم هر سوئیچ whitebox، یک صفحه کنترل خیلی کوچک در دل خود خواهد داشت (برای اطلاعات بیشتر نگاه کنید به Switch Agent در جدول 2 – لایه‌های اصلی در معماری شبکه نرم‌افزار محور).

وجود یک کنترلر مرکزی در رأس صفحه کنترل شبکه نرم‌افزار محور، مانند یک سرور عام منظوره¹¹ است که یک فرآیند همیشه در جریان را اجرا می‌کند. کارهای کنترلر مرکزی عمدتاً شامل موارد زیر می‌باشد:

- کارهای proactive که شامل بارگذاری فایل‌های پیکربندی در سوئیچ‌های whitebox به محض اتصال به کنترلر می‌باشد.

- کارهای reactive که در واکنش به یک اتفاق¹² رخ میدهند مانند هندل کردن Flow Exception. صفحه داده متشکل از سخت‌افزارهای شبکه و پردازنده‌های آن سخت‌افزارهاست که در سوئیچ‌های whitebox قرار گرفته‌اند.

از این رو می‌توان شبکه نرم‌افزار محور را به مانند یک کامپیوتر دانست که در آن کنترلر حکم سیستم عامل را دارد؛ نیز برنامه‌های آن سیستم عامل، ارتباط کنترلر با سوئیچ‌های whitebox حاضر در شبکه نرم‌افزار محور را برقرار می‌کنند. بر مبنای همین مثال، اصطلاح NOS که مخفف Network Operating System می‌باشد شکل گرفته است.

¹¹ General Purpose

¹² Event

2.1.2 محیط‌های مناسب برای بکارگیری^{۱۳} شبکه‌های نرم‌افزار محور

ابتدا لازم است دو اصطلاح از دنیای توسعه نرم‌افزار را تعریف کنیم [3]:

- Greenfield Software Development: به معنای توسعه یک سیستم در محیطی کاملاً جدید است، آن هم بدون نسخه‌های قدیمی^{۱۴} و فارغ از محدودیت یا وابستگی.

- Brownfield Software Development: به معنای توسعه یا بکارگیری سیستم نرم‌افزاری جدید در حضور سیستم‌های موجود و قدیمی که معمولاً جهت ارتقای برنامه‌های قبلی انجام می‌شود.

این دو اصطلاح در دنیای شبکه نیز وجود دارند طوریکه به شبکه‌های ایجاد شده بر مبنای این روش‌ها شبکه‌های greenfield و brownfield گفته می‌شود.

برای شبکه‌های نرم‌افزار محور، شبکه‌های greenfield نسبت به brownfield محیط مناسب‌تری است چراکه ایجاد شبکه‌های brownfield معمولاً به پیچیدگی بیش از حد سناریوهای شبکه و پشتیبانی سازمان می‌انجامد. استقرار شبکه‌های نرم‌افزار محور به معنای از بین رفتن تمامی شبکه‌های سنتی نیست. همه چیز به این بستگی دارد که چه شبکه‌ای با چه نیازمندی‌هایی مد نظر است. برای بکارگیری و قرارگیری شبکه‌های نرم‌افزار محور در دنیای واقعی، بسته به این که چه شبکه‌ای مد نظر بوده 3 سناریوی احتمالی رخ می‌دهد:

1- همان شبکه سنتی موجود برای ایجاد شبکه مد نظر کفایت کند. در این حالت اصلاً نیازی به بکارگیری شبکه نرم‌افزار محور نخواهد بود.

2- هیچ شبکه سنتی جوابگوی نیازمندی‌های شبکه مد نظر نباشد و از همان اول لازم باشد شبکه تماماً نرم‌افزار محور طراحی گردد (greenfield deployment).

3- نیازمندی بخشی از شبکه مد نظر، با همان شبکه سنتی موجود برطرف شود و تنها برای بخش خاصی از شبکه مد نظر نیاز باشد شبکه نرم‌افزار محور ایجاد گردد. به چنین شبکه‌ای بعثت پشتیبانی همزمان از شبکه سنتی و شبکه نرم‌افزار محور شبکه ترکیبی^{۱۵} گفته می‌شود.

بعنوان مثال چنانچه شبکه‌ای مدنظر باشد که در آن مسیریابی بین دامنه‌ای^{۱۶} (IDR) با همان پروتکل سنتی BGP نیازمان را برطرف کند، شبکه سنتی موجود در سناریوی 1 جوابگوی ما خواهد بود. حال چنانچه IDR در شبکه مدنظر مستلزم استفاده از پروتکلی خاص و ویژه باشد که تا کنون در شبکه‌های سنتی نظیر آنرا نداشتیم،

¹³ Deployment

¹⁴ Legacy

¹⁵ Hybrid

¹⁶ Inter-Domain Routing

باید مطابق سناریوی 2 برای آن شبکه‌ای نرم‌افزار محور ایجاد کرد که در آن با استفاده از پروتکل‌های رایج شبکه‌های نرم‌افزار محور (مانند OpenFlow) پروتکل IDR دلخواه بصورت برنامه‌ریزی شده ایجاد شود. و نهایتاً چنانچه IDR بخشی از شبکه مد نظر، و نه تمام آن، مستلزم پروتکل جدید باشد، سناریوی 3 مورد استفاده قرار می‌گیرد که در آن فقط برای بخش جدید شبکه مد نظر از شبکه نرم‌افزار محور و پروتکل شبکه نرم‌افزار محور (مانند OpenFlow) استفاده می‌کنیم و IDR سایر بخش‌های شبکه مطابق سناریوی 1 با همان پروتکل سنتی BGP پیش خواهد رفت [4].

2.1.3 زنجیره‌ای کردن سرویس‌های شبکه^{۱۷} در شبکه‌های نرم‌افزار محور

منظور از زنجیره‌ای کردن سرویس‌های شبکه یک سرویس مدیریتی در شبکه است که زنجیره‌ای از سرویس‌های متصل به هم را - عمدتاً سرویس‌های مرتبط با لایه‌های 4 تا 7 نظیر دیواره آتش^{۱۸}، کنترلر تحویل به برنامه^{۱۹} و حفاظت از حمله^{۲۰} - در شبکه ایجاد می‌کند. هدف از این کار، مجازی‌سازی سرویس‌ها و اجرای آنها در منابع ذخیره‌سازی شبکه، و همچنین امکان ایجاد، حذف و یا مقیاس‌پذیر کردن سرویس‌ها در محیط‌های مجازی توسط مدیران شبکه می‌باشد [5].

مزیت اصلی این کار، ایجاد امکان اتوماسیون جهت تأمین ارتباطات و مدیریت جریان ترافیک در محیط‌های مجازی است. از جمله مزایای دیگر زنجیره‌ای کردن سرویس‌های شبکه می‌توان به انعطاف‌پذیری و مقرون بصرفگی برای مشاغل آینده در شبکه اشاره کرد.

مقوله مجازی‌سازی کارکردهای شبکه در کنار شبکه‌های نرم‌افزار محور، با حذف کردن نیاز به منابع سخت افزاری اضافی و انتقال مدیریت و کارکرد شبکه از سخت‌افزار شبکه (صفحه داده) به واحد مرکزی نرم‌افزاری (صفحه کنترل) از مشکلات بالقوه‌ای همچون تأمین بیش از اندازه^{۲۱} و یا تأخیر زمانی زیاد برای مدیریت و یا مشکلات ناشی از دخالت انسانی جلوگیری می‌کند.

به کمک شبکه‌های نرم‌افزار محور، برای هر ترافیک می‌توان بسته به نوع، آنرا از زنجیره سرویس‌های منحصربفرد برای همان ترافیک هدایت کرد. بعنوان مثال، پارامترهای تقاضا، پهنای باند، رمزنگاری و QoS را در نظر بگیرید

¹⁷ Network Service Chaining

¹⁸ Firewall

¹⁹ Application Delivery Controller

²⁰ Intrusion Protection

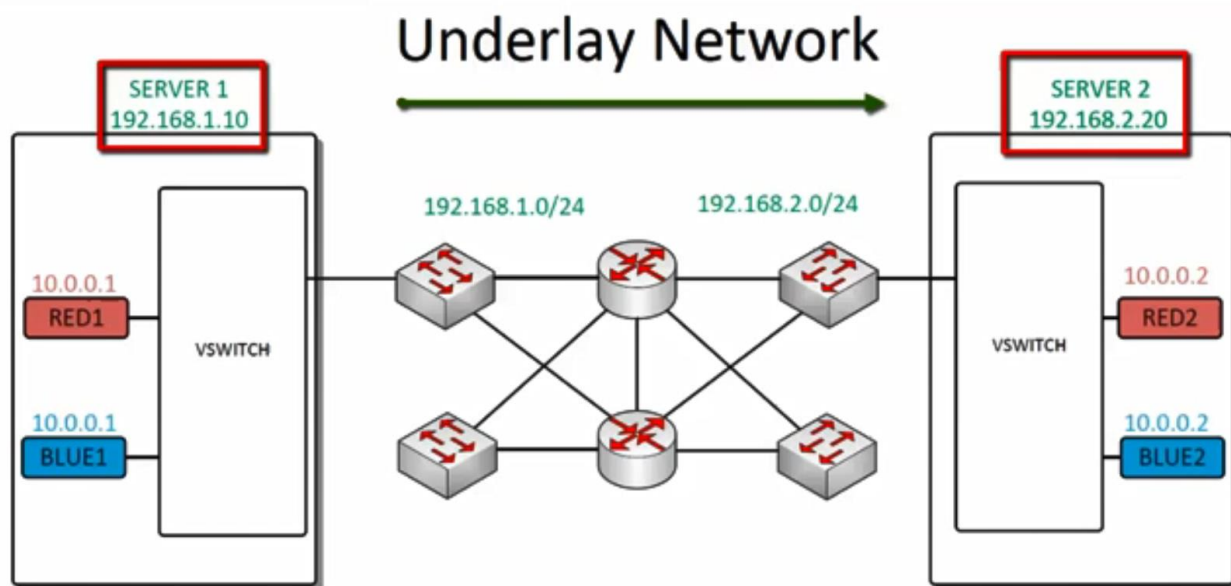
²¹ Overprovisioning

که برای هر نوع جریان منحصر بفرد است. این پارامترها برای جریان‌های ترافیک مربوط به VoIP²² نسبت به جریان‌های ترافیک مربوط به simple web access متفاوت میباشند. با عبور و هدایت هریک از جریان‌های VoIP و simple web access از زنجیره سرویس‌های منحصر بفرد خودشان، ضمانت می‌کنیم که پارامترهای هر یک از این دو جریان بطور جداگانه برآورده گردد [6].

2.1.4 تحولات از underlay های سنتی به underlay های شبکه نرم افزار محور

به زیرساخت فیزیکی یک شبکه اصطلاحاً underlay گفته می‌شود. از این رو شبکه overlay یک شبکه مجازی است که بر روی زیرساخت فیزیکی underlay بنا شده است گفته می‌شود. هدف از این کار پیاده‌سازی سرویس‌هایی از شبکه است که در شبکه فیزیکی وجود ندارند [7].

در شبکه‌های نرم افزار محور می‌توان شبکه مجازی overlay را بر روی شبکه underlay ایجاد نمود. برای توضیح ملموس تر ارتباط شبکه نرم افزار محور با شبکه‌های overlay و underlay از مثال شکل زیر استفاده می‌کنیم:

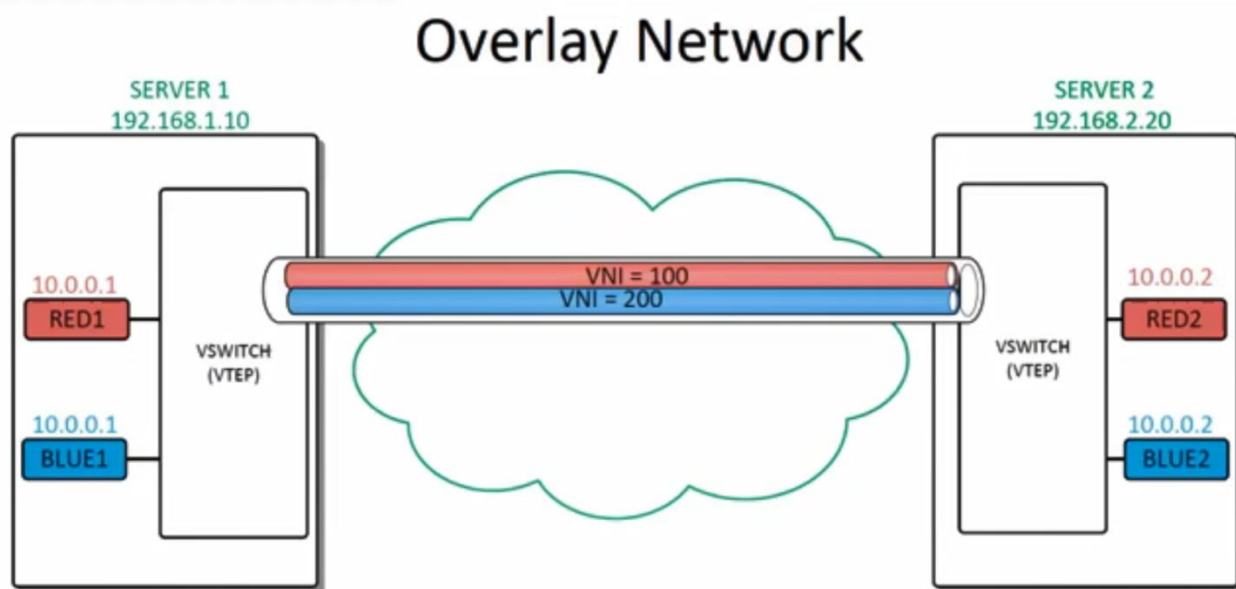


شکل 3 - مثالی از یک شبکه underlay [8]

فرض کنید در محیط ابری بالا، یک کاربر به نام RED ماشین‌های مجازی RED1 و RED2 را مطابق شکل فوق اجاره کرده است که روی دو سرور مختلف با نام‌های SERVER1 و SERVER2 قرار دارند. مشابه کاربر

²² Voice over IP

دیگری به نام BLUE ماشین‌های مجازی BLUE1 و BLUE2 را روی دو سرور مختلف اجاره کرده است. شبکه underlay صرفاً آدرس‌های IP مربوط به SERVER1 (192.168.1.10) در زیرشبکه 1 (192.168.1.0/24) و نیز SERVER2 (192.168.2.20) در زیرشبکه 2 (192.168.2.0/24) را می‌شناسد. به عبارتی دیگر شبکه underlay از آدرس‌های IP مربوط به ماشین‌های مجازی RED1,2 و BLUE1,2 کاملاً بی‌اطلاع است. در نتیجه آدرس‌های IP آن‌ها را (که برای هر دو کاربر RED و BLUE همان 10.0.0.1 و 10.0.0.2 می‌باشد) نمی‌شناسد. اکنون برای اتصال ماشین‌های مجازی RED1 و RED2 (یا BLUE1 و BLUE2) از شبکه overlay کمک گرفته می‌شود که مطابق شکل زیر نشان داده شده است:



شکل 4 - مثالی از یک شبکه overlay [8]

در شبکه overlay، سوئیچ مجازی به یک نقطه انتهایی در شبکه مجازی تبدیل می‌شود که در شبکه LAN توسعه یافته مجازی (VXLAN²³) به آن VTEP گفته می‌شود. در شکل بالا VTEP1 و VTEP2 در دو انتهای شبکه overlay قرار گرفته‌اند که به ترتیب در سمت SERVER1 و SERVER2 قرار دارند. هر VTEP مسئول کپسوله‌سازی فریم‌های لایه 2 اترنت در سرآیند VXLAN جهت فرستادن به شبکه در سطح لایه 3 می‌باشد.

²³ Virtual Extensible LAN

اکنون انجام این 2 کار برقراری ارتباط بین ماشین‌های مجازی RED1 و RED2 (یا BLUE1 و BLUE2) را تکمیل خواهد نمود:

- 1- اطلاع هر VTEP از نگاشت آدرس IP ماشین مجازی به آدرس IP سرور واقع در زیرشبکه آن VTEP
 - 2- ایجاد ساز و کاری جهت تفکیک منطقی ترافیک ماشین‌های مجازی RED از BLUE چراکه از دید کاربران RED و BLUE در شبکه overlay آدرس‌های IP یکسانی دارند. برای این کار سرآیند VNI در VXLAN در نظر گرفته شده است. همانگونه که در شکل میبینید برای ترافیک RED، مقدار VNI برابر 100 است که آنرا از ترافیک BLUE با $VNI=200$ متمایز می‌سازد.
- یکی از راه‌های هندل کردن این نگاشت، نگهداری جدول نگاشت در کنترلر شبکه نرم‌افزار محور است که امکان نگهداری و بروزرسانی آنرا بطور اتوماسیون شده فراهم می‌کند.
- بعلاوه، در پروتکل OpenFlow، بحث tunneling وجود ندارد و می‌توان سیاست‌گذاری را طوری انجام داد که بسته‌ها و جریان‌ها ظاهر بیرونی یکپارچه‌ای در لایه 2 داشته باشند.
- جدول زیر بطور کلی به برخی تفاوت underlay های موجود در شبکه های نرم‌افزار محور و شبکه‌های سنتی اشاره می‌کند:

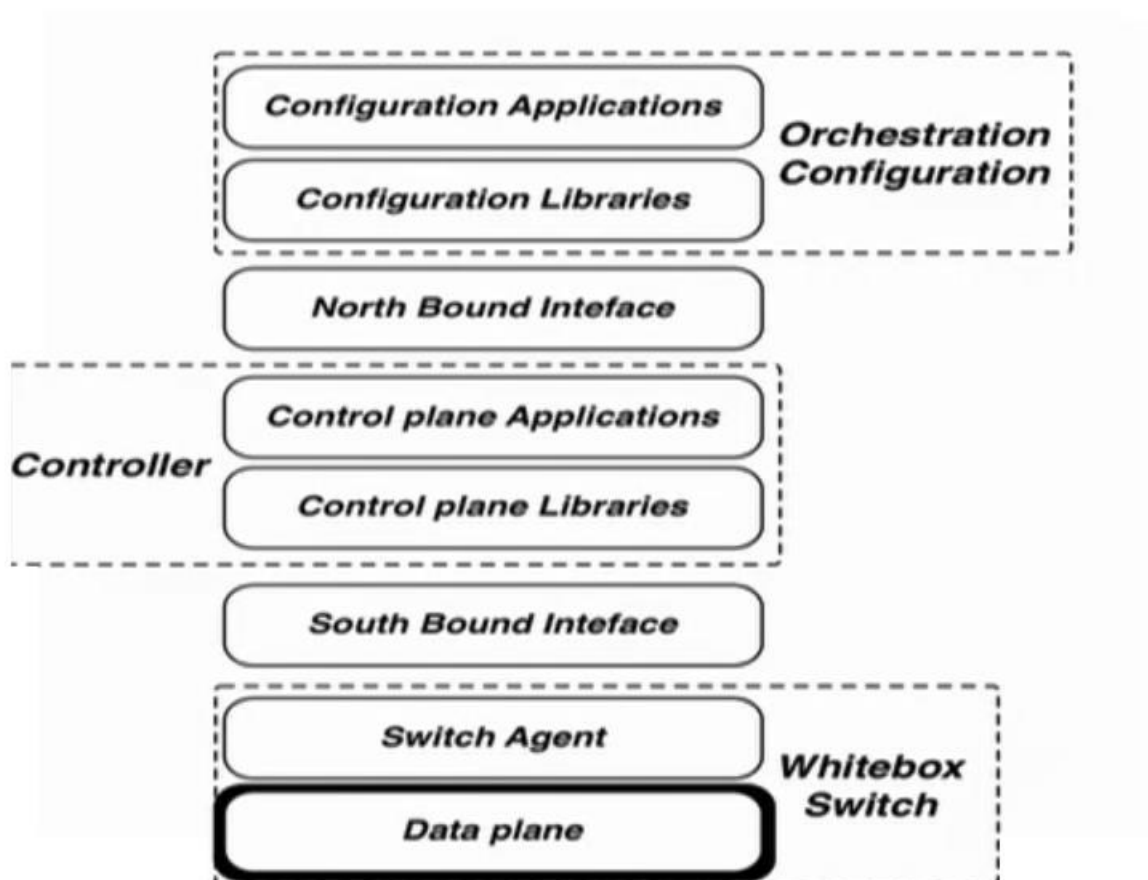
Underlay های موجود در شبکه‌های سنتی	Underlay های موجود در شبکه‌های نرم‌افزار محور	
پروتکل‌های Ethernet، VLAN و پروتکل‌های مسیریابی مثل OSPF، BGP و غیره	پروتکل SDN مانند OpenFlow	پروتکل
بورد ASIC سفارشی جهت اجرای هزاران RFC	بورد ASIC موجود در بازار، بمراتب ارزان‌تر نسبت به ASIC سفارشی	سخت‌افزار

جدول 1 – تفاوت underlay های موجود در شبکه‌های سنتی و شبکه‌های نرم‌افزار محور

2.1.5 ریزدانگی در شبکه‌های نرم‌افزار محور

تا بدین جا دانستیم ایده کلی شبکه نرم‌افزار محور، نوشتن برنامه‌ای است برای کنترلر(های) شبکه نرم‌افزار محور که قابلیت یکپارچه شدن و سازگاری با محیط‌های مختلف با وسایل متفاوت را دارد. برنامه‌ای که واسط آن یک زبان برنامه نویسی (مثل جاوا، پایتون و غیره) است که در عین داشتن قابلیت‌های بسیار، عاری از هرگونه ابهام می‌باشد.

پیش از این، کنترلر را صرفاً در رأس صفحه کنترل میدانستیم و از آن بیشتر درباره ساختار صفحه کنترل صحبت نکرده بودیم. صفحه کنترل، خود دارای لایه‌های گوناگونی است که هر یک از این لایه‌ها، با سطح بخصوصی از اهمیت^{۲۴} و عملکرد^{۲۵} شبکه متناظر است. شکل زیر بطور دقیق‌تری لایه‌بندی صفحه کنترل را نشان می‌دهد:



شکل 5 - لایه‌بندی صفحه کنترل در معماری شبکه‌های نرم‌افزار محور [4]

همانطور که از شکل پیداست 3 لایه اصلی می‌توان برای صفحه کنترل در نظر گرفت:

1- لایه Orchestration Configuration

²⁴ Concern

²⁵ Functionality

2- لایه Controller

3- لایه Whitebox Switch

برای بررسی بیشتر این 3 لایه لازم است ابتدا 2 اصطلاح در رابطه با ریزدانگی و مقیاس در شبکه‌های نرم‌افزار محور بیان شود. بصورت کلی 2 رویکرد با ریزدانگی متفاوت در شبکه‌های نرم‌افزار محور به چشم می‌خورد:

1- تعریف سرویس^{۲۶}

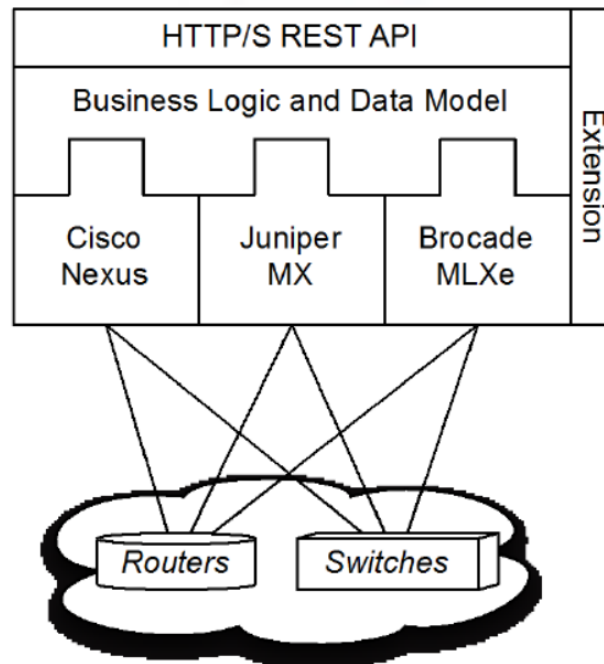
2- پیکربندی سرویس^{۲۷}

در ادامه هر یک از این دو در زیربخش جداگانه‌ای بررسی شده‌اند.

2.1.5.1 تعریف سرویس

عبارتست از تعاریفی که به تغییر رفتار یک دستگاه می‌انجامد.

شکل زیر معماری شبکه نرم‌افزار محور در سطح تعریف سرویس را نشان می‌دهد:



شکل 6 – معماری شبکه نرم‌افزار محور در سطح تعریف سرویس [9]

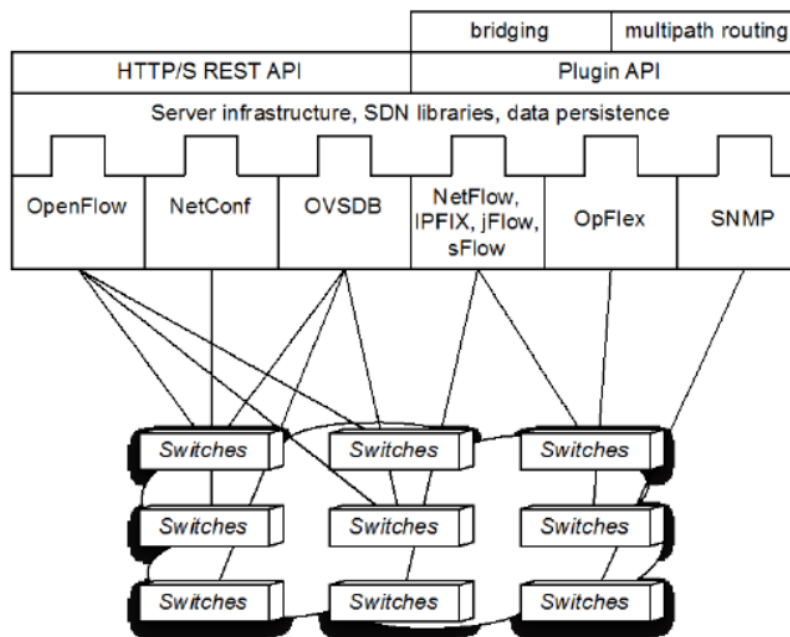
²⁶ Service Definition

²⁷ Service Configuration

REST²⁸، مجموعه‌ای از اصول طراحی است که با آن‌ها از HTTP بعنوان یک زبان بدون حالت²⁹ برای فراخوانی رویه از راه دور (RPC³⁰) استفاده می‌شود. REST api یکی از متداول‌ترین api ها برای ترافیک northbound می‌باشد. در لایه میانی Data Model و Business Logic هستند که منطق برنامه را در خود جای می‌دهند. منطق برنامه می‌تواند عملکردی همچون BGP، VPN و یا هر عملکرد دیگری باشد. و نهایتاً در لایه زیرین، plugin هایی همچون Cisco Nexus، Juniper MX و Brocade MLXe قرار دارند که عملکردهای لایه میانی را به سخت‌افزار انتقال می‌دهند.

2.1.5.2 پیکربندی سرویس

عبارتست از تأمین یک واسط اتوماسیون شده برای مجموعه دستگاه‌های موجود در شبکه. شکل زیر معماری شبکه نرم‌افزار محور در سطح پیکربندی سرویس را نشان می‌دهد:



شکل 7 – معماری شبکه نرم‌افزار محور در سطح پیکربندی سرویس [9]

در این سطح از ریزدانگی برنامه‌های کاربردی را به طرق زیر می‌توان اجرا نمود:

²⁸ Representational State Transfer

²⁹ Stateless

³⁰ Remote Procedure Call

- از طریق REST api و در خارج از کنترلر
- از طریق plugin در خود کنترلر

لایه میانی نیاز به توضیح خاصی ندارد و در لایه زیرین نیز پروتکل‌های شبکه نرم‌افزار محور قرار دارند که بسته به نیاز و کاربرد هر یک از آن‌ها می‌توانند در شبکه‌های نرم‌افزار محور استفاده شوند. در این پروژه، پروتکل OpenFlow را انتخاب کرده‌ایم که در ادامه همین فصل به توضیحات مفصل درباره آن خواهیم پرداخت.

2.1.6 جمع‌بندی و نکات تکمیلی

همانطور که از اسم مفاهیم نیز پیداست ریزدانگی در سطح تعریف سرویس بیشتر از سطح پیکربندی سرویس می‌باشد و سطح تعریف سرویس جزئیات بیشتری در خود دارد. با داشتن ریزدانگی و مفاهیم مجرد^{۳۱} در سطح تعریف سرویس می‌توان به سطح پیکربندی سرویس رسید ولی باید دقت داشت عکس این مطلب برقرار نیست. شبکه نرم‌افزار محور بیشتر یک استراتژی برای پیکربندی سرویس است تا یک استراتژی برای تعریف آن. پس قرار نیست زیرساخت‌ها را جهت استفاده از بستر شبکه نرم‌افزار محور تغییر دهیم. بعنوان مثال نمی‌توان یک Load Balancer را به یک سوئیچ یا روتر تبدیل کرد. نکته‌ای که ذکر آن در اینجا اهمیت دارد این است که در شبکه نرم‌افزار محور یا شبکه با مجازی سازی کارکردها، عناصر مجرد^{۳۲} دیگر وصله یک دستگاه فیزیکی خاص نیستند و خود وجودی مستقل دارند. درباره مفاهیم مجرد در پروتکل OpenFlow در ادامه در همین فصل بطور مبسوط سخن به میان رفته است.

با توجه به آنچه گفته شد اکنون می‌توان مطالب بیشتری درباره شکل 4 گفت. هر یک از دو سطح تعریف سرویس و پیکربندی سرویس برای خود یک کنترلر دارد که وظایف کلی این دو کنترلر مشابه یکدیگر می‌باشد. تفاوت کنترلرهای این دو سطح صرفاً در ریزدانگی و مقیاس کارهایی است که این کنترلرها به عهده دارند.

لایه orchestration configuration مربوط به همان کنترلر سطح پیکربندی سرویس می‌باشد.

لایه controller نیز مربوط به همان کنترلر سطح تعریف سرویس می‌باشد.

³¹ Abstraction

³² Abstract Elements

جدول زیر به شرح بیشتر هر یک از لایه‌های شکل 4 پرداخته است:

نام لایه	تعریف	شرح وظایف و ویژگی‌های لایه
Dataplane	بخش سریع ولی نه چندان هوشمند دستگاه که به سرعت و دقت سرنوشت بسته‌های جریان ترافیک شبکه را تعیین می‌کند.	تجریدهای مطرح شده در این لایه: Header classification, header modification, output handling, flow statistics, flow metering
Switch Agent	قسمتی از صفحه داده که با صفحه کنترل در ارتباط است، در حکم مغز کوچکی برای صفحه داده که بتواند فرامین صفحه کنترل را اجرا کند.	<ul style="list-style-type: none"> ترجمه فرامین OpenFlow به دستورهای سطح پایین نکته: صفحه داده ترجمه فرامین OpenFlow را اجرا می‌کند نه خود فرامین را. گزارش رخدادهای صفحه داده به SBI هندل کردن Dataplane offload شامل: <ul style="list-style-type: none"> فیچرهای پشتیبانی نشده در پروتکل OpenFlow فیچرهای آزمایشی^{۳۳}
South Bound Interface (SBI)	پروتکل احراز هویت ^{۳۴} ، مجوز دسترسی ^{۳۵} و حسابرسی ^{۳۶} محسوب می‌شود.	<ul style="list-style-type: none"> مذاکره نسخه^{۳۷} شناسایی ارتباطات از کار افتاده کشف قابلیت‌های^{۳۸} سوئیچ whitebox ارسال پرس و جو به Switch Agent و دریافت رخدادهای آسنکرون (Exceptions, Unexpected Messages) از

³³ Experimental Features

³⁴ Authentication

³⁵ Authorization

³⁶ Accounting

³⁷ Version Negotiation

³⁸ Capability Discovery

Switch Agent • صفحه کنترل توزیع شده		
• نرمالسازی انحرافات در SBI: • هندل کردن پروتکل‌های SBI • مختلف بعثت حضور دستگاه‌های متفاوت از شرکت‌ها یا خط تولیدهای مختلف در زیرشبکه‌های ناهمگون ³⁹ از شبکه نرم‌افزار محور • ارائه مدل داده واحد • ارائه کتابخانه‌های آماده برای برنامه‌های کاربردی	وظیفه مدیریت سوئیچ‌های whitebox را بعده دارد.	Control Plane Libraries
تعدادی از برنامه‌ها: • Ethernet Bridge • IP Router • Application Load Balancer • Firewall • NAT/PAT	به برنامه‌های کاربردی گفته می‌شود که در مجازی‌سازی کارکردهای شبکه یا شبکه‌های نرم‌افزار محور به کار می‌روند.	Control Plane Applications
ایجاد REST api	دقیقا مانند SBI پروتکلی از شبکه نرم‌افزار محور می‌باشد که وظیفه پیکربندی برنامه‌های کاربردی را بعده دارد.	North Bound Interface (NBI)
تخصیص منابعی از قبیل آدرس IP، تگ‌های VLAN، ظرفیت پورت‌ها در سوئیچ whitebox و غیره	مدل‌های رایج بکارگیری شبکه نرم‌افزار محور است که مواردی همچون دامنه‌های همه‌پخشی لایه 2 ⁴⁰ ، افزونه VPN و مسیریابی بین دامنه‌ای را شامل می‌شود.	Configuration Libraries
عملیات رایج در این لایه: • نصب و حذف نصب سرویس • شروع یا متوقف کردن سرویس	شامل مدل‌های قرارگیری و بکارگیری متناسب با dataset مشتری که پیکربندی و واسط اتوماسیون و کتابخانه‌های	Configuration Applications

³⁹ Heterogeneous⁴⁰ Layer 2 Broadcast Domains

• تغییر پارامترهای سرویس	Openstack را راه اندازی می کنند.	
• بروزرسانی سرویس		

جدول 2 - لایه های اصلی در معماری شبکه نرم افزار محور [4]

برخی از مفاهیم بکار رفته در جدول نیازمند توضیحات بیشتری است که در ذیل به آن ها می پردازیم:

- نرمالسازی انحرافات در SBI به کمک ارائه کتابخانه های آماده: باید دقت داشت در محیطی که دستگاه ها همگی از یک شرکت هستند مشکل ناهمگونی وجود ندارد و نرمالسازی در چنین شرایطی صرفاً یک هزینه اضافی است که نفعی به همراه نخواهد داشت. در شبکه های ناهمگون، نویسنده برنامه کاربردی در شبکه نرم افزار محور نباید راجع به دستگاه مقصد نگران باشد چون اساساً فلسفه شبکه نرم افزار محور این است که واسط (زبان برنامه نویسی) از صفحه داده مستقل و مجرد باشد. در چنین شرایطی از کتابخانه های آماده استفاده می گردد که کار آن، مخفی کردن جزئیات مربوط به پروتکل SBI از برنامه نویسی شبکه نرم افزار محور می باشد.

- درباره مشابهت پروتکل NBI و SBI: هر دوی آنها پروتکل هایی هستند که وظایفی همچون احراز هویت، مجوز دسترسی و حسابرسی را بعهده دارند. تفاوت آن ها در مقیاس و ریزدانه گی شان هست که در NBI سطح ریزدانه گی برای پیکربندی سرویس و در SBI برای تعریف سرویس می باشد.

- Openstack یک فریم ورک برای کنترل کردن مخازن بزرگ محاسبات، ذخیره سازی و منابع شبکه می باشد که از طریق NBI و api مربوط به آن، به کنترلر ها متصل شده و کار پیکربندی سرویس را انجام می دهد [10].

در شبکه های نرم افزار محور نوعی مصالحه⁴¹ بین سرعت و کارایی صفحه داده وجود دارد. پیاده سازی صفحه داده هر چه بیشتر متمایل به نرم افزار باشد، شاهد کارایی بالاتر، سرعت پایین تر و انعطاف پذیری بالاتری خواهیم بود. در طرف مقابل، متمایل بودن صفحه داده به سخت افزار، آنرا به کارایی پایین تر، سرعت بالاتر در عین انعطاف پذیری پایین تر سوق خواهد داد.

2.2 پروتکل OpenFlow

پروتکل OpenFlow یک پروتکل TLV⁴² می باشد که در آن هر مؤلفه از پیام، ابتدا با فیلد type آغاز می شود که در ادامه آن فیلدهای length و value قرار دارند.

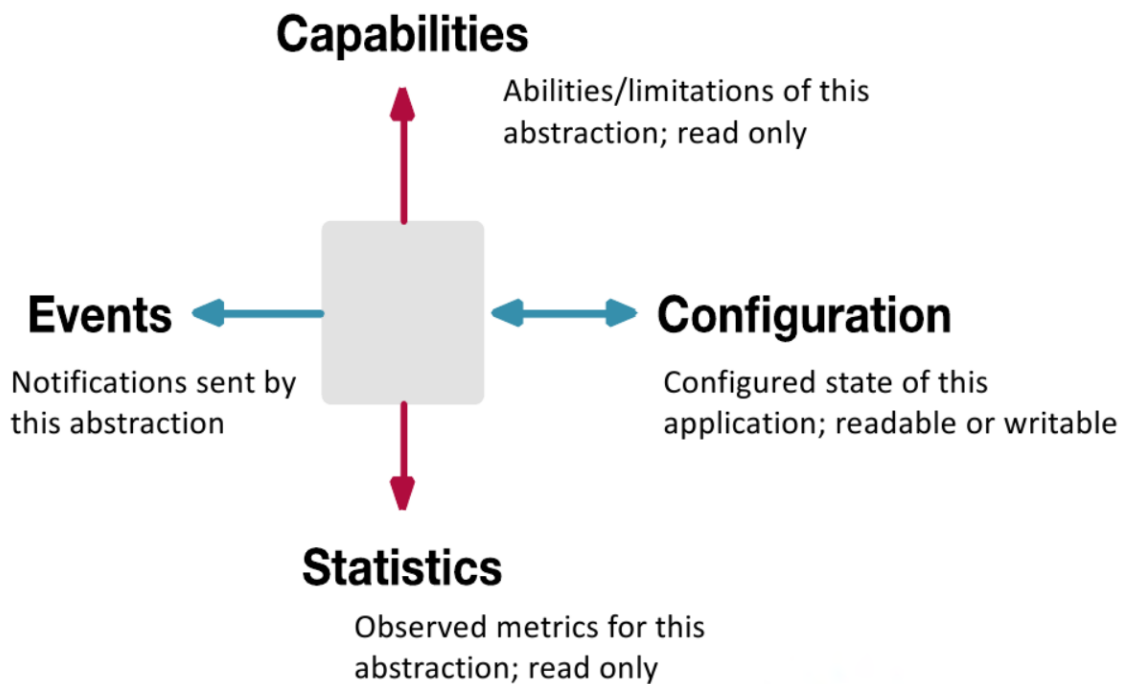
پروتکل OpenFlow معمولاً روی TCP اجرا می شود و یک اتصال TLS نیز جهت امنیت برقرار می شود.

⁴¹ Trade-off⁴² Type Length Value

در این بخش قصد داریم به پردازش بسته‌ها در پروتکل OpenFlow بصورت مجرد نگاهی داشته باشیم تا بر مبنای آن، بتوانیم برنامه monitor را پیاده‌سازی کنیم. عملگرهای اصلی در OpenFlow، کنترلر و switch agent هستند. کنترلر، یک سرور واقعی، و یا یک نرم‌افزار عام منظوره می‌باشد.

2.2.1 تعریف تجرید در پروتکل OpenFlow

هر تجرید در OpenFlow با 4 ویژگی شناخته می‌شود. شکل زیر شمایی کلی از ویژگی‌های تجرید نشان می‌دهد. در ادامه هر یک را توضیح خواهیم داد:



شکل 8 - شمای کلی از 4 ویژگی اصلی هر تجرید

همانطور که از شکل نیز پیداست، این 4 ویژگی عبارتند از:

- 1- قابلیت‌ها (Capabilities): برنامه‌ای که کنترلر OpenFlow را اجرا می‌کند نیاز دارد بداند آن کنترلر دقیقاً دارای چه قابلیت‌هایی است. هر تجرید، باید این توانایی برشمردن قابلیت‌هایش را داشته باشد تا آن را بتوان بعنوان یک تجرید به شمار آورد. و باید بتوان از یک تجرید این سؤال را پرسید که دقیقاً

چه قابلیت‌هایی دارد. با دانستن قابلیت‌های یک تجرید و همچنین قابلیت‌های کنترلر OpenFlow می‌توان دریافت که آیا آن تجرید روی آن کنترلر قابل پیاده‌سازی و اجراست یا خیر.

2- پیکربندی (Configuration): پیکربندی می‌تواند برای یک سناریوی کم تکرار مثل پیکربندی صف⁴³ باشد که کمتر دستخوش تغییر و پیکربندی می‌گردد. و یا می‌تواند برای یک سناریوی پرتکرار مانند پیکربندی وضعیت پورت باشد که شامل مواردی همچون up/down بودن، half/full duplex، نوع بسته‌های فوروارده شده و غیره شود.

3- آمار (Statistics): حضور این ویژگی در تجرید اختیاری است و هر تجریدی الزام به ارائه آمار ندارد.

4- رخداد (Event): حضور این ویژگی نیز اختیاری است و هر تجریدی الزاماً یک رخداد تولید نمی‌کند. رخداد، معمولاً مکانیزمی برای مطلع کردن صفحه داده از ناتوانی در هندل کردن یک امر خاص می‌باشد.

اکنون که با تجرید کمی بیشتر آشنا شدیم، لازم است چند نمونه از تجریدهای مهم در پروتکل OpenFlow را مورد بررسی قرار دهیم. شناخت این تجریدها، جهت درک درست مطالب بیان شده در فصول 3 و 4 ضروری می‌باشد.

2.2.2 تجریدهای اصلی در پروتکل OpenFlow

تجریدهای اصلی در پروتکل OpenFlow که ما در اینجا قصد بررسی آنها را داریم عبارتند از:

- 1- Datapath
- 2- Port
- 3- Queues & Tables
- 4- Flow
- 5- Match
- 6- Instruction

در زیربخش‌های آتی هر یک از این تجریدها بطور جداگانه شرح داده شده است.

Datapath 2.2.2.1

Datapath تجریدی نیست که بطور مکرر مورد تغییر قرار بگیرد. از جمله ویژگی سراسری datapath می‌توان به موارد زیر اشاره کرد:

⁴³ Queue

- dpid به طول 64 بیت که شناسه datapath می‌باشد و آن را بطور یکتا از سایرین متمایز می‌کند.
- Fragmentation Handling
- Packet Buffer: بسته‌هایی که هنوز در موردشان تصمیم‌گیری نشده است در این بافر قرار می‌گیرند و تا معلوم شدن پاسخ نهایی کنترلر در همین بافر منتظر می‌مانند.

Port 2.2.2.2

ویژگی‌های اصلی پورت عبارتند از:

- Id
- نام پورت (مثال: Eth1 برای پورت شماره 1 اترنت)
- آدرس MAC
- موارد زیر در یک پورت قابل پیکربندی است:
- حالت administrative: که می‌تواند up/down باشد که اصطلاحاً آن پورت را administratively up/down می‌گویند.
- رفتار پورت:

- این که فوروارد کردن بسته روی این پورت مجاز نباشد.
 - این که دریافت کردن بسته روی این پورت مجاز نباشد.
 - این که مجاز نباشد این پورت، ارور Packet In را از کنترلر دریافت کند.
- وضعیت پورت یکی از مواردی است که در شبکه‌های ترکیبی در صورت استفاده از Spanning Tree مشخص می‌شود:

- Link/carrier
- Blocked state

قابلیت‌های پورت را می‌توان به 4 گروه دسته‌بندی کرد:

- Supported: قابلیت‌هایی که این پورت از آن‌ها پشتیبانی می‌کند.
- Advertised: قابلیت‌هایی که این پورت، آن‌ها را به سایر پورت‌ها advertise می‌کند.
- Peer: قابلیت‌هایی که یک پورت دیگر آن‌ها را به این پورت advertise کرده است.
- Current: قابلیت‌هایی که در مرحله مذاکره^{۴۴} بر سر آن‌ها توافق حاصل شده.

⁴⁴ Negotiation

بصورت خاص، قابلیت مهم پورت‌ها Current Speed و Max Speed می‌باشد که به ترتیب سرعت فعلی و بیشینه سرعت ترافیک عبوری از آن پورت را مشخص می‌کنند.

در نسخه‌های اخیر OpenFlow، قابلیت‌های فیبر نوری از قبیل تغییر فرکانس و طول موج، و همچنین تغییر قدرت سیگنال نیز به لیست قابلیت‌های موجود اضافه شده است.

پورت‌های مجازی⁴⁵، به مجموعه خاصی از پورت‌ها گفته می‌شود که رزرو شده هستند. آنها معنای خاصی دارند و برنامه‌نویس مجاز به استفاده از این پورت‌ها نیست. جدول زیر شامل تعدادی از این پورت‌ها به همراه معنای استفاده از آنها می‌باشد:

معنا	نام پورت مجازی
بسته دریافتی از این پورت روی همین پورت ارسال می‌شود.	Ingress
روی بسته‌های عبوری از این پورت پردازش packet in/out انجام می‌شود.	Table
ترافیک عبوری از این پورت، بجای عبور از پایپلاین OpenFlow از پایپلاین شبکه‌های سنتی رد می‌شود. کاربرد: در شبکه‌های ترکیبی	Normal
بسته، بسته مدیریتی است که پورت local باید آنرا بگیرد و در خود ماشین آنرا پردازش کند.	Local (local delivery)
ضمانت می‌شود بسته‌های عبوری از این پورت به کنترلر تحویل داده شوند.	Controller
به ازای هر بسته دریافتی روی این پورت، یک کپی از این بسته به تمامی پورت‌ها بغیر از پورت دریافت‌کننده این بسته ارسال می‌گردد.	All
به ازای هر بسته دریافتی روی این پورت، یک کپی از این بسته به تمامی پورت‌ها بغیر از پورت‌های در وضعیت blocked ارسال می‌گردد.	Flood

جدول 3 - توضیح برخی از پورتهای مجازی [4]

⁴⁵ Virtual Ports

Queues & Tables 2.2.2.3

در پروتکل OpenFlow پیکربندی خاصی برای صف‌ها در نظر گرفته نشده و آنرا خارج از OpenFlow هم می‌توان پیکربندی کرد. چراکه اساساً پارامترهای زیادی برای تغییر نداشته و تغییر چندانی روی آن انجام نمی‌شود.

از داخل صفحه داده OpenFlow، یک صف فقط بعنوان target قابل تنظیم است. آدرس‌دهی صف به ازای پورت‌ها انجام می‌شود. عبارتی بسته عبوری از یک پورت را می‌توان به صف خاصی از آن پورت فرستاد و یک بسته را به خودی خود نمی‌توان به صف خاصی ارسال کرد. اکنون وارد بحث جداول (Tables) می‌شویم که از گستره مطالب بیشتری نسبت به صف‌ها برخوردار است. جداول بطن اصلی پردازش در OpenFlow هستند. در پروتکل OpenFlow، کلکسیون‌های جداول جریان وجود دارد. هر جدول از این جداول، رفتارها و قابلیت‌های منحصر بفرد خود را دارد. این جداول در پروتکل OpenFlow پیمایش می‌شوند و نحوه پیمایش آن‌ها نیز بصورت خطی و بر اساس افزایش id جداول می‌باشد. علت این کار جلوگیری از بروز حلقه^{۴۶} جهت اطمینان از پردازش بسته‌ها در زمان متناهی می‌باشد. در سیستمی متشکل از چندین جدول، می‌توان یک زنجیره استدلال بین جداول برقرار نمود به گونه‌ای که هر جدول از آن زنجیره در صورت عدم موفقیت در انجام پردازش، کار پردازش را به جدول بعد از خود در این زنجیره بسپارد. علت اصلی داشتن چند جدول یا همان جداول، به جای یک جدول، تقسیم بندی رفتارها و فرآیندهای تفکر^{۴۷} می‌باشد. با ذکر مثالی درباره پروتکل اترنت این مطلب را بررسی می‌کنیم. در پروتکل اترنت، 3 نوع پردازش انجام می‌شود:

1- یادگیری

2- فوروارد

3- هرس کردن جدول سوئیچینگ

در مرحله یادگیری سئوالات حول منبع می‌باشند، حال آنکه در مرحله فوروارد سئوالات پیرامون مقصد هستند. پر واضح است پردازش‌های یادگیری و فوروارد ماهیت متفاوتی دارند که به رفتارها و فرآیندهای تفکر متفاوتی منجر می‌شود. حال چنانچه تعداد رکوردهای مربوط به یادگیری را با a و تعداد رکوردهای مربوط به فوروارد را با b نشان دهیم، در صورت استفاده از جدول یکسان برای یادگیری و فوروارد، تعداد مدخل‌های^{۴۸} این جدول a.b

⁴⁶ Loop

⁴⁷ Thought Processes

⁴⁸ Entries

خواهد بود؛ حال آنکه در صورت استفاده از 2 جدول متفاوت جمع تعداد مدخل‌های هر دو جدول $a + b$ خواهد شد. جهت جلوگیری از افزایش نجومی و نمایی پیچیدگی زمانی و حافظه بهتر است از دو جدول استفاده شود [4].

جریان‌ها اعضای اصلی تشکیل‌دهنده هر جدول می‌باشند که در زیربخش بعد آن‌ها را تعریف کرده مورد بررسی قرار خواهیم داد.

Table 2.2.2.4

در زیربخش قبل جداول را در ارتباط با یکدیگر و بصورت کلی بررسی کردیم. این زیربخش حاوی مطالبی درباره عملکرد هر جدول است که نشان می‌دهد هر جدول به تنهایی چه کاری را و چگونه انجام می‌دهد. عنصر اصلی تشکیل‌دهنده هر جدول، جریان نام دارد. جریان به توالی بسته‌ها بین یک مبدأ و یک مقصد گفته می‌شود که برای رفتار صفحه داده در قبال همه آن بسته‌ها سیاست‌گذاری واحدی اعمال می‌گردد. با این حساب در تعریف جریان، مجموعه مقادیر فیلدهای بسته داده به همراه معیار مطابقت (فیلتر) و مجموعه دستورالعمل‌های مورد انجام بروی آن بسته داده همگی دخالت دارند [11].

هر جدول دارای نام (name) و شناسه (id) است؛ و کلکسیونی از جریان‌هاست که بر اساس اولویت‌شان سازمان‌دهی شده‌اند. هر جریان، یک ردیف از ردیف‌های جدول است.

هنگامی که یک بسته با هیچ مدخلی از مدخل‌های یک جدول (جریان‌ها یا همان ردیف‌های جدول) مطابقت پیدا نکند یک miss اتفاق می‌افتد. نحوه هندل کردن miss در نسخه‌های مختلف OpenFlow متفاوت است. در OpenFlow 1.0، رخداد miss به تولید ارور در صفحه داده منجر می‌شد که باید آنرا در برنامه هندل می‌کردیم.

در OpenFlow 1.1, 1.2 دیگر ارور نداریم و بجای آن 3 گزینه پیش روی ماست:

1- خودمان ارور را تولید کرده و به کنترلر ارسال کنیم.

2- بسته را دور بیندازیم⁴⁹.

3- به جدول بعدی برویم.

در OpenFlow ≥ 1.3 ، در انتهای هر جدول یک جریان miss قرار می‌گیرد که برای آن، مطابقت با تمامی بسته‌ها تعریف شده است. پس چنانچه بسته‌ای با هیچ یک از جریان‌های قبل از جریان miss مطابقت پیدا نکند

⁴⁹ Drop

با خود جریان miss مطابقت داده شده و دستورالعمل‌های جریان miss روی آن انجام می‌گیرد. عبارتی با قرار دادن جریان miss در انتهای جدول، از بروز رخداد miss جلوگیری می‌کنیم چون هر بسته قطعاً با یکی از جریان‌های جدول (که این جریان یا قبل از جریان miss و یا خود جریان miss است) مطابقت پیدا می‌کند. پیش‌تر گفتیم جریان‌ها به نوعی مدخل‌ها و ردیف‌های جدول هستند. از این سو گاهی به جداول، جداول جریان نیز گفته می‌شود. باید دقت داشت تعداد جریان‌های هر جدول، محدودیتی دارد و چنانچه تعداد جریان‌های جدول به آن حد مشخص شده برسد جدول پر شده و اصطلاحاً exhaustion رخ می‌دهد. با رخ دادن exhaustion در یک جدول تلاش برای ایجاد جریان‌های بعدی در آن جدول با مشکل مواجه خواهد شد. در OpenFlow 1.1-1.3 این امر به تولید ارور می‌انجامد. در $\text{OpenFlow} > 1.3$ این امکان به برنامه‌نویس داده شده که استراتژی اخراج را پیش بگیرد. این استراتژی، در استانداردها موجود نیست و پیاده‌سازی این مورد بعهد خود برنامه‌نویس است. در این استراتژی، جدول برای ایجاد جریان‌های جدید، جریان‌های دیگری را با استفاده از روشی (به دلخواه برنامه‌نویس) از جدول حذف کرده اخراج می‌نماید و با این کار، جا را برای استقرار جریان‌های جدید در جدول فراهم می‌کند.

علاوه بر محدودیت جریان، محدودیت‌های دیگری در مطابقت، دستورالعمل‌ها و اعمال وجود دارند. برای مطابقت نمی‌توان روی هر تعداد دلخواهی از فیلدها عمل مطابقت را انجام داد. محدودیت فیزیکی حافظه‌های TCAM^{50} و SRAM^{51} که پیاده‌سازی منطقی و فیزیکی جداول در آن‌ها انجام می‌گیرد ما را وادار به تجدید نظر در نوع و تعداد فیلدهای مورد مطابقت می‌نماید. چرا که مطابقت در سطح منطقی و فیزیکی، بصورت بیتی انجام می‌شود و حافظه‌ها در تعداد بیت‌های مورد مطابقت محدودیت دارند.

هر فیلد نسبت به عمل مطابقت در یکی از دسته‌های زیر قرار می‌گیرد:

- Wildcard: با فیلد متناظر تمامی بسته‌ها مطابقت داده می‌شود.
- Enabled: قرار گرفتن یا نگرفتن این فیلد در مطابقت بسته‌ها تعیین می‌شود.
- Maskable: امکان استفاده از عملگرهای بیتی روی برخی از بیت‌ها یا همه بیت‌های فیلد [12] را در عمل مطابقت تعیین می‌کند.

⁵⁰ Ternary Content-Addressable Memory

⁵¹ Static Random-Access Memory

Flow 2.2.2.5

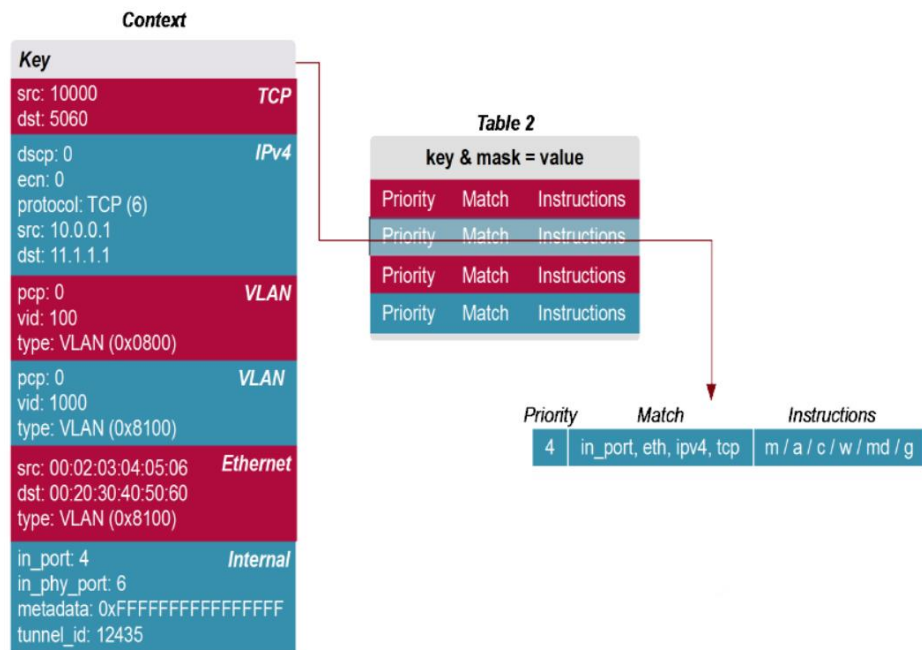
هر ردیف یا مدخل از جدول، یک جریان (flow) را برایمان مشخص می‌کند. در زیربخش Table، به ناچار برای توضیح بخشی از رفتارهای جدول نیازمند تعریف جریان بودیم. در اینجا، به بیان دیگر ویژگی‌های جریان می‌پردازیم.

ویژگی‌های جریان بصورتی است که بتواند بسته‌ها را هندل کند.

ویژگی‌های اصلی یک جریان عبارتند از:

- اولویت: جهت مرتب‌سازی جریان‌ها در جدول.
- مجموعه مطابقت
- مجموعه دستورالعمل‌ها

این موارد در شکل زیر قابل مشاهده است:



شکل 9 - طریقه کلی انتخاب جریان با توجه به بسته ورودی مطابقت شده [4]

تا بدین جا درباره اولویت به ارائه توضیحات پرداخته شد و دانستیم که جریان‌ها به ترتیب اولویت بررسی می‌شوند تا اولین جریان مطابقت داده شده انتخاب گردد. در اینجا از ارائه توضیحات درباره مطابقت و دستورالعمل خودداری می‌کنیم چرا که درباره هر یک از این دو بطور مشروح در دو زیربخش بعدی صحبت به میان رفته است.

هر جریان مدت زمان مشخصی در جدول نگهداری می‌شود. با منقضی شدن آن، یکی از رفتارهای زیر انجام می‌گیرد:

- رفتار بر مبنای زمان: بدینصورت که به ازای ایجاد هر جریان و یا هربار فعالیت یک جریان (منظور از فعالیت بروز مطابقت برای آن جریان است بطوریکه یک بسته عبوری از شبکه، با آن جریان از جدول مطابقت پیدا کند) یک تایمر برای آن جریان ایجاد می‌کنیم. در صورت منقضی شدن تایمر، آن جریان را از جدول جریان حذف می‌نماییم.
 - رفتار رخدادگرا^{۵۲}: این که یک جریان در جدول باقی بماند تا موقعی که با حذف، از جدول جریان برداشته شود. در این حالت به ازای هر حذف صورت گرفته، باید با ایجاد یک رخداد حذف جریان، کنترلر را باخبر کنیم تا در خصوص حذف آن جریان، اقدامات لازم را انجام دهد.
- یکی از مشکلاتی که به وفور در شبکه‌های نرم‌افزار محور رخ می‌دهد قطع ارتباط بین جداول جریان (موجود در صفحه داده) و کنترلر است که در نسخه‌های مختلف OpenFlow به طرق متفاوتی با آن برخورد می‌گردد:
- در OpenFlow 1.0 می‌توان برای یک جریان برچسب اضطراری^{۵۳} تعریف نمود. بنابراین پس از قطع ارتباط کنترلر با جداول، تمامی جریان‌های غیر اضطراری از جداول جریان حذف می‌شوند. جریان‌های اضطراری تا برقراری مجدد اتصال جداول با کنترلر، در جداول جریان باقی می‌مانند؛ نیز پس از اتصال مجدد، تصمیم‌گیری درباره جریان‌های اضطراری بعهد کنترلر و برنامه‌های کاربردی خواهد بود.
 - در OpenFlow ≥ 1.1 دو حالت جدید معرفی شده‌اند که هر یک رفتار متفاوتی در قبال قطع اتصال کنترلر با جداول جریان دارد:
 - Fail Standalone Mode: در این حالت قطع اتصال کنترلر با جداول جریان نوعی اختلال در پروتکل OpenFlow تلقی می‌شود. از این رو تا برطرف شدن اختلال و خرابی، از بررسی ترافیک در پروتکل OpenFlow اجتناب می‌گردد. در عوض، با فرستادن ترافیک به پورت normal، ترافیک در همان پروتکل سنتی شبکه‌های سنتی بررسی می‌گردد.
 - Fail Secure Mode: در این حالت از اجرای تمامی اعمالی که باعث ارسال ترافیک به کنترلر می‌شدند جلوگیری می‌شود.

⁵² Event Driven

⁵³ Emergency

Match 2.2.2.6

به جرأت می‌توان گفت یکی از مهم‌ترین تجربدهایی که در پروتکل OpenFlow بررسی می‌شود همین تجرید مطابقت (Match) می‌باشد.

انواع مطابقت را می‌توان به صورت‌های مختلفی دسته‌بندی کرد که یکی از آنها، مطابقت بر اساس فیلدها و مقادیر است که بدین صورت دسته‌بندی می‌شود:

- مطابقت بر اساس مقدار که انواع اصلی آن عبارتند از:

○ Protocol Field: مقدار یکی از فیلدهای پروتکل OpenFlow است که در بسته حضور دارد (مثال: آدرس IP مبدأ).

○ Internal Value: مقداری که در یک داده جانبی^{۵۴} یا رجیستر جانبی^{۵۵} اصطلاحاً در متن بسته^{۵۶} وجود دارد. این اطلاعات جانبی، با گذر بسته و پردازش پایپلاین روی آن ایجاد می‌شود. بعنوان مثال in_port یا پورت ورودی که بسته از طریق آن وارد سوئیچ whitebox می‌گردد، در خود بسته وجود ندارد ولی با ورود بسته به سوئیچ whitebox این اطلاعات جانبی نیز برای آن بسته نگهداری می‌شود. در این مثال in_port فیلدی نیست که در بسته وجود داشته باشد و داده جانبی محسوب می‌شود. مطالب بیشتر درباره پایپلاین در ادامه در همین زیربخش گفته شده است. بخشی دیگر از این اطلاعات بر رجیسترهای جانبی ثبت می‌شود که در ادامه توضیح بیشتری برای آن به میان رفته است.

پیش از این گفته بودیم که یکی از کارهایی که در صورت بروز miss (یا hit شدن با جریان miss در OpenFlow ≥ 1.3) انجام می‌شود محول کردن پردازش بسته به جدولی دیگر است. در همین راستا رجیستری با نام metadata وجود دارد که علاوه بر خود بسته و به همراه آن، بین جداول گردش می‌کند. لازم بذکر است هر جدول، فقط توانایی تغییر بخش‌های (بیت‌های) خاصی از این رجیستر را دارد و لزوماً به کل رجیستر metadata دسترسی ندارد.

از آنچه گفته شد می‌توان دریافت علاوه بر خود بسته، اطلاعات دیگری نیز در مرتبط با بسته وجود دارند که آن‌ها نیز در جدول مورد پردازش قرار می‌گیرند. بخشی از این اطلاعات در رجیسترهای جانبی ذخیره می‌شود که یکی از مهم‌ترین آن‌ها رجیستر metadata می‌باشد.

⁵⁴ Side Data

⁵⁵ Side Register

⁵⁶ Packet Context

صورت دیگر دسته‌بندی مطابقت‌ها بر اساس صراحت است:

- مطابقت صریح: که در آن مقدار مشخص شده جهت مطابقت صراحتاً بیان می‌شود.
- مطابقت mask شده یا wild-card شده: مطابقت بر اساس wildcard یا mask صورت می‌پذیرد که قطعاً صراحت مطابقت صریح را ندارد. تعریف wildcard و mask در زیربخش مربوط به تجرید Table انجام شده است.

Match Set: به مجموعه‌ای از مطابقت‌ها گفته می‌شود. علت اهمیت آن، به خاطر معضل وابستگی^{۵۷} است که از ضعف‌های OpenFlow محسوب می‌شود. یکی از ضعف‌های OpenFlow این است که در ظاهر از مطابقت با انواع مختلفی از پروتکل‌ها پشتیبانی می‌کند ولی در عمل، چیزی که واقعاً توسط پروتکل OpenFlow پشتیبانی می‌شود یک زیرمجموعه از پروتکل‌هاست. بعنوان مثال مطابقت با Ethernet & IP، مطابقت با Ethernet & ARP در پروتکل OpenFlow پشتیبانی می‌شود ولی ممکن است مطابقت با Ethernet & IP & UDP در یک نسخه مورد استفاده از پروتکل OpenFlow پشتیبانی نگردد. به همین خاطر، محتمل است برنامه‌نویس، از مطابقت‌هایی استفاده کند که نامعتبر^{۵۸} باشند.

در مجموعه مطابقت، مطابقت‌ها بر مبنای یک ترتیب انجام می‌شوند. منظور از ترتیب، آنست که هر مجموعه مطابقت را می‌توان مسیری از مطابقت‌ها دانست که برای معتبر بودن باید معضل وابستگی را نداشته باشند. معضل وابستگی وقتی پیش می‌آید که در مطابقت $X \text{ (field = Y)}$ که در آن فیلد Y از پروتکل X مورد مطابقت قرار گرفته، بررسی مطابقت فیلد Y از پروتکل X در حالی انجام شود که خود پروتکل X در شبکه حاضر نباشد. در این حالت شبکه به پروتکل X وابسته است ولی بعلمت در اختیار نداشتن پروتکل X، مطابقت فیلد Y از این پروتکل یک مطابقت نامعتبر خواهد بود. مطابقت، وقتی معتبر خواهد شد که قبل از اعمال مطابقت روی فیلد Y، در مسیر منتهی به مطابقت فیلد Y، از حضور پروتکل X در شبکه اطمینان حاصل کرده باشیم که بدین طریق معضل وابستگی از میان برود.

مثال زیر یک مثال از مطابقت معتبر^{۵۹} را نشان می‌دهد [4]:

-in_port=4, eth(type=0x0806), arp(tpa=10.2.3.4)

⁵⁷ Dependency

⁵⁸ Invalid

⁵⁹ Valid Match

دقت کنید ترتیب مطابقت از چپ بر راست است. این مطابقت یک مطابقت معتبر است بدین خاطر که پیش از مطابقت `arp(tpa=10.2.3.4)` در مسیر منتهی به آن، از حضور پروتکل ARP در شبکه اطمینان حاصل کرده ایم. این حصول اطمینان توسط مطابقت `eth(type=0x0806)` انجام شده و پروتکل اترنت (eth) نیز از جمله پروتکل‌های مورد پشتیبانی OpenFlow می‌باشد که باعث می‌شود مطابقت فیلد type از آن پروتکل مطابقتی امن⁶⁰ محسوب گردد.

مثال زیر نیز یک مثال از مطابقت نامعتبر را نشان می‌دهد [4]:

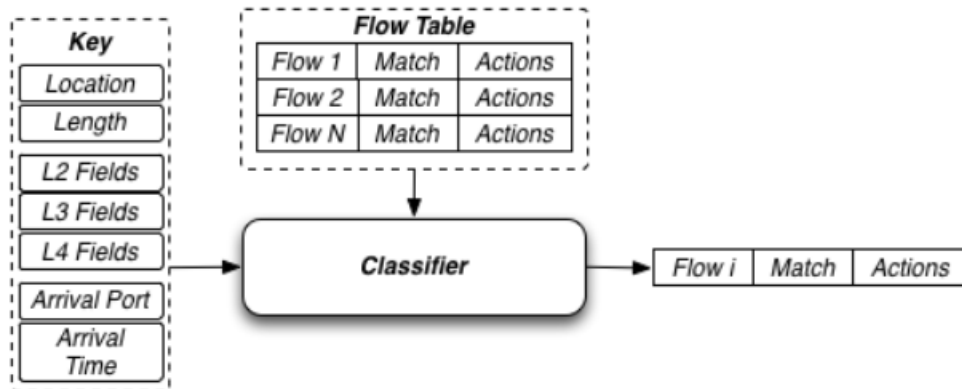
`-eth(type=0x800), ipv4(proto=1), tcp(dst=5060)`

درست است که مطابقت `eth(type=0x800)` برایمان حضور `ipv4` در شبکه را تأیید می‌کند، ولی نه این مطابقت نه مطابقت `ipv4(proto=1)` هیچکدام حضور TCP یا UDP را در شبکه بررسی نمی‌کنند. مطابقت `ipv4(proto=1)` صرفاً حضور پروتکل ICMP را در شبکه بررسی می‌کند. بررسی حضور TCP در شبکه با مطابقت `ipv4(proto=6)` و بررسی حضور UDP در شبکه با مطابقت `ipv4(proto=17)` انجام می‌شود. بنابراین مطابقت `tcp(dst=5060)` بدون اطلاع از حضور یا عدم حضور TCP در شبکه انجام شده که این مطابقت را نامعتبر می‌کند.

نحوه طراحی پروتکل‌های شبکه سنتی به گونه‌ای است که چنانچه پروتکل A از لایه پایین‌تر، بسته‌ای از پروتکل B از یک لایه بالاتر را حمل کند این امر در سرآیند مربوطه در پروتکل A مشخص شده است. همانگونه که در مثال‌های بالا نیز گفته شد، شناسه Ethertype (type) از پروتکل اترنت در لایه 2 مشخص می‌کند داده کدام پروتکل لایه 3 توسط اترنت حمل می‌شود. مشابهاً شناسه Protocol (proto) از پروتکل IPv4 در لایه 3 مشخص کننده آن پروتکل لایه 4 است که داده‌اش توسط پروتکل IPv4 حمل می‌گردد.

⁶⁰ Safe

شکل زیر بطور کلی بیانگر فرآیند کلی یک مطابقت می‌باشد:



شکل 10 - فرآیند کلی مطابقت [13]

اکنون به توضیح آنچه در شکل آمده می‌پردازیم. بخش اصلی مطابقت را طبقه‌بند (Classifier) تشکیل می‌دهد. هر جریان در جدول جریان، شامل یک طبقه‌بند است. خروجی طبقه‌بند، اولین جریانی از جدول (دقت کنیم پیمایش جدول به ترتیب اولویت جریان‌ها انجام می‌گیرد) است که طبقه‌بند آن با بسته همخوانی داشته باشد. به چنین مدخل جریانی که سبب بروز مطابقت شده یک مدخل جریان فعال^{۶۱} برای آن بسته گفته می‌شود. چرا که جریان فعال، حاوی اعمالی است که برای آن بسته فعال بوده و روی آن اجرا می‌شوند. کلید (Key) به ساختمان داده‌ای منظم^{۶۲} گفته می‌شود که از متن بسته استخراج شده و برای طبقه‌بند نقش شاخص^{۶۳} را بازی می‌کند. البته این نظم، از دید برنامه‌نویس است و در داخل سیستم، کلید به شکل رشته‌ای از 0 و 1 نگاه می‌شود. هدف از مطابقت، نوعی همخوانی این رشته 0 و 1 با طبقه‌بند می‌باشد. لازم به ذکر است چنانچه کار پردازش بسته بعلت عدم مطابقت یا هر علت دیگری، به جدول جریان دیگری محول شود، در صورت تغییر متن بسته باید از استخراج مجدد کلید بروزرسانی شده مطابق متن جدید اطمینان حاصل نمود.

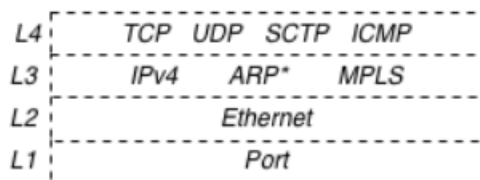
پیش‌تر گفتیم در پروتکل OpenFlow معضل وابستگی وجود دارد و باید حتماً مجموعه مطابقت، یک مطابقت معتبر باشد تا از بروز اشکالات ناخواسته در کنترلر جلوگیری شود. علت چنین امری، عدم پشتیبانی احتمالی آن نسخه پروتکل OpenFlow از پروتکل‌ها و یا فیلدهای مورد استفاده در مطابقت می‌باشد. لازم بذکر است در هر نسخه جدید OpenFlow دامنه تعداد پروتکل‌ها و تعداد فیلدهای مورد پشتیبانی توسط این پروتکل گسترده‌تر می‌گردد.

⁶¹ Active Flow Entry

⁶² Regular

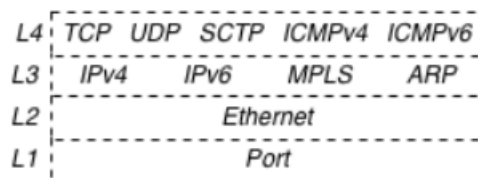
⁶³ Index

شکل زیر، پشته طبقه‌بند^{۶۴} را برای OpenFlow 1.1 نشان می‌دهد که در آن پشته پروتکل مورد پشتیبانی در این نسخه از OpenFlow را مشاهده می‌کنید:



شکل 11 - پشته طبقه‌بند برای OpenFlow 1.1

شکل زیر، پشته طبقه‌بند برای OpenFlow 1.4 را نشان می‌دهد که در آن پشته پروتکل مورد پشتیبانی گسترش قابل ملاحظه‌ای نسبت به پشته طبقه‌بند OpenFlow 1.1 داشته است:

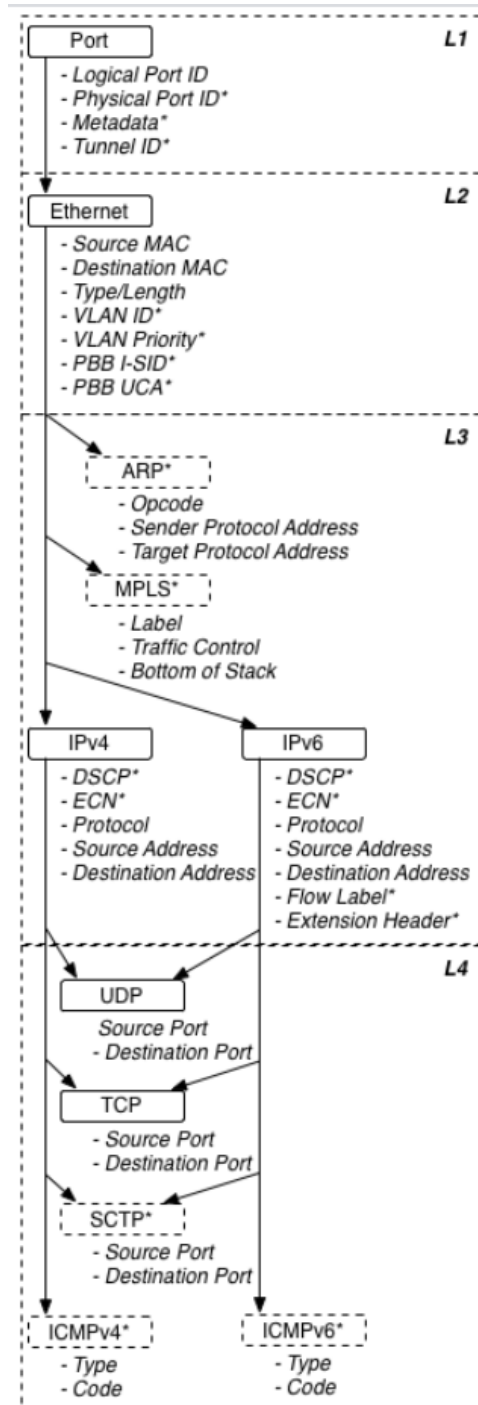


شکل 12 - پشته طبقه‌بند برای OpenFlow 1.4

در هر نسخه از OpenFlow برخی از طبقه‌بندها اختیاری هستند یعنی ممکن است بخشی یا تمام فیلدهای یک پروتکل در OpenFlow آن نسخه غیرقابل پشتیبانی باشد. مثلاً در OpenFlow 1.4، پشتیبانی از پروتکل ARP اختیاری می‌باشد. طبقه‌بندهای مورد پشتیبانی توسط پیام FlowMod در سوئیچ‌های whitebox نصب می‌شوند. آگاهی از پشتیبانی یک سوئیچ whitebox نسبت به طبقه‌بندهای اختیاری در پیام Table StatsRes که وضعیت هر جدول جریان را به کنترلر اطلاع می‌دهد مشخص می‌گردد.

⁶⁴ Classifier Stack

شکل زیر، نمودار وابستگی طبقه‌بند^{۶۵} را برای OpenFlow 1.4 نشان می‌دهد:



شکل 13 – نمودار وابستگی طبقه‌بند برای OpenFlow 1.4

⁶⁵ Classifier Dependency

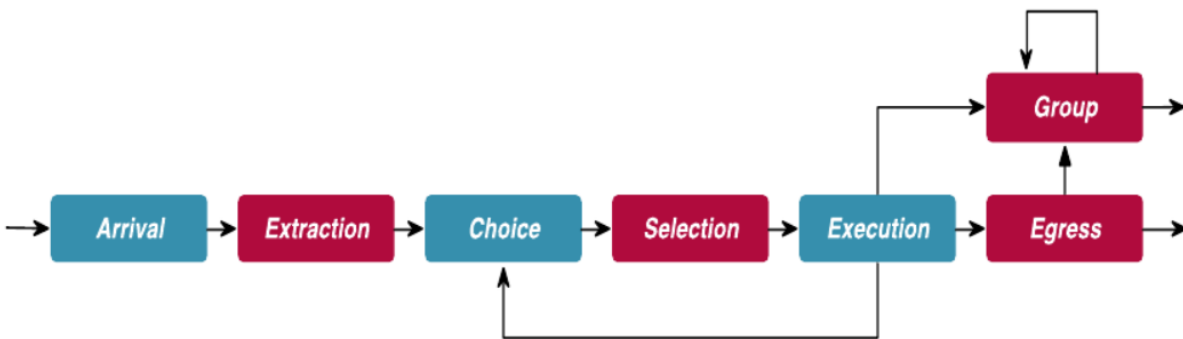
در شکل نمودار وابستگی برای OpenFlow 1.4، علامت "*" در مقابل نام یک پروتکل یا فیلد، بمعنای اختیاری بودن آن پروتکل یا فیلد در طبقه‌بند است و توسط پیام Table StatsRes از سوئیچ whitebox است که می‌توان از پشتیبانی یا عدم پشتیبانی سوئیچ از آن فیلد یا پروتکل اطلاع حاصل نمود.

چک کردن مدام زنجیره‌های وابستگی برای برنامه‌نویس کار نسبتاً دشوار و طاقت‌فرسایی می‌باشد و یک مشکل اضافه تلقی می‌گردد. برای حل این مشکل، کتابخانه‌هایی برای هر کنترلر نوشته شده است که برنامه‌نویس را از چک کردن زنجیره وابستگی‌ها حتی الامکان بی‌نیاز می‌کند. کتابخانه خوب برای کنترلر، کتابخانه‌ای است که قصد و هدف برنامه‌نویس را خوب برآورده کند؛ بطوریکه برنامه‌نویس دیگر درگیر حصول اطمینان از حضور پروتکل X نباشد و تمامی نمونه‌های^{۶۶} پروتکل X از دید برنامه‌نویس یکسان بنظر برسند. کتابخانه، مجموعه مطابقت‌های سطح بالای مشخص شده توسط برنامه‌نویس را به مجموعه مطابقت‌های سطح پایین معتبر برای OpenFlow تبدیل می‌کند [4].

Instruction 2.2.2.7

پس از مطابقت، این نوبت دستورالعمل‌های آن جریان است که روی بسته مطابقت شده اجرا شوند. اجرای این دستورالعمل‌ها فقط یک مرحله از مراحل^{۶۷} پردازش بسته در پایپلاین صفحه داده را تشکیل می‌دهد. برای درک چگونگی انجام این مرحله، لازم است با برخی مراحل که قبل از مرحله اجرا اتفاق می‌افتند آشنایی بیشتری داشته باشیم.

شکل زیر مراحل پایپلاین صفحه داده را نشان می‌دهد:



شکل 14 - شکل پایپلاین صفحه داده جهت پردازش بسته‌ها [9]

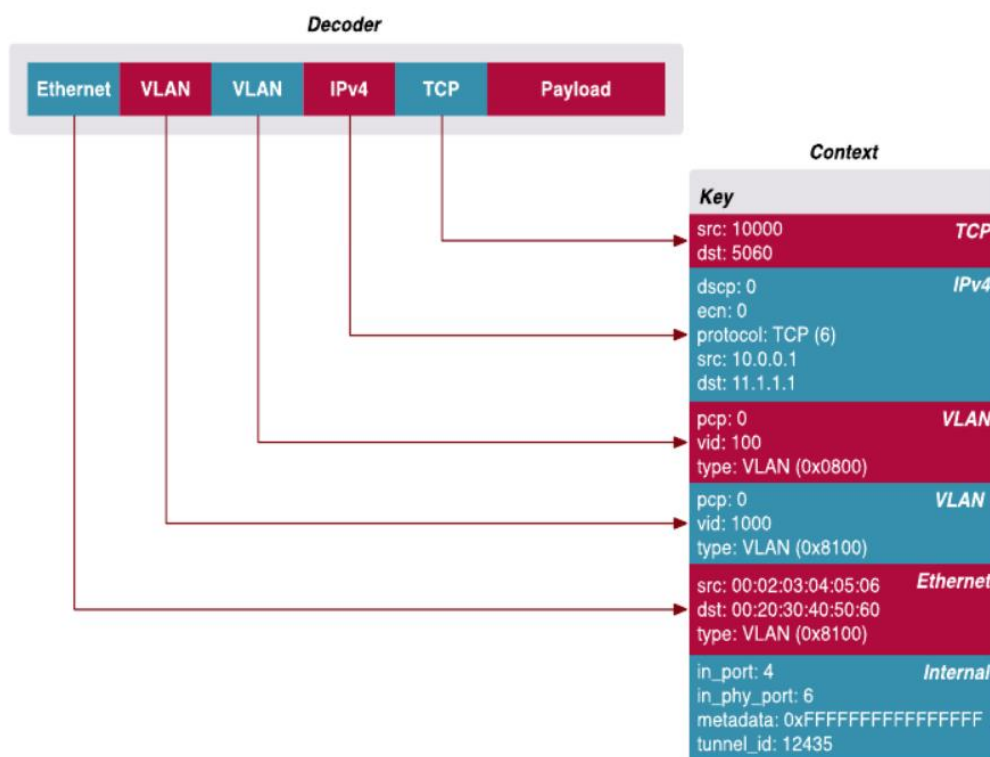
^{۶۶} Instances

^{۶۷} Stages

مراحل پردازش بسته در پایپلاین صفحه داده تا قبل از اجرا (Execution) بدین شرح می‌باشد:

1- Arrival: در این مرحله بسته صرفاً وارد پایپلاین صفحه داده می‌شود. در این مرحله دو موجودیت جدید شکل می‌گیرند:

- a. متن (Context): حاوی بیت‌های اطلاعات درباره خود بسته و همچنین درباره پردازش آن.
 - b. کلید (Key): که تعریف دقیق آن را در قسمت‌های قبل انجام دادیم و بنوعی نقش شاخص را برای جدول جریان ایفا می‌کند. دقت کنید کلید، همواره زیرمجموعه و بخشی از متن می‌باشد.
- 2- Extraction: در این مرحله با رمزگشایی⁶⁸، اطلاعات لازم از بسته استخراج شده در کلید قرار می‌گیرند. عبارتی در این مرحله کلید که در مرحله قبل شکل گرفته بود بروزرسانی می‌گردد. در این مرحله فیلدها و شناسه‌ها استخراج می‌گردند. شکل زیر، بیانگر آنچه در مرحله استخراج روی می‌دهد می‌باشد:



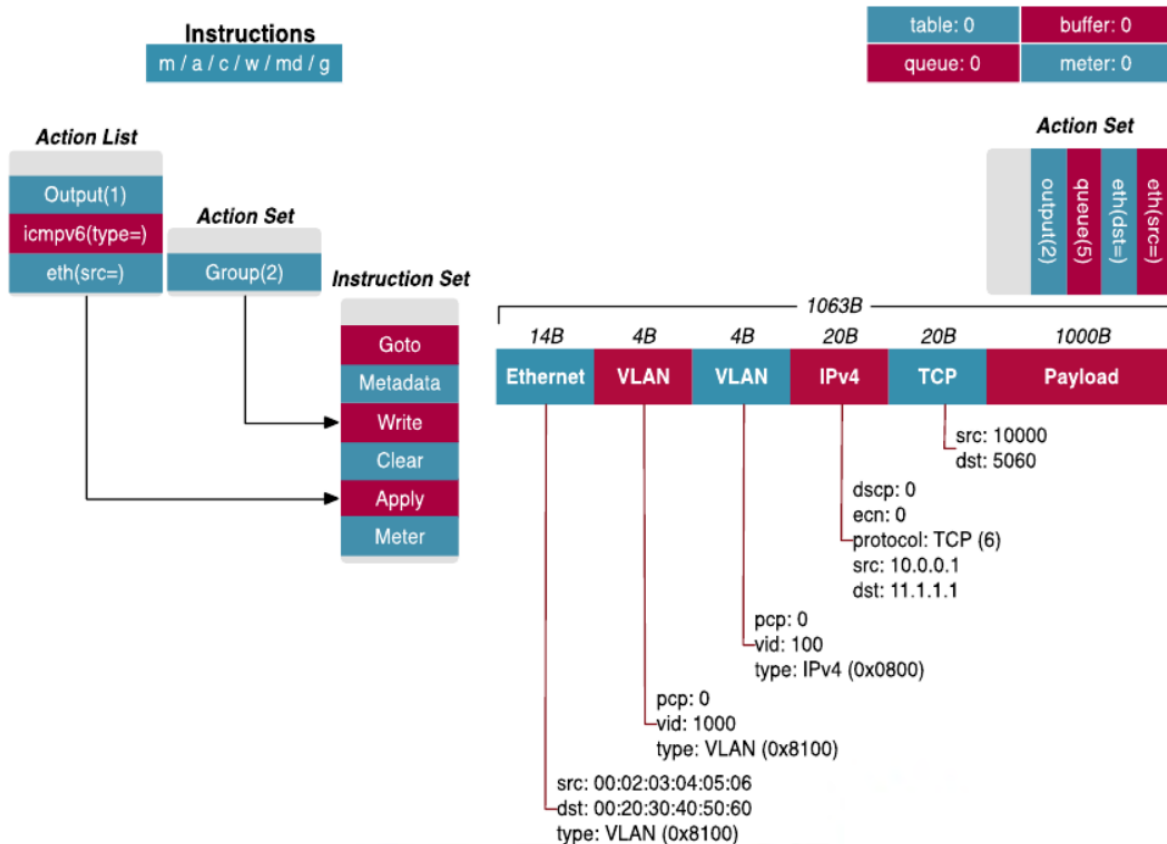
شکل 15 - نحوه رمزگشایی و استخراج کلید

3- Choice: در این مرحله با در دست داشتن متن (یا درست‌تر، table_id در متن) یک جدول از مجموعه جداول جهت پردازش بسته انتخاب می‌گردد.

⁶⁸ Decode

4- Selection: در این مرحله، طبقه‌بند از طریق مطابقت با کلید که از متن استخراج شده است مورد استفاده قرار می‌گیرد تا جریان مطابقت‌شده انتخاب گردد.

5- Execution: شرح مبسوط اتفاقاتی را که در این مرحله می‌افتد در ادامه آورده‌ایم. برای درک درست از مرحله Execution ابتدا تصویری از این مرحله آورده‌ایم که بطور کلی و شهودی این مرحله را نشان می‌دهد؛ سپس در ادامه به تشریح این مثال و اصطلاحات بکار رفته در شکل خواهیم پرداخت:



شکل 16 - شکل کلی از مرحله اجرا [9]

در شکل، بخشی مربوط به دستورالعمل‌ها تحت عنوان Instructions مشاهده می‌شود که در پایین آن 6 عبارت نوشته شده است. این عبارات، فرم اختصاری انواع دستورالعمل‌ها می‌باشد که با ممیز از یکدیگر جدا شده‌اند:

- **Meter (m):** مربوط به تجرید meter می‌باشد. بطور دقیق‌تر، meter یک تجرید مشخص شده در متن می‌باشد که این دستور، آنرا بروزرسانی می‌کند. از تجرید meter برای سیاست‌گذاری و شکل‌دهی به ترافیک استفاده می‌شود.

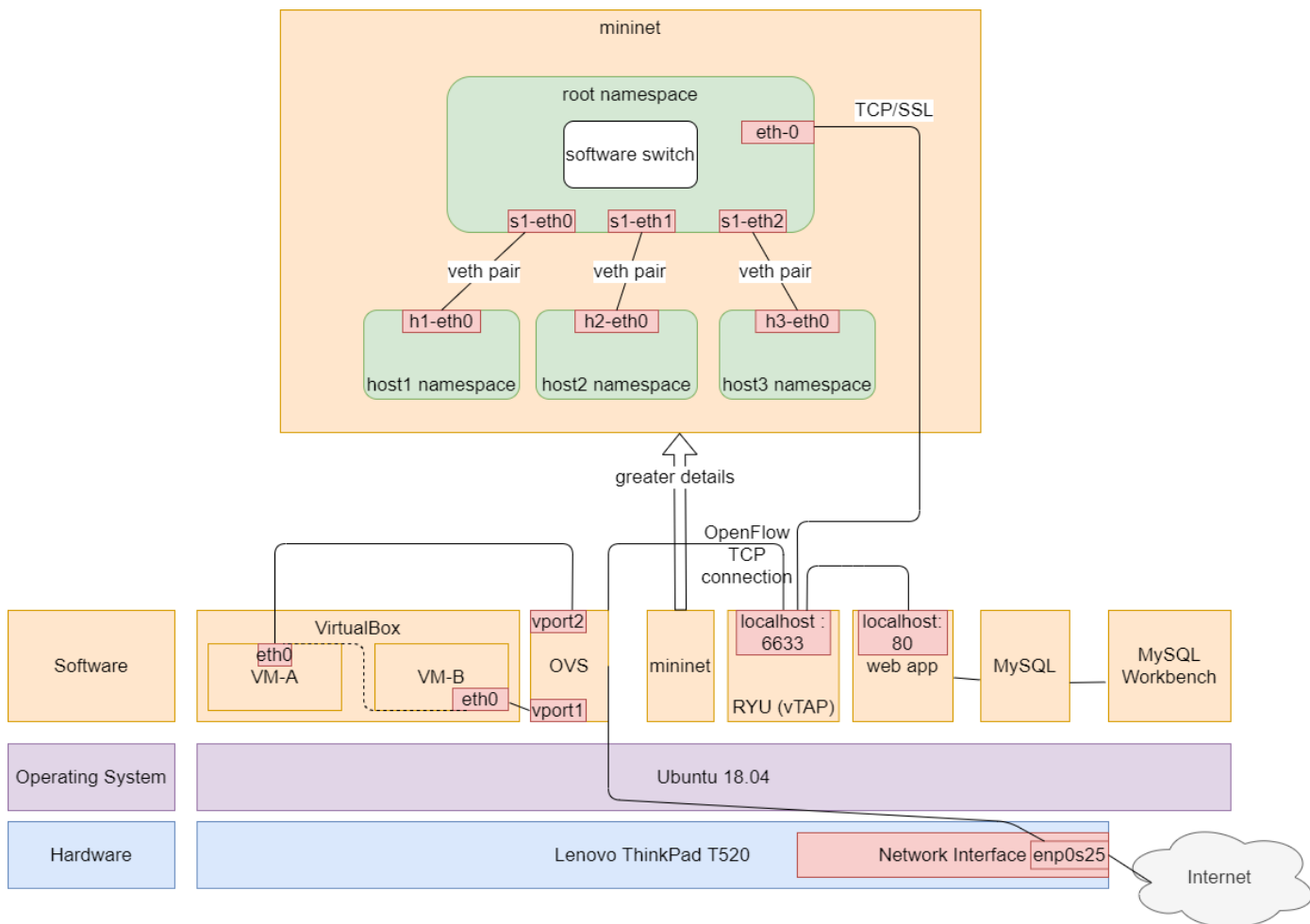
- Apply (a): تغییراتی را فوراً^{۶۹} روی بسته انجام می‌دهد. تغییرات با اجرای دستورالعمل‌های لیست اعمال بر متن بسته اتفاق می‌افتند. در انتهای این تغییرات بسته به یک پورت یا یک گروه تحویل داده می‌شود.
 - Clear (c): مجموعه اعمالی^{۷۰} که در حال حاضر در متن بسته مورد پردازش مشخص شده است را پاک می‌کند.
 - Write (w): این دستورالعمل مجموعه اعمالی را که در خود حمل می‌کند به مجموعه اعمال فعلی در متن بسته الحاق^{۷۱} می‌کند که نهایتاً^{۷۲} پس از اتمام پردازش بسته، دستورالعمل‌های مجموعه اعمال روی بسته اجرا می‌شوند.
 - Metadata (md): بسته را به همراه داده‌های جانبی به جدول بعدی پاس می‌دهد. بدین منظور از مقدار mask شده‌ای که در خود حمل می‌کند استفاده می‌کند تا فیلد metadata از متن بسته را بروزرسانی نماید.
 - Goto (g): با بروزرسانی فیلد table_id به جدول بعدی پرش^{۷۳} انجام می‌دهد.
- نکته‌ای که بیان آن در اینجا حائز اهمیت می‌باشد دقت به تفاوت مجموعه اعمال و لیست اعمال^{۷۴} است که در شکل به کار رفته‌اند. ابتدا باید دقت کرد که عمل‌ها در مجموعه اعمال و لیست اعمال، دستورالعمل نیستند و هر عمل، مجموعه دستورالعمل‌هایی است که صرفاً توسط دستورالعمل‌های write و apply استفاده می‌شوند. از دیگر تفاوت‌های مجموعه اعمال و لیست اعمال می‌توان به تفاوت در قیود استفاده از آن‌ها اشاره کرد. عمل‌های مجموعه اعمال نهایتاً اجرا می‌شوند (قید نهایتاً) ولی عمل‌های لیست اعمال فوراً اجرا می‌شود (قید فوراً). چنانچه هیچ عملی در مجموعه اعمال نباشد، در صورت عدم مطابقت با جریان‌های جدول، بسته دور ریخته خواهد شد. دیگر آن که در مجموعه اعمال، اعمال باید شرط یکتایی را برآورده کنند بدین منظور عمل‌ها نباید تکراری باشند، نیز بر اجرای آن‌ها ترتیبی حاکم باشد.
- توجه کنید مجموعه اعمال، مربوط به دستورالعمل write بوده و لیست اعمال، مربوط به دستورالعمل apply می‌باشد.

⁶⁹ Immediately⁷⁰ Action Set⁷¹ Append⁷² Eventually⁷³ Jump⁷⁴ Action List

در اینجا بحث درباره تجربدهای پروتکل OpenFlow را به پایان می‌بریم. اکنون، آماده آن هستیم تا کد یک سوئیچ یادگیرنده (learning switch) را مورد بررسی و اجرا قرار دهیم تا با استفاده از آن، اقدام به پیاده‌سازی monitor نماییم.

3 فصل سوم: معماری کلی سیستم

در این فصل معماری کلی سیستمی که در آن راهکار vTAP همراه monitor مورد پیاده‌سازی و اجرا قرار گرفته است را مورد بررسی قرار می‌دهیم. ابتدا شکل کلی سیستم را آورده و سپس به توضیح مؤلفه‌های سیستم خواهیم پرداخت:



شکل 17 - معماری کلی سیستم (بخش mininet برگرفته از [14])

Hardware 3.1

در لایه زیرین، سخت‌افزار قرار دارد. سیستم‌عامل و تمامی برنامه‌های نصب و یا اجرا شده سیستم‌عامل از منابع سخت‌افزار استفاده می‌کند. در این پروژه جهت پیاده‌سازی و اجرای راهکار vTAP همراه monitor از منابع

سخت‌افزاری لپ‌تاپ Lenovo ThinkPad T520 استفاده شده است. اتصال با اینترنت از طریق کابل LAN صورت می‌گیرد که به واسطه enp0s25 از لپ‌تاپ متصل شده است.

Operating System (Ubuntu) 3.2

روی سخت‌افزار، سیستم‌عامل یا همان سیستم میزبان^{۷۵} قرار می‌گیرد (مفهوم میزبان و میهمان در بخش مربوط به نرم‌افزار VirtualBox تعریف شده است). سیستم میزبان در این پروژه لینوکس Ubuntu نسخه 18.04 می‌باشد.

3.3 نرم‌افزار VirtualBox

ماشین‌های مجازی، به نرم‌افزاری جهت مدیریت، تخصیص و اشتراک‌گذاری منابعی از قبیل حافظه و پردازنده نیاز دارند که به آن hypervisor گفته می‌شود.

نرم‌افزار VirtualBox یک محصول مجازی‌سازی قدرتمند برای x86 و AMD64/Intel64 می‌باشد که برای مصارف تجاری و خانگی ساخته شده است. در حال حاضر، VirtualBox روی سیستم عامل‌های ویندوز، لینوکس، مکینتاش و سولاریس قابلیت اجرا دارد [15].

در این پروژه، نرم‌افزار VirtualBox نسخه 6.1.12 را بعنوان hypervisor محیط شبکه مجازی انتخاب کرده‌ایم. به هر ماشین مجازی در VirtualBox میهمان^{۷۶} و به سیستمی که VirtualBox در آن اجرا می‌شود میزبان گفته می‌شود. در ادامه جزئیات بیشتری راجع به انواع حالات شبکه کردن در VirtualBox و همچنین آماده‌سازی آن جهت پیاده‌سازی راهکار vTAP گفته شده است.

3.3.1 انواع حالات شبکه کردن در VirtualBox

با توجه به این که هدف نهایی پروژه مانیتورینگ در محیط مجازی است ابتدا باید شبکه مناسب را در محیط مجازی ایجاد کنیم. بدین منظور ابتدا با انواع حالات شبکه کردن در VirtualBox آشنا می‌شویم تا بتوانیم حالت متناسب با پروژه را انتخاب نماییم. جدول زیر 8 حالت اصلی شبکه کردن در VirtualBox را نشان می‌دهد:

⁷⁵ Host

⁷⁶ Guest

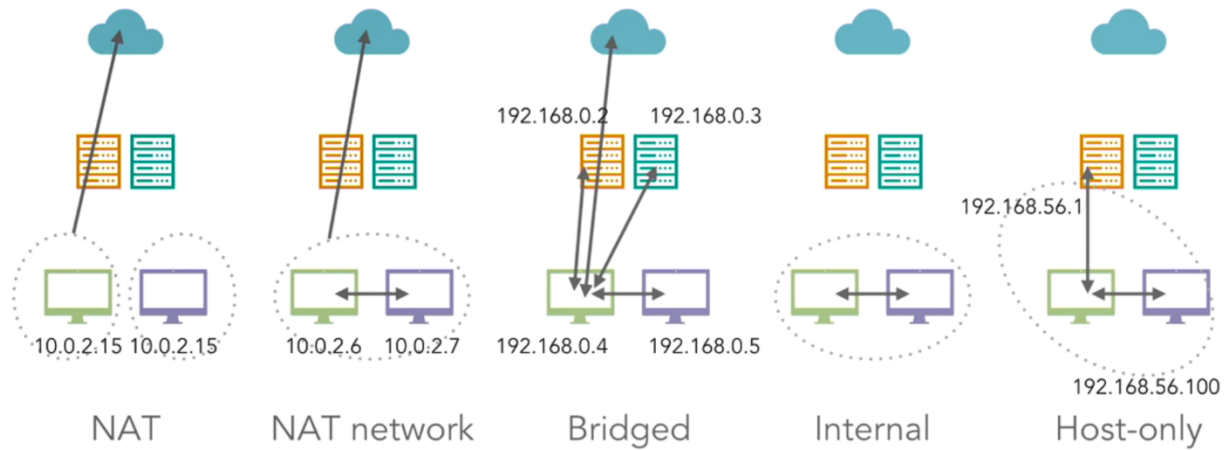
توضیحات	حالت شبکه
در این حالت، VirtualBox به میهمان گزارش می‌دهد که کارت شبکه موجود است، ولی هیچ اتصالی برقرار نمی‌شود. انگار کابل شبکه را کشیده و اتصال را قطع کرده‌ایم. از این حالت جهت الزام سیستم‌عامل میهمان به پیکربندی مجدد تنظیمات شبکه استفاده می‌گردد.	Not Attached
در این حالت تمامی ماشین‌های مجازی می‌توانند با IP یکسانی به اینترنت متصل شوند، ولی نمی‌توانند با یکدیگر ارتباط برقرار کنند.	Network Address Translation (NAT)
در این حالت کماکان NAT برقرار است با این تفاوت که یک شبکه مجازی برای اتصال ماشین‌های مجازی به یکدیگر نیز تعبیه شده است. محدودیت این شبکه، عدم اتصال ماشین‌های مجازی میهمان به میزبان می‌باشد.	NAT Network
برای نیازمندی‌های پیشرفته‌تر در شبکه (مثل شبیه سازی شبکه و اجرای سرور در میهمان) از این حالت استفاده می‌شود. در این حالت امکان اتصال ماشین‌های مجازی به یکدیگر و همچنین به میزبان وجود دارد و محدودیت حالت NAT Network برطرف شده است. برای استفاده از این حالت آدرس IP برای تمام سیستم‌ها باید در یک دامنه باشد. نیز آدرس‌دهی سیستم‌ها بعهده پروتکل DHCP در شبکه میزبان خواهد بود.	Bridged Networking
در این حالت ماشین‌های مجازی فقط و فقط به یکدیگر وصل هستند. مثل حالت NAT Network منتها بدون امکان اتصال به اینترنت.	Internal Networking
این حالت همانند Internal Networking است با این تفاوت که ماشین‌های مجازی علاوه بر یکدیگر، به	Host-Only Networking

<p>میزبان نیز می‌توانند متصل شوند تا نوعی شبکه خصوصی^{۷۷} شکل بگیرد.</p> <p>در این حالت، بجای واسط شبکه فیزیکی میزبان، واسطهای مجازی شبکه در میزبان ایجاد شده از آنها جهت ارتباط استفاده می‌گردد.</p>	
<p>چنین حالتی بندرت و در مواقعی که واسطهای کلی شبکه بین سیستم‌ها مشترک باشد استفاده می‌گردد. در این حالات کاربر می‌تواند یک راه‌انداز^{۷۸} موجود در VirtualBox و یا توزیع شده در بسته افزونه^{۷۹} را انتخاب نماید.</p> <p>در حال حاضر دو زیرمجموعه عمده برای این حالت وجود دارند:</p> <p>1-UDP Tunnel: برای اتصال مستقیم ماشین‌های مجازی میهمان که روی میزبان‌های مختلفی قرار گرفته‌اند (اطلاعات بیشتر با ذکر مثال در بخش 2.1.4).</p> <p>2-VDE⁸⁰: برای اتصال به یک سوئیچ اترنت مجازی روی میزبان لینوکس یا FreeBSD. لازم به ذکر است نرم‌افزار VirtualBox مورد نیاز در این سناریو باید از source کامپایل شده باشد چرا که پکیج‌های Oracle در نرم‌افزار VirtualBox نصب شده با package manager چنین امکاناتی را ندارد.</p>	Generic Networking

جدول 4 - انواع حالات شبکه کردن در VirtualBox [16]

⁷⁷ Private Network⁷⁸ Driver⁷⁹ Extension Pack⁸⁰ Virtual Distributed Ethernet

شکل زیر بطور خلاصه گویای برخی از حالات شبکه می‌باشد:



شکل 18 - انواع حالات شبکه کردن در VirtualBox [17]

با توجه به این که در این پروژه از طرفی اتصال به اینترنت برای کارکرد صحیح web app، از طرفی دیگر برقراری اتصال بین ماشین‌های مجازی به یکدیگر و همچنین سیستم میزبان (برای متصل شدن به OVS روی سیستم میزبان) الزامی می‌باشد از حالت Bridged Networking برای شبکه کردن ماشین‌های مجازی استفاده کرده‌ایم. چرا که امکاناتی از قبیل دسترسی به اینترنت، تخصیص آدرس‌های IP گوناگون به ماشین‌های مجازی و اتصال ماشین‌های مجازی میهمان به OVS به تمامی در حالت Bridged Networking برآورده می‌گردند.

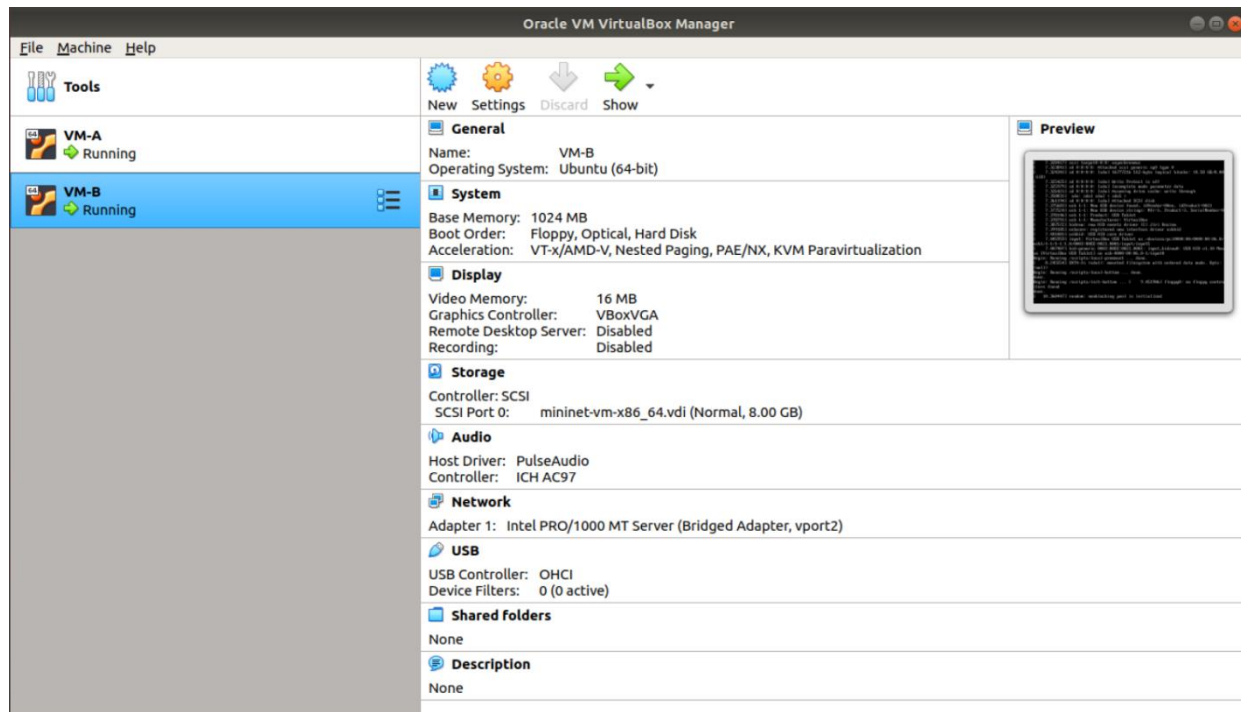
3.3.2 آماده‌سازی VirtualBox برای مانیتورینگ

برای آماده‌سازی VirtualBox ابتدا به 2 ماشین مجازی نیاز داریم تا بتوانیم با اجرای vTAP بین آن‌ها به مانیتورینگ ترافیک بین آن دو بپردازیم. جهت استفاده بهینه منابع سیستم، از ایمج آماده mininet استفاده کرده‌ایم. ماشین مجازی mininet که از روی ایمج آن نصب می‌شود، خود دارای ابزار mininet می‌باشد. توجه داشته باشید در این پروژه با ابزار mininet نصب شده در ماشین‌های مجازی هیچ گونه تعاملی نداریم و ابزار mininet مورد استفاده در پروژه آنیست که روی سیستم میزبان نصب گردیده است. این مطلب در شکل مربوط به معماری کلی سیستم بوضوح قابل تحصیل است.

در پاسخ به سؤال چرایی استفاده از ایمج‌های mininet علی‌رغم عدم استفاده از ابزار mininet نصب شده روی ماشین‌های مجازی باید گفت در این پروژه از ایمج mininet صرفاً جهت نصب 2 ماشین مجازی روی VirtualBox استفاده شده است. ماشین‌های مجازی که هر یک، نسخه متنی و بدون گرافیک ساده‌ای از لینوکس Ubuntu هستند. بعبارتی دیگر هدف در اینجا نصب دو ماشین مجازی ساده است و می‌توانستیم بجای

ایمیج آماده mininet، از نسخه آماده هر توزیع دلخواه دیگری از لینوکس استفاده کنیم؛ بشرطی که مصرف منابع آن کم بوده و بدون مشکل خاصی روی سیستم میزبان در لپ‌تاپ Lenovo ThinkPad T520 قابل اجرا باشد.

شکل زیر نمایی از نرم‌افزار VirtualBox را نشان می‌دهد که روی آن دو ماشین مجازی با نام‌های VM-A و VM-B ایجاد شده‌اند:



شکل 19 – نرم‌افزار VirtualBox پس از نصب ماشین‌های مجازی VM-A و VM-B

هر ماشین مجازی mininet در ابتدا برای ورود به سیستم، نام کاربری و رمز عبور می‌خواهد که در حالت پیشفرض هر دوی آنها mininet می‌باشند. برای اطلاع دقیق‌تر از نسخه Ubuntu و کرنل لینوکس نصب شده در هر یک از این ماشین‌های مجازی می‌توان دستور `cat /proc/version` را اجرا نمود. خروجی این دستور برای ماشین‌های مجازی نصب شده در این پروژه بشرح زیر است:

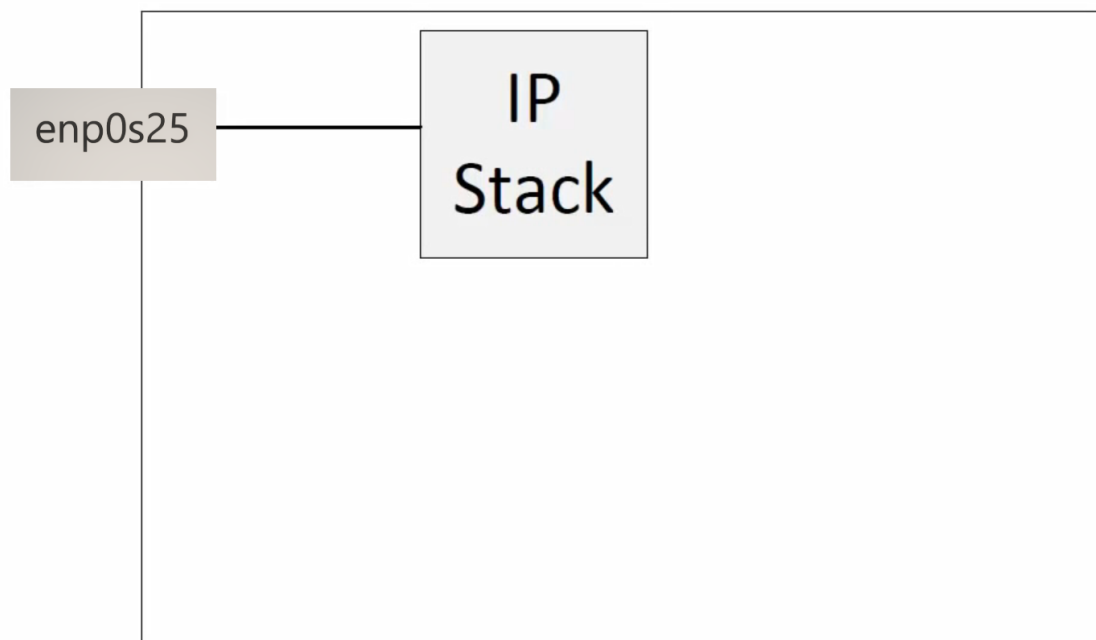
```
mininet@mininet-vm:~$ cat /proc/version
Linux version 4.2.0-27-generic (buildd@lcy01-23) (gcc version 4.8.2 (Ubuntu 4.8.2-19ubuntu1) ) #32~14.04.1-Ubuntu SMP Fri Jan 22 15:32:26 UTC 2016
```

شکل 20 – نسخه Ubuntu و کرنل لینوکس نصب شده در ماشین‌های مجازی VM-A و VM-B

در ادامه، اسکریپت ovs-activate را با دستور ./ovs-activate اجرا می‌کنیم. وظیفه این اسکریپت، ساختن واسطه‌های مجازی جهت ارتباط سوئیچ مجازی با ماشین‌های VM-A و VM-B می‌باشد. متن اسکریپت حاوی این دستورات است:

```
ovs-vsctl add-port mybridge enp0s25
ifconfig enp0s25 0
sleep 1
dhclient mybridge
sleep 1
ping google.com
ip tuntap add mode tap vport1
ip tuntap add mode tap vport2
sleep 1
ifconfig vport1 up
ifconfig vport2 up
ovs-vsctl add-port mybridge vport1
ovs-vsctl add-port mybridge vport2
```

در ادامه به توضیح کارکرد هر دستور می‌پردازیم. در ابتدای اجرای این فایل، پیکربندی سیستم به شکل زیر است:



شکل 21 – پیکربندی اولیه سیستم (برگرفته از [18])

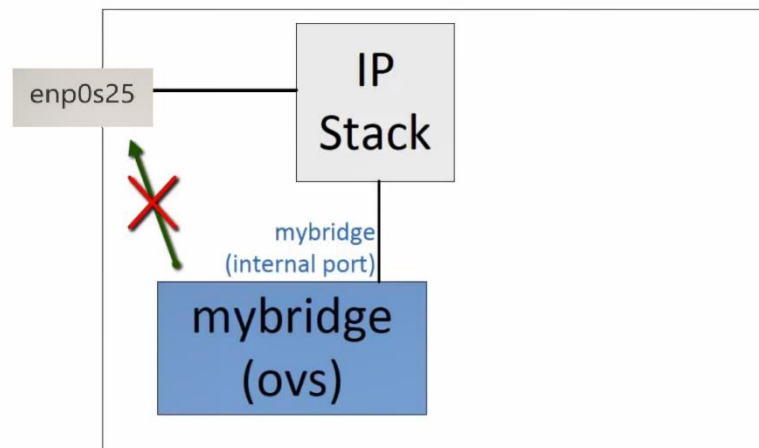
پیش‌تر گفتیم که برای شبکه مورد نیاز در پروژه، حالت Bridged Networking را انتخاب کرده‌ایم. علت این انتخاب، آنست که اولاً امکان اتصال به اینترنت هم برای میزبان و هم ماشین‌های مجازی میهمان (VM-A و VM-B) برقرار باشد و سلب نگردد؛ ثانیاً برای اتصال ماشین‌های مجازی به سوئیچ مجازی، لازم است در سیستم میزبان (که سوئیچ مجازی در آن قرار دارد) واسطه‌های مجازی ساخته شود و برای این کار حالتی از شبکه کردن نیاز است که در آن واسطه‌های شبکه در میزبان برای ماشین‌های مجازی میهمان قابل رؤیت باشند.

متأسفانه حالت bridged برای واسطه‌های بی‌سیم با مشکلات و محدودیت‌هایی مواجه است که برای واسطه‌های سیم‌دار یا همان با سیم چنین مشکلات و محدودیت‌هایی وجود ندارد. بطور کلی می‌توان گفت حالت bridged همواره برای میزبان متصل به واسط شبکه با سیم کار می‌کند. ولی حالت bridged لزوماً برای میزبان با واسط بی‌سیم کار نمی‌کند. بصورت فنی، این امر ناشی از محدودیت شدید در پیاده‌سازی پروتکل‌های Wifi توسط راه‌انداز آداپتور بی‌سیم یا میان‌افزار نقطه دسترسی⁸¹ می‌باشد. پیاده‌سازی‌های خیلی خاصی از پروتکل‌های Wifi هستند که در آن‌ها امکان استفاده از امکانات حالت bridged برقرار است [19]؛ پس در نتیجه برای استفاده از حالت bridged، دستگاه میزبان را با کابل LAN به شبکه وصل می‌کنیم. واسط با سیم مربوط به این اتصال، در

⁸¹ Access Point Firmware

خروجی دستور ifconfig تحت عنوان enp0s25 آمده است که آنرا در شکل نیز مشاهده می‌کنید. البته واسط بی‌سیم نیز در سیستم میزبان حضور دارد ولی بعلت بی‌تأثیر بودن آن در روند انجام کارهای مربوط به پروژه، از بحث درباره آن و آوردن نام آن در شکل‌ها خودداری نموده‌ایم.

تا بدینجا فرض شده است که سوئیچ مجازی با نام mybridge در سیستم میزبان ساخته شده است. ولی صرف ساختن آن و اتصال آن به ماشین‌های مجازی، باعث نمی‌شود که ماشین‌های مجازی دسترسی به اینترنت داشته باشند. دسترسی به اینترنت در واسط enp0s25 وجود دارد ولی دقت کنید این واسط به mybridge وصل نمی‌باشد:

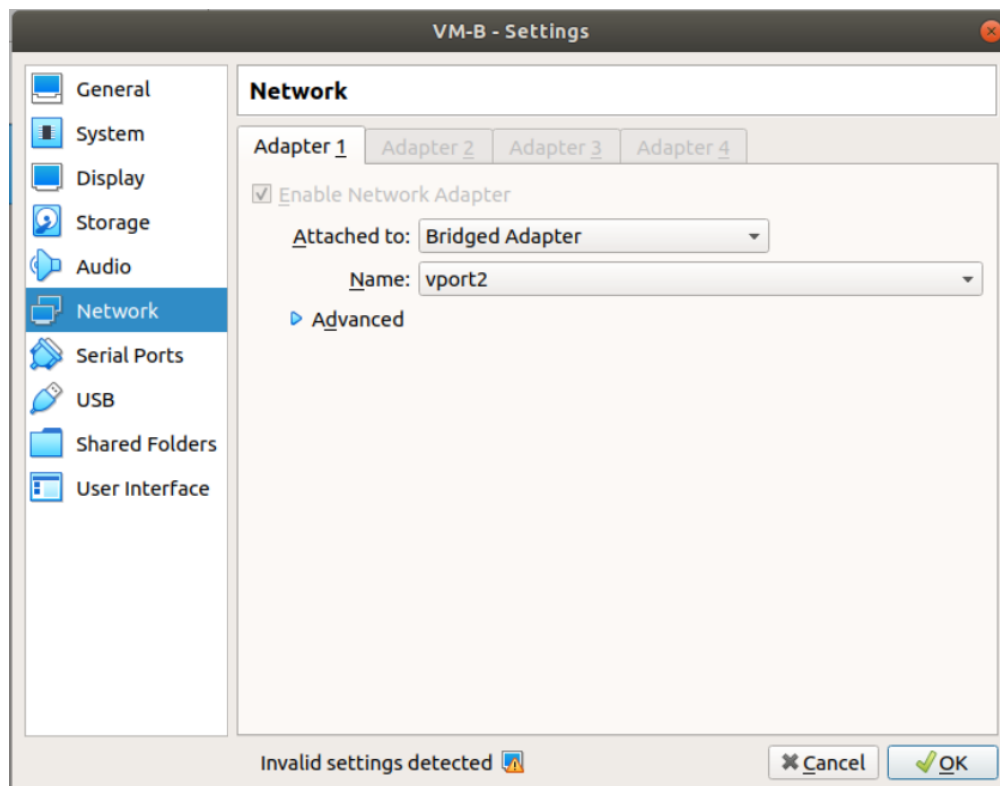


شکل 22 – عدم اتصال سوئیچ mybridge به اینترنت (برگرفته از [18])

برای رفع این مشکل ابتدا واسط enp0s25 را به mybridge اضافه می‌کنیم. ولی در آنصورت سیستم میزبان اتصالش به اینترنت را از دست می‌دهد. با توجه به این که پروتکل DHCP می‌تواند آدرس‌دهی سیستم‌ها را در حالت bridged انجام دهد، راهکارمان این است که ابتدا آدرس IP مربوط به واسط enp0s25 را با دستور `ifconfig enp0s25 0` از بین ببریم؛ سپس با دستور `dhclient mybridge`، پورت داخلی (internal port) از mybridge را به DHCP client تبدیل کنیم. با این کار، همه سیستم‌های متصل به آن، آدرس IP خواهند گرفت. پس از انجام این کار، می‌توان با وارد کردن دستور `ping google.com` از اتصال میزبان به اینترنت اطمینان حاصل کرد.

برای اتصال ماشین‌های مجازی به mybridge، نوع خاصی از واسط‌ها را باید در سوئیچ mybridge ایجاد کنیم که به آن‌ها واسط TAP گفته می‌شود. در اسکریپت، دو واسط TAP با نام‌های vport1 و vport2 ایجاد کرده‌ایم. در سناریو مربوط به VirtualBox، راهکار vTAP همان واسط‌های TAP با نام‌های vport1 و vport2 هستند. نرم‌افزار جمع‌آوری اطلاعات از vTAP (همان monitor) برنامه پایتون نوشته شده در کنترلر Ryu می‌باشد.

ایجاد واسطه‌های TAP1 و TAP2، عملیات IO از نوع حساس است و برنامه ایجاد این واسطه‌ها در ناحیه بحرانی قرار دارد. چنانچه هنگام ایجاد این واسطه‌ها و قبل از اتمام پروسه ایجادکننده این واسطه‌ها، پروسه دیگری بخواهد با این واسطه‌ها کار کند ارور device or resource busy : ioctl (TUNSETIFF) رخ خواهد داد. برای جلوگیری از چنین اتفاقی یک دستور sleep 1 هم کفایت می‌کند که اجرای دستور بعدی جهت فعال کردن این واسطه‌ها را 1 ثانیه به تأخیر بیندازد [20]. اینگونه از خاتمه عملیات IO پیش از شروع دستور بعدی اطمینان حاصل می‌کنیم. پس از اضافه کردن vport1 و vport2 به mybridge اکنون نوبت انتخاب این واسطه‌ها در ماشین مجازی است. شکل زیر نحوه انتخاب vport2 برای VM-B را نشان می‌دهد:

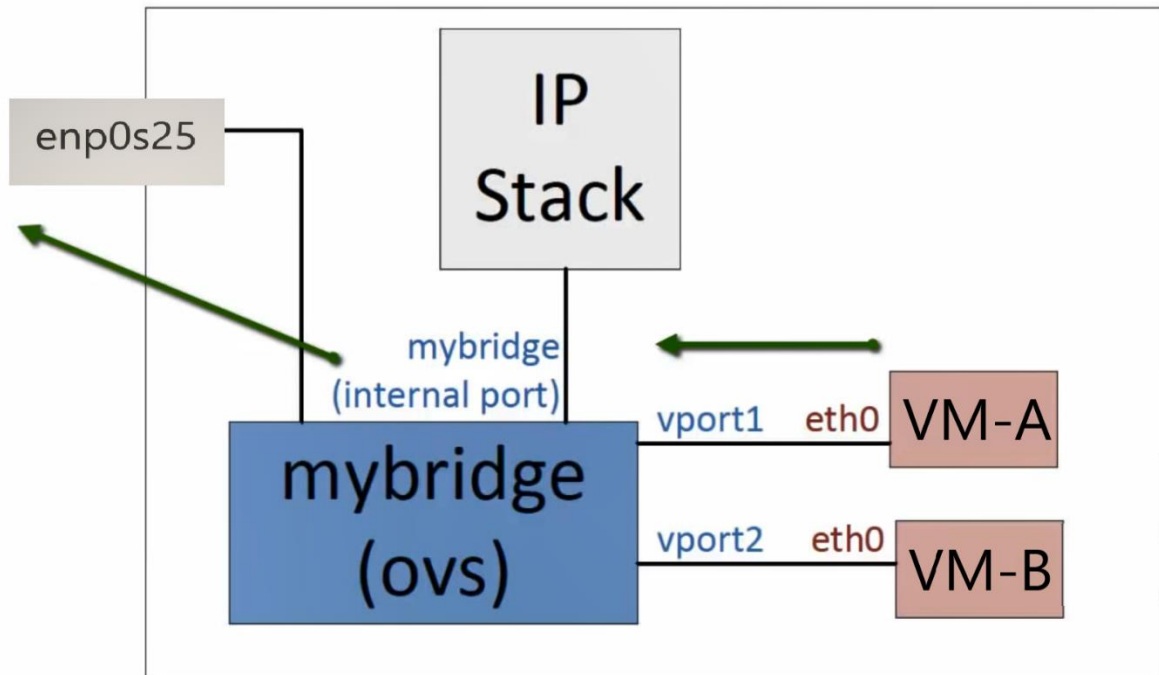


شکل 23 - نحوه انتخاب واسطه مجازی vport2 برای ماشین مجازی VM-B

مطابق شکل در قسمت Network از پنل تنظیمات مربوط به ماشین مجازی، در قسمت Attached to، با انتخاب گزینه Bridged Adapter، حالت شبکه Bridged Networking را برمی‌گزینیم. جلوی قسمت Name نیز از منوی combo box، گزینه vport2 را انتخاب می‌نماییم.

مشابه کاری که در شکل بالا نشان داده شده است را برای VM-A نیز انجام می‌دهیم و برای آن vport1 را انتخاب می‌کنیم. برای اطمینان از اتصال VM-A و VM-B به یکدیگر، در یکی از ماشین‌های مجازی با وارد کردن دستور ifconfig در قسمت eth0 آدرس IP آن ماشین مجازی را بدست آورده و آنرا از سمت ماشین

مجازی دیگر ping می‌کنیم. اطمینان از اتصال آن دو به اینترنت نیز کاملاً مشابه میزبان می‌باشد. پس از این اتصال پیکربندی سیستم بصورت شکل زیر خواهد بود:



شکل 24 - پیکربندی سیستم پس از آماده‌سازی شبکه مد نظر برای انجام پروژه (برگرفته از [18])

نهایتاً برای برگرداندن شبکه موجود به همان حالت اولیه (پیش از اجرای ovs-activate) می‌توان اسکریپت ovs-deactivate را با دستور ovs-deactivate اجرا نمود. متن اسکریپت در ذیل آورده شده است:

```
ifconfig mybridge 0
ovs-vsctl del-port mybridge vport1 -- del-port mybridge vport2
ovs-vsctl del-port mybridge enp0s25
ip link delete vport1
ip link delete vport2
ifconfig enp0s25 192.168.1.7
ping google.com
```

برای خارج کردن پورت داخلی mybridge از حالت DHCP client، با دستور ifconfig mybridge 0 آدرس IP

آنها بر می‌داریم. سپس با دستور ovs-vsctl del-port به قطع اتصال واسط‌های مجازی vport1 و vport2 و

همچنین واسط enp0s25 از سوئیچ می‌پردازیم. حال با دستور ip link delete به حذف واسط‌های vport1 و

vport2 از سیستم میزبان اقدام می‌کنیم [21]. چراکه در غیر اینصورت، این واسط‌ها در سیستم باقی می‌مانند

(در خروجی ifconfig می‌توان آنها را مشاهده نمود). این امر اجرای مجدد اسکریپت ovs-activate جهت

آماده‌سازی مجدد بستر شبکه مورد نیاز جهت اجرا و تست vTAP را با مشکل مواجه خواهد ساخت.

3.4 معرفی OVS (Open vSwitch)

3.4.1 درباره OVS

OVS یک سوئیچ مجازی چندلایه تحت لیسانس Apache 2.0 می‌باشد. این سوئیچ مجازی، ابزاری برای اتوماسیون شبکه‌های انبوه توسط افزونه‌های برنامه‌نویسی می‌باشد که از واسطها و پروتکل‌های مدیریت شبکه (از قبیل NetFlow، sFlow، IPFIX، RSPAN، CLI، LACP، 802.1ag و غیره) پشتیبانی می‌کند. بعلاوه، این سوئیچ می‌تواند از سرویس‌های توزیع شده در چند سرور فیزیکی مشاله سوئیچ مجازی توزیع شده vNetwork از VMware و یا Nexus 1000V از Cisco پشتیبانی کند [22].

3.4.2 نصب OVS

نصب OVS از سورس کاری نسبتاً دشواری می‌باشد به همین خاطر از بیان آن صرفنظر کرده تنها به ذکر نصب آن با ابزار مدیریت پکیج apt اکتفا می‌کنیم:

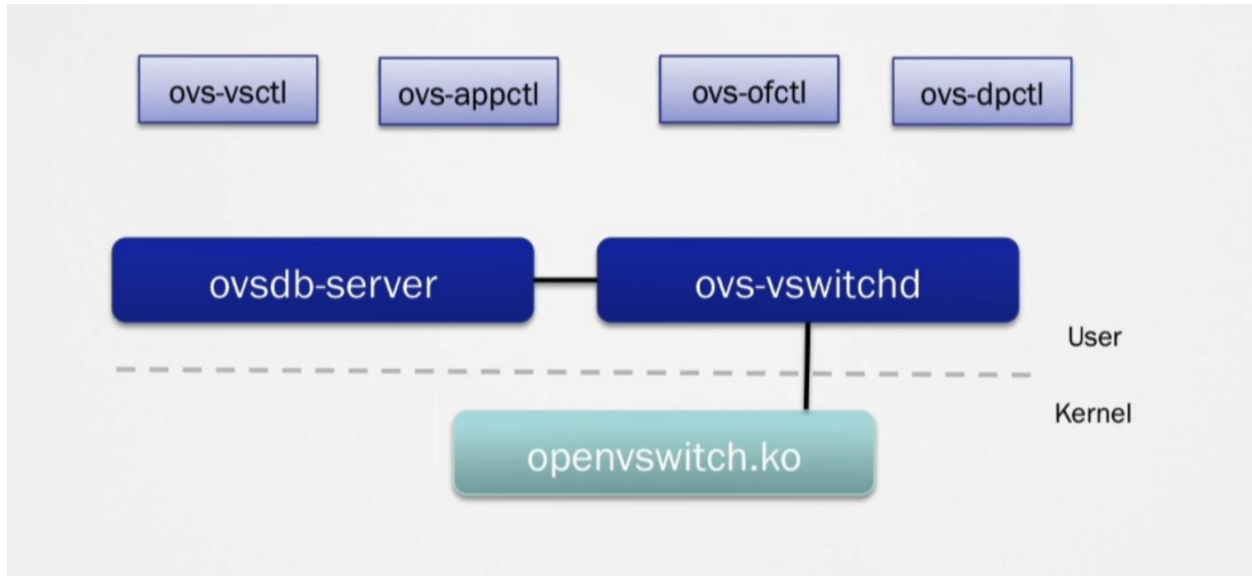
```
apt-get install openvswitch-switch
```

پس از نصب با دستور `ovs-vsctl -V` می‌توان مطلع شد که کدام نسخه از OVS و DB Schema روی سیستم میزبان نصب شده است. در این پروژه، در سیستم میزبان OVS نسخه 2.9.5 به همراه DB Schema نسخه 7.15.1 نصب شده‌اند.

در قسمت بعدی درباره `ovs-vsctl` و همچنین سایر دستورات اصلی OVS که با مؤلفه‌های اصلی OVS سر و کار دارند توضیحاتی به میان رفته است.

3.4.3 مؤلفه‌های کلیدی در OVS

شکل زیر معماری OVS را به همراه مؤلفه‌های اصلی آن نشان می‌دهد:



شکل 25 - معماری کلی OVS [23]

مؤلفه‌های کلیدی در OVS را می‌توان به دو نوع کلی تقسیم کرد:

1- مؤلفه‌های مربوط به حالت کاربر^{۸۲}:

a. ovs-vswitchd: برنامه کمکی^{۸۳} حالت کاربر برای سوئیچینگ

b. ovsdb-server: پایگاه داده پیکربندی OVS

2- مؤلفه‌های مربوط به حالت کرنل^{۸۴}:

a. openvswitch kernel module: در شکل بصورت openvswitch.ko مشخص شده است.

همانطور که از شکل نیز پیداست این مؤلفه با ovs-vswitchd ترکیب شده عمل سوئیچینگ را

در حالت کرنل انجام می‌دهد.

ابزارهای واسط خط فرمان^{۸۵} برای OVS عبارتند از:

- ovs-vsctl: ابزاری برای مدیریت وضعیت پیکربندی در ovsdb-server (برخلاف نامش که با vs شروع می‌شود با ovs-vswitchd ارتباطی ندارد)

⁸² User Mode

⁸³ Daemon

⁸⁴ Kernel Mode

⁸⁵ Command Line Interface (CLI)

- ovs-appctl: ابزاری برای ارسال دستورات و فرامین به ovs-vswitchd
 - ovs-dpctl: ابزاری برای پیکربندی ماژول کرنل
 - ovs-ofctl: ابزاری برای کار با پروتکل OpenFlow
- راه‌های عمده ارتباط مؤلفه‌ها با یکدیگر بدینصورت می‌باشند:
- OVSDb management protocol که در RFC 7047 تحت عنوان JSON-RPC آورده شده است. این راه، ارتباط بین ovs-vswitchd و ovsdb-server را برقرار می‌کند (خط افقی بین آن دو در شکل).
 - OpenFlow که ارتباط بین ovs-vswitchd و ovs-ofctl را برقرار می‌کند.
 - Netlink که ارتباط بین ovs-vswitchd و ماژول کرنل را برقرار می‌کند (خط عمودی بین آن دو در شکل).

نهایتاً اطلاعات گفته شده را بطور خلاصه می‌توان در جدول زیر مشاهده کرد:

نام زیربخش	هدف	واسط‌های مربوط	ابزارهای واسط خط فرمان
ovsdb-server	<ul style="list-style-type: none"> ذخیره اطلاعات پیکربندی برای OVS ماندگاری وضعیت در راه‌اندازی مجدد^{۸۶} 	<ul style="list-style-type: none"> پروتکل مدیریت ovs-vswitchd پروتکل مدیریت کنترلر خارجی 	<ul style="list-style-type: none"> ovs-vsctl
ovs-vswitchd	<ul style="list-style-type: none"> هسته اصلی سیستم مدیریت سوئیچ‌های متعدد پیکربندی شده در میزبان درگیر در پردازش‌های مرتبط با مسیر داده^{۸۷} (این که کجا و چگونه توسط ماژول کرنل صدا زده شود) 	<ul style="list-style-type: none"> پروتکل مدیریت ovsdb-server پروتکل OpenFlow برای اتصال به کنترلر Netlink برای ارتباط با ماژول کرنل 	<ul style="list-style-type: none"> ovs-appctl ovs-ofctl

^{۸۶} Reboot

^{۸۷} Datapath

• ovs-dpctl	• Netlink برای ارتباط با ovs-vswitchd	• پردازش بسته‌ها در کرنل	Openvswitch kernel module یا همان ماژول کرنل
-------------	---------------------------------------	--------------------------	---

جدول 5 – مؤلفه‌های کلیدی در OVS به‌مراه اهداف، واسطه‌ها و ابزارهای خط فرمان برای هر مؤلفه [23]

3.5 ابزار mininet

Mininet یک ابزار شبیه‌سازی شبکه است که یک شبکه مجازی واقع‌گرا ایجاد می‌کند، و این کار را با اجرای کرنل واقعی، سوئیچ و کد یک برنامه کاربردی، آن هم روی یک ماشین (ماشین مجازی، ابر یا بومی) انجام می‌دهد [24].

تعامل آسان با شبکه توسط واسطه خط فرمان (و همچنین api)، امکان سفارشی‌سازی⁸⁸ شبکه، اشتراک‌گذاری آن با دیگران و همچنین بکارگیری آن روی سخت‌افزار واقعی، ابزار mininet را به یک ابزار پرکاربرد برای توسعه، یادگیری و تحقیق تبدیل می‌کند [24].

بعلاوه mininet راهی عالی برای توسعه، مشارکت و آزمایش با پروتکل OpenFlow و شبکه‌های نرم‌افزار محور محسوب می‌گردد. گزینه‌های نصب mininet بطور عمده عبارتند از [24]:

- استفاده از ایمیج ماشین مجازی mininet که یک سیستم Ubuntu ساده در اختیارمان می‌گذارد که روی آن mininet نصب شده است.
- نصب mininet از سورس
- نصب mininet با ابزارهای مدیریت پکیج در لینوکس

در این پروژه از دو ماشین مجازی با نام‌های VM-A و VM-B استفاده شده است. برای شبیه‌سازی شبکه با mininet از روش نصب با ابزارهای مدیریت پکیج استفاده شده است. بدین منظور در سیستم میزبان دستورهای مقابل را وارد می‌کنیم:

```
apt update
apt install mininet
```

⁸⁸ Customization

ابزار mininet یکی از کامل‌ترین ابزارهای شبیه‌سازی شبکه است که از شبکه‌های نرم‌افزار محور پشتیبانی می‌کند. این ابزار بصورت پیش‌فرض از توپولوژی‌های شناخته شده مانند درخت و ستاره پشتیبانی می‌کند و برای ساخت توپولوژی‌های پیچیده‌تر می‌توان از زبان پایتون برای توصیف آن استفاده کرد. جدول زیر تعدادی از دستورات پرکاربرد این ابزار را نشان می‌دهد [25]:

دستور	توضیحات
<host> ping <host>	برای ارسال ping از یک میزبان به میزبان دیگر
net	نمایش متنی نحوه اتصال سوئیچ‌ها و میزبان‌ها
ports	نمایش پورت‌های متصل در سوئیچ‌ها
dump	نمایش کلی اطلاعات مربوط به شبکه شامل IP میزبان‌ها، مدل سوئیچ‌ها و کنترلر شبکه

جدول 6 - تعدادی از دستورات پرکاربرد mininet

برای این که mininet بتواند سوئیچ‌های مجازی و همچنین میزبان‌ها را روی یک کامپیوتر شبیه‌سازی کند، از تکنولوژی‌های پیشرفته‌ای در کرنل لینوکس بهره می‌گیرد. بطور کلی، mininet مجازی‌سازی را بطور کامل انجام نمی‌دهد و سوئیچ‌های مجازی و همچنین میزبان‌ها بصورت فرآیندهایی ساده روی سیستم اصلی (در اینجا همان سیستم عامل Ubuntu 18.04 که ابزار mininet روی آن نصب شده است) تقلید^{۸۹} می‌شوند.

در mininet نیاز است که هر یک از این فرآیندها بتواند بعنوان دستگاهی منفرد و منحصر بفرد از شبکه رفتار نماید. بدین منظور از namespace های شبکه استفاده می‌گردد. مفهوم namespace در لینوکس به ما این امکان را می‌دهد که هر فرآیند را به واسطه شبکه، جدول مسیریابی و جدول ARP منحصر بفرد خودش مجهز کنیم. بدین طریق هر فرآیند میتواند به مثابه دستگاهی منحصر بفرد در شبکه رفتار نماید.

همانطور که در شکل معماری کلی سیستم پیداست در یک توپولوژی mininet، سوئیچ مجازی در یک namespace به نام root قرار گرفته که توسط واسطه eth0 خود به کنترلر (در اینجا Ryu) متصل میگردد. نیز هر یک از میزبان‌های ایجاد شده در توپولوژی در یک namespace جدا قرار میگیرد. ارتباط هریک از namespace های میزبان با root namespace حاوی سوئیچ مجازی از طریق veth pair برقرار می‌شود. Veth مخفف Virtual

⁸⁹ Emulate

Ethernet Device می‌باشد. Veth ها میتوانند بعنوان تونل بین namespace های شبکه عمل نمایند [26]. راهکار vTAP در این سناریو، پورت‌های سوئیچ نرم‌افزاری با نام s<i>-eth0 می‌باشند. i شماره واسطی از سوئیچ نرم‌افزاری است که توسط veth pair به یک host بخصوص متصل می‌باشد. در این سناریو نیز همانند VirtualBox، نرم‌افزار جمع‌آوری اطلاعات از vTAP (همان monitor) برنامه پایتون نوشته شده در کنترلر Ryu می‌باشد.

3.6 معرفی کنترلر Ryu

کنترلر Ryu یک فریم‌ورک مبتنی بر مؤلفه⁹⁰ برای شبکه‌های نرم‌افزار محور می‌باشد. Ryu از مؤلفه‌های نرم‌افزاری برخوردار است که هر یک، api خوش تعریف برای توسعه‌دهندگان فراهم می‌کنند تا بتوانند برنامه‌های کاربردی مختص مدیریت و کنترل شبکه ایجاد نمایند. Ryu از پروتکل‌های مختلفی از قبیل OpenFlow، NETCONF، OF-CONFIG و غیره پشتیبانی می‌کند. بعلاوه Ryu از افزونه‌های Nicira بطور کامل پشتیبانی می‌کند. کنترلر RYU تماماً به زبان پایتون نوشته شده است.

3.6.1 نصب کنترلر Ryu

یک راه نصب کنترلر Ryu توسط pip است که ابزار مدیریت پکیج برای پایتون می‌باشد:

```
pip install ryu
```

راه دیگر، نصب آن از سورس است:

```
git clone https://github.com/osrg/ryu.git
```

```
cd ryu
```

```
sudo pip install .
```

مطالب بیشتر در مورد طرز کار کنترلر Ryu در بخش 3.3 مربوط به بررسی و اجرای کد سوئیچ یادگیرنده گفته شده است.

⁹⁰ Component-Based

MySQL Workbench و MySQL، Web App 3.7

در این پروژه MySQL و MySQL Workbench آنچنان نقش محوری ندارند و بیشتر در تعامل با واسط کاربری وب مطرح می‌شوند. بنابراین هر دوی آن‌ها را با هم در کنار web app در یک زیربخش بررسی کرده‌ایم. برنامه کاربردی وب یا همان web app پروژه توسط فریم‌ورک لاراول ایجاد شده است. از سویی web app با کنترلر Ryu ارتباط برقرار کرده و داده‌های مربوط به مانیتورینگ شبکه ماشین‌های مجازی (و یا شبکه مربوط به توپولوژی mininet) را از برنامه کنترلر (همان monitor که به زبان پایتون نوشته شده) دریافت می‌نماید. از سویی دیگر web app به پایگاه داده MySQL متصل است که بتواند اطلاعات دریافت‌شده از برنامه کنترلر را در پایگاه داده جهت نمایش روی صفحات وب مربوط به پروژه ذخیره نماید.

برای سهولت کار با MySQL واسط‌های کاربری گرافیکی مختلفی همچون dbForge Studio، MySQL Workbench، phpMyAdmin و غیره وجود دارند که در این پروژه از واسط گرافیکی MySQL Workbench استفاده شده است.

4 فصل چهارم: بررسی سوئیچ یادگیرنده

در این فصل ضمن بررسی کد پایتون مربوط به یک سوئیچ یادگیرنده، با کلیاتی از سوئیچ مجازی Open vSwitch و همچنین کنترلر Ryu آشنا می‌شویم. آشنایی با این مطالب، پایه و اساس کار پیاده‌سازی vTAP خواهد بود.

4.1 بررسی و اجرای کد سوئیچ یادگیرنده

4.1.1 بررسی کد سوئیچ یادگیرنده

کد سوئیچ یادگیرنده (learning switch) مربوط به فایل simple_switch_13.py در ذیل آمده است [27]:

```
# Copyright (C) 2011 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ether_types

class SimpleSwitch13(app_manager.RyuApp):
    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]

    def __init__(self, *args, **kwargs):
        super(SimpleSwitch13, self).__init__(*args, **kwargs)
        self.mac_to_port = {}

    @set_ev_cls(ofp_event.EventOFPSwitchFeatures, CONFIG_DISPATCHER)
```

```

def switch_features_handler(self, ev):
    datapath = ev.msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    # install table-miss flow entry
    #
    # We specify NO BUFFER to max_len of the output action due to
    # OVS bug. At this moment, if we specify a lesser number, e.g.,
    # 128, OVS will send Packet-In with invalid buffer_id and
    # truncated packet data. In that case, we cannot output packets
    # correctly. The bug has been fixed in OVS v2.1.0.
    match = parser.OFPMatch()
    actions = [parser.OFPActionOutput(ofproto.OFPP_CONTROLLER,
                                      ofproto.OFPCML_NO_BUFFER)]
    self.add_flow(datapath, 0, match, actions)

def add_flow(self, datapath, priority, match, actions, buffer_id=None):
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    inst = [parser.OFPInstructionActions(ofproto.OFPIT_APPLY_ACTIONS,
                                         actions)]

    if buffer_id:
        mod = parser.OFPFlowMod(datapath=datapath, buffer_id=buffer_id,
                                priority=priority, match=match,
                                instructions=inst)
    else:
        mod = parser.OFPFlowMod(datapath=datapath, priority=priority,
                                match=match, instructions=inst)
    datapath.send_msg(mod)

@set_ev_cls(ofp_event.EventOFPPacketIn, MAIN_DISPATCHER)
def _packet_in_handler(self, ev):
    # If you hit this you might want to increase
    # the "miss_send_length" of your switch
    if ev.msg.msg_len < ev.msg.total_len:
        self.logger.debug("packet truncated: only %s of %s bytes",
                          ev.msg.msg_len, ev.msg.total_len)

    msg = ev.msg
    datapath = msg.datapath
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser
    in_port = msg.match['in_port']

    pkt = packet.Packet(msg.data)
    eth = pkt.get_protocols(ethernet.ethernet)[0]

    if eth.ethertype == ether_types.ETH_TYPE_LLDP:
        # ignore lldp packet
        return
    dst = eth.dst
    src = eth.src

    dpid = format(datapath.id, "d").zfill(16)

```

```

self.mac_to_port.setdefault(dpid, {})

# self.logger.info("packet in %s %s %s %s", dpid, src, dst, in_port)

# learn a mac address to avoid FLOOD next time.
self.mac_to_port[dpid][src] = in_port

if dst in self.mac_to_port[dpid]:
    out_port = self.mac_to_port[dpid][dst]
else:
    out_port = ofproto.OFPP_FLOOD

actions = [parser.OFPActionOutput(out_port)]

# install a flow to avoid packet_in next time
if out_port != ofproto.OFPP_FLOOD:
    match = parser.OFPMatch(in_port=in_port, eth_dst=dst)
    # verify if we have a valid buffer_id, if yes avoid to send both
    # flow_mod & packet_out
    if msg.buffer_id != ofproto.OFP_NO_BUFFER:
        self.add_flow(datapath, 1, match, actions, msg.buffer_id)
        return
    else:
        self.add_flow(datapath, 1, match, actions)
data = None
if msg.buffer_id == ofproto.OFP_NO_BUFFER:
    data = msg.data

out = parser.OFPPacketOut(datapath=datapath, buffer_id=msg.buffer_id,
                           in_port=in_port, actions=actions, data=data)
datapath.send_msg(out)

```

اکنون به توضیح بخش‌هایی از کد می‌پردازیم. برای این که کد پایتون تحت عنوان برنامه کاربردی در کنترلر Ryu شناخته شود باید کلاس اصلی برنامه از کلاس `ryu.base.app_manager.RyuApp` ارث‌بری گردد. بعلاوه، برای نگهداری جداول آدرس MAC، متغیر `mac_to_port` از نوع ساختار داده دیکشنری در پایتون تعریف شده است.

در کنترلر Ryu، با دریافت هر پیامی از پروتکل OpenFlow، یک رخداد متناظر با آن پیام تولید می‌گردد. در برنامه کنترلر Ryu، می‌توان با پیاده‌سازی یک هندلر رخداد دلخواه متناظر با پیام دریافتی، رفتار کنترلر و در نتیجه رفتار شبکه نرم‌افزار محور را تعیین نمود. یکی از راه‌های پیاده‌سازی هندلر رخداد در پایتون دکوراتورها^{۹۱} هستند که در کنترلر Ryu از دکوراتور `ryu.controller.handler.set_ev_cls` برای هندلر رخداد استفاده شده است. در متن برنامه، توابع `_packet_in_handler` و `switch_features_handler` توابعی هستند که هندلرهای

^{۹۱} Decorators

رخداد را پیاده‌سازی کرده‌اند و توسط دکوراتور `set_ev_cls` رفتار آن‌ها اصطلاحاً دکورات^{۹۲} شده است. دکوراتور `set_ev_cls` دو پارامتر دریافت می‌کند. یکی از این دو، کلاس رخداد مرتبط با پیام دریافتی و دیگری حالتی از سوئیچ `OpenFlow` (سوئیچ `whitebox`) می‌باشد.

آرگومان نام کلاس رخداد بطور کلی بصورت `OpenFlow + ryu.controller.ofp_event.EventOFP` `<message name>` می‌باشد. که در این کد برای پیام `Packet In` و پیام قابلیت‌های سوئیچ (`Switch Features`) کلاسهای `EventOFPSwitchFeatures` و `EventOFPPacketIn` مشخص شده‌اند. پیام `Packet In` راهی برای سوئیچ است تا بسته گرفته‌شده از شبکه را به کنترلر بفرستد. چنین کاری به دو علت ممکن است روی دهد: یکی این که بسته در جدول جریان `hit` شده باشد و عمل ارسال به کنترلر بطور صریح برای آن مطابقت مشخص گردد. دوم این که بسته در جدول جریان `miss` شده باشد و پردازش به کنترلر محول گردد [28].

پیام `Switch Features` مربوط به وقتی است که یک کانال انتقال (نظیر `TCP`، `SCTP`، `TLS`) بین سوئیچ و کنترلر مستقر شود. پس از استقرار کانال انتقال اولین کاری که انجام می‌گردد تشخیص و تعیین قابلیت‌های سوئیچ است. بدین منظور کنترلر پیامی تحت عنوان `FeatureReq` برای درخواست قابلیت‌های سوئیچ به آن می‌فرستد. پاسخ سوئیچ که قابلیت‌های سوئیچ را به اطلاع کنترلر می‌رساند در پیام `FeatureRes` به کنترلر برمی‌گردد [29].

آرگومان حالت سوئیچ، نشان‌دهنده یک یا چند حالت از فازهای مذاکره است که هندلر رخداد برای ورود به آن حالات نوشته می‌شود. جدول زیر شامل انواع حالات مربوط به فازهای مذاکره است:

نام حالت	توضیحات
<code>ryu.controller.handler.HANDSHAKE_DISPATCHER</code>	ارسال و انتظار برای پیام <code>hello</code>
<code>ryu.controller.handler.CONFIG_DISPATCHER</code>	انجام مذاکره نسخه و ارسال پیام <code>features-request</code>
<code>ryu.controller.handler.MAIN_DISPATCHER</code>	دریافت پیام <code>switch-features</code> و ارسال پیام <code>set-config</code>
<code>ryu.controller.handler.DEAD_DISPATCHER</code>	قطع اتصال از زوج، چه بصورت عمد چه غیر عمد در نتیجه یک ارور غیرقابل بازیابی

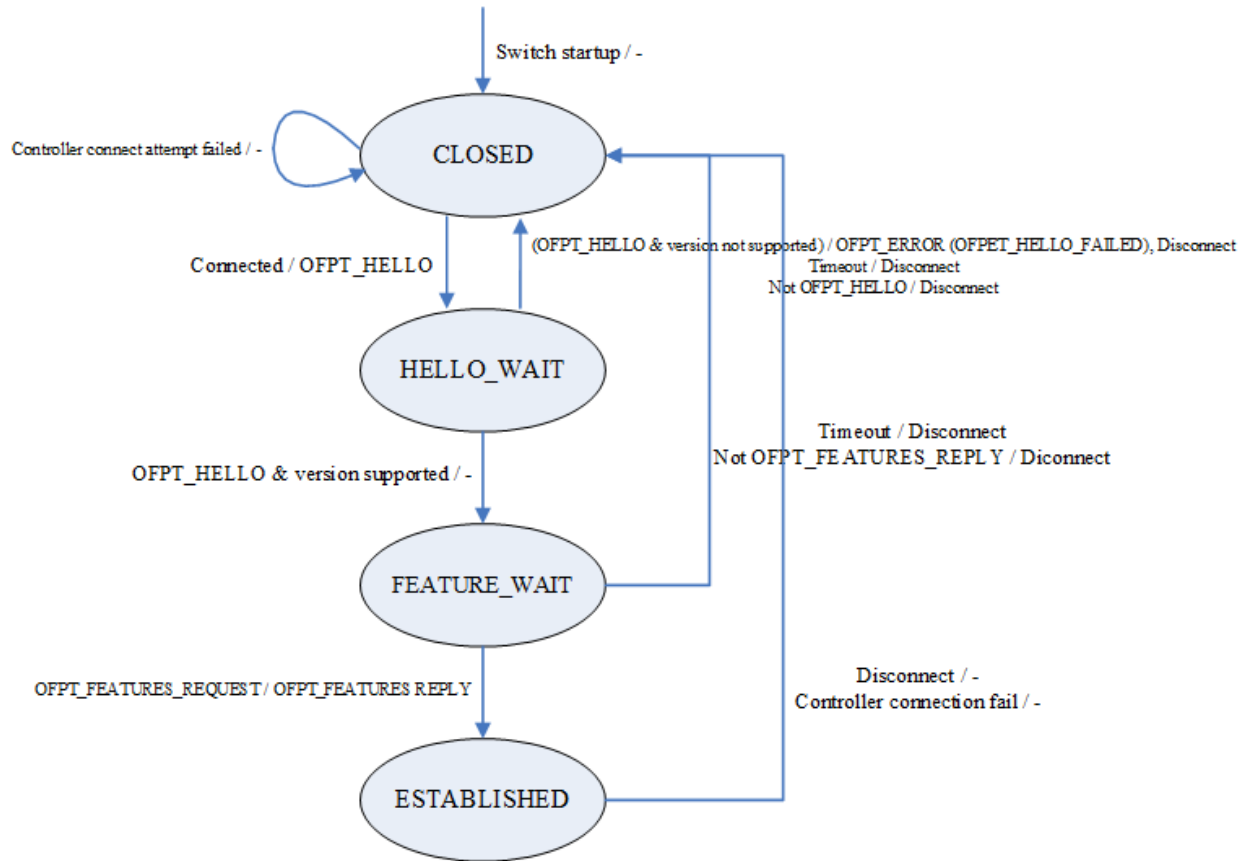
جدول 7 - حالات مختلف سوئیچ در ارتباط با کنترلر در پروتکل `OpenFlow` [30]

پروتکل اتصال کنترلر و سوئیچ با رسم شکل ماشین حالت بهتر دریافت می‌شود.

⁹² Decorate

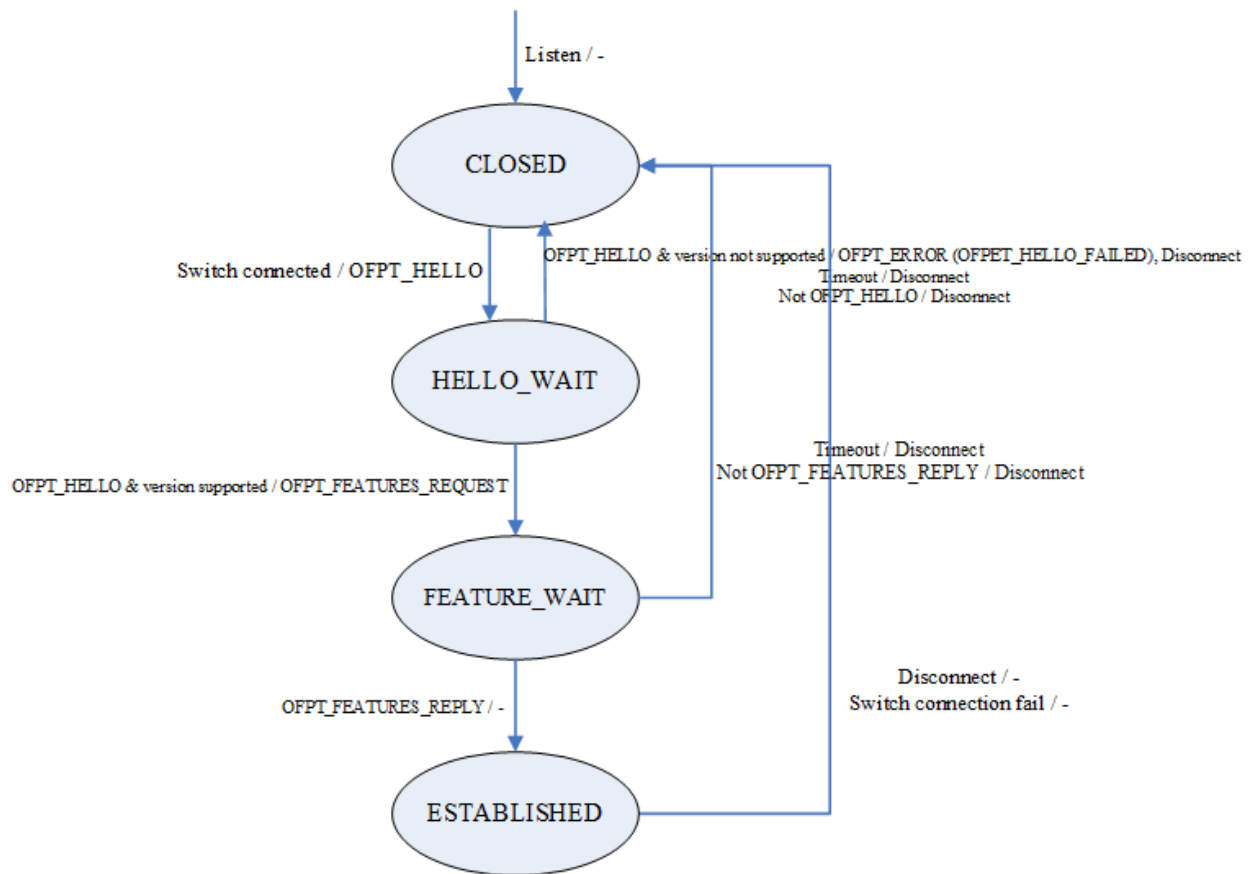
شکل زیر، ماشین حالات سوئیچ را در اتصال اصلی نشان می‌دهد. حالت اصلی حالت عادی اتصال سوئیچ با کنترلر است و در آن، برخی از حالات مربوط به فازهای مذاکره را مشاهده می‌کنید:

Switch, main connection



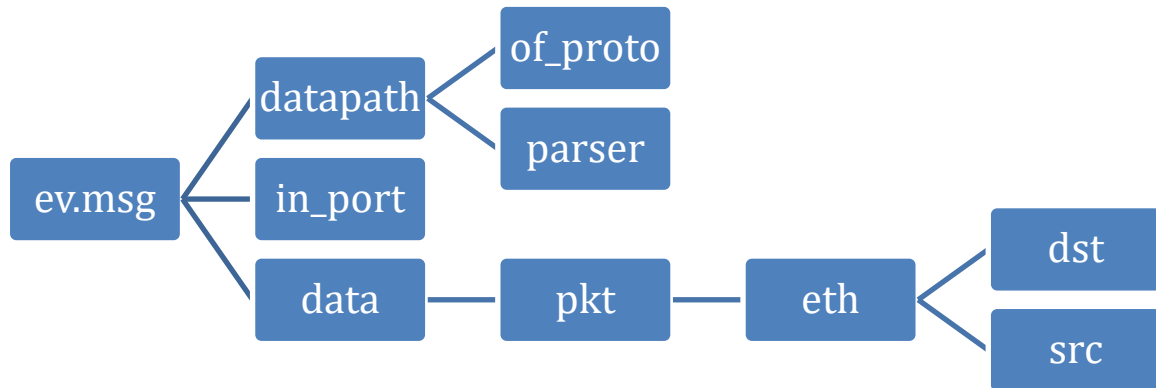
شکل 26 - ماشین حالت سوئیچ در حالت اتصال اصلی [31]

شکل زیر نیز ماشین حالات کنترلر را در اتصال اصلی نشان می‌دهد:



شکل 27 - ماشین حالات کنترلر در حالت اتصال اصلی [31]

شکل زیر بصورت نموداری، وابستگی متغیرها را نشان می‌دهد.



شکل 28 - نمودار وابستگی متغیرها در برنامه سوئیچ یادگیرنده

ev.msg حاوی یک نمونه از پیام پروتکل OpenFlow است. در متد switch_features_handler، ev.msg یک نمونه از کلاس ryu.ofproto.ofproto_v1_3_parser.OFPSwitchFeatures می‌باشد.

msg.datapath حاوی یک نمونه از کلاس ryu.controller.controller.Datapath می‌باشد که متناظر با سوئیچ OpenFlow است که این پیام را تولید کرده است. دستور add_flow در خط آخر این تابع، جریان miss را به جدول جریان سوئیچ مشخص شده با datapath اضافه می‌نماید. دقت کنید جریان miss، دارای اولویت 0 می‌باشد که آنرا کم اولویت‌تر از تمامی جریان‌های دیگر می‌کند. و این که بعلت مشخص نکردن هیچ قیدی در قسمت parser.OFPMatch() این جریان، با تمامی بسته‌ها مطابقت می‌کند. actions برای این جریان، ارسال بسته به پورت کنترلر است و بدین معناست که چنانچه بسته‌ای با جریان miss مطابقت پیدا کند، به کنترلر جهت بررسی فرستاده می‌شود. مقدار OFPCML_NO_BUFFER برابر max_len تنظیم شده است تا تمامی بسته‌های miss شده به کنترلر فرستاده شوند.

متد add_flow، پیام Flow Mod را می‌سازد و به datapath می‌فرستد. پیام Flow Mod از کنترلر به سوئیچ جهت تغییر وضعیت سوئیچ فرستاده می‌شود. کلاس OFPFlowMod مربوط به پیام Flow Mod می‌باشد. توسط متد send_msg است که پیام Flow Mod به سوئیچ ارسال می‌گردد.

در متد _packet_in_handler مهم‌ترین کاری که انجام می‌شود بروزرسانی جدول آدرس MAC می‌باشد. بدین منظور، پورت دریافت‌کننده (in_port) به همراه آدرس MAC مبدأ و مقصد که به ترتیب توسط eth.src و

eth.dst از eth استخراج شده‌اند در src و dst ذخیره می‌شوند (eth حاوی شناسه اترنت در کتابخانه بسته در Ryu می‌باشد).

پیش‌تر گفتیم که نگهداری جداول آدرس MAC توسط متغیر mac_to_port از نوع دیکشنری در پایتون می‌باشد. اکنون، لازم است توضیح را کمی دقیق‌تر کنیم. متغیر mac_to_port، یک دیکشنری تودرتو^{۹۳} می‌باشد. با توجه به این که dpid بیانگر یک سوئیچ می‌باشد، می‌توان از mac_to_port[dpid] بعنوان جدول آدرس MAC سوئیچی با شناسه dpid استفاده کرد. بدین‌صورت، mac_to_port[dpid] خود یک دیکشنری مثل d خواهد بود. هدف از ساخت این دیکشنری d آنست که هر کلید از این دیکشنری، یک آدرس MAC مثل A باشد که مقدار آن کلید، پورتی از سوئیچ مانند P را نشان دهد بطوریکه برای رسیدن بسته به مقصد A، بسته از پورت P خارج گردد. با داشتن in_port بعنوان پورت ورودی و همچنین src بعنوان آدرس MAC مبدأ، می‌توان انتصاب زیر را انجام داد:

```
self.mac_to_port[dpid][src] = in_port
```

دقت کنید که دیکشنری d همان mac_to_port[dpid] می‌باشد که خود مقداری برای کلید dpid از دیکشنری بزرگتر mac_to_port است که دیکشنری d را در خود دارد. پس می‌توان گفت mac_to_port یک دیکشنری تودرتو خواهد بود.

شرط `if dst in self.mac_to_port[dpid]`، وجود یا عدم وجود آدرس MAC مقصد یا همان dst را در جدول آدرس MAC مربوط به سوئیچ dpid بررسی می‌کند. در صورت وجود، پورت مربوط به آدرس MAC مقصد را بعنوان پورت خروجی انتخاب می‌کنیم. در غیر این‌صورت، با انجام عمل Flood (که توسط OFPP_FLOOD مشخص می‌شود) باعث می‌شویم بسته به‌مراه کپی‌هایی از آن به تمامی پورت‌ها غیر از in_port که پورت ورودی بود فرستاده شود.

و نهایتاً چنانچه عمل Flood رخ نداده باشد، یک مدخل جریان به جدول سوئیچ مشخص شده با datapath فرستاده می‌شود. همانطور که مشاهده می‌شود، اولویت برای این جریان، 1 است که آنرا بطور خودکار بالاتر از جریان miss با اولویت 0 قرار می‌دهد. ضمن این که مطابقت در چنین حالتی برخلاف جریان miss بی‌قید و شرط نیست و قیود مطابقت در این حالت، روی پورت ورودی (in_port) و آدرس MAC مقصد (dst) اعمال می‌شوند. بعبارتی دیگر می‌خواهیم جریانی در جدول جریان سوئیچ اضافه کنیم که باعث شود تمامی بسته‌هایی که از این پس روی آن پورت ورودی و از آن سوئیچ به سمت مقصد dst می‌خواهند بروند، با مطابقت با آن جریان

⁹³ Nested Dictionary

به سمت پورت خاصی از سوئیچ (که با out_port مشخص می‌شود) روانه شوند و بدینگونه از بروز عمل Flood جلوگیری شود.

دقت کنید که عمل add_flow در چنین حالتی (برای این جریان‌ها با اولویت 1 که جریان miss نیستند)، یکبار بیشتر انجام نمی‌شود و اینطور نیست که جریانی مثل f با dpid، in_port و src مشخص، دو بار در جدول جریان سوئیچ dpid ثبت گردد. چرا که پس از عمل add_flow برای جریان f، اکنون جریان f در جدول جریان سوئیچ dpid وجود دارد و دفعه بعدی که بسته‌ای از جریان f به سوئیچ dpid می‌رسد با همین جریان f موجود در جدول مطابقت پیدا می‌کند. پس دیگر پیام Packet In به کنترلر ارسال نمی‌گردد و متد add_flow نیز اجرا نخواهد شد. چرا که این پیام، در سناریوی ذکر شده تنها در صورت بروز miss به کنترلر فرستاده می‌شود. بررسی کد سوئیچ یادگیرنده در اینجا به پایان می‌رسد. برای اجرای آن، ابزار mininet را انتخاب کرده‌ایم که پس از آشنایی کلی و معرفی آن، با اجرای کد سوئیچ یادگیرنده این بخش به پایان خواهد رسید.

4.1.2 اجرای سوئیچ یادگیرنده در محیط mininet

بخطرات این که xterm از mininet شروع می‌شود، ابتدا باید با دستور mn محیط mininet را ایجاد کنیم. محیط mininet جهت اجرای سوئیچ یادگیرنده، ساختار ساده‌ای مشتمل بر 3 میزبان و 1 سوئیچ می‌باشد. پارامترهای دستور mn مورد استفاده مطابق جدول زیر می‌باشند [27]:

پارامتر	مقدار	توضیحات
topo	single, 3	توپولوژی شامل 3 میزبان و 1 سوئیچ تولید می‌کند.
mac	None	بطور خودکار برای میزبان‌ها آدرس MAC تخصیص می‌دهد.
switch	ovsk	از سوئیچ متن باز Open vSwitch استفاده می‌کند.
controller	remote	از کنترلر خارجی مبتنی بر OpenFlow استفاده می‌کند.
x	None	دستور را در ترمینال xterm اجرا می‌کند.

جدول 8 – پارامترهای دستور mn جهت ایجاد توپولوژی در محیط mininet

دستور mn را مطابق آنچه در جدول 5 گفته شد اجرا می‌کنیم:

```
mn -topo single,3 -mac -switch ovsk -controller remote -x
```

با اجرای این دستور 5 ترمینال xterm در میزبان آغاز می‌شوند که 3 تا از آن‌ها مربوط به 3 میزبان بوده و دوتای دیگر سوئیچ مجازی Open vSwitch و کنترلر می‌باشند.

با توجه به این که برنامه کنترلر با OpenFlow نسخه 1.3 نوشته شده است برای سازگاری با سوئیچ مورد استفاده، در ترمینال مربوط به سوئیچ (که با نام switch:s1 (root) مشخص شده است) دستور زیر را وارد می‌کنیم:

```
ovs-vsctl set bridge s1 protocols=OpenFlow13
```

برای نمایش جداول جریان سوئیچ s1 از دستور زیر استفاده می‌کنیم:

```
ovs-ofctl -O OpenFlow13 dump-flows s1
```

با توجه به این که هنوز کنترلر را اجرا نکرده‌ایم و هنوز ترافیکی از سوئیچ رد نشده است، جدول جریان در ابتدا مدخلی ندارد پس خروجی دستور به شکل زیر خواهد بود:

```
# ovs-ofctl -O OpenFlow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
#
```

شکل 29 - خروجی دستوری ovs-ofctl جهت مشاهده جدول جریان سوئیچ s1

اکنون نوبت به مهم‌ترین بخش کار یعنی اجرای برنامه کاربردی Ryu می‌رسد. بدین منظور، در ترمینال کنترلر (با نام (controller: c0 (root) دستور زیر را وارد می‌کنیم:

```
ryu-manager -verbose ryu.app.simple_switch_13
```

اتصال کنترلر به OVS کمی زمان می‌برد. آپشن -verbose باعث می‌شود جزئیات اضافی اجرای این دستور، در خروجی ترمینال نشان داده شود. در صورت موفقیت‌آمیز بودن اتصال، باید جریان miss مطابق برنامه به جدول جریان سوئیچ s1 (که در ابتدا خالی بود) اضافه شده باشد:

```
# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
cookie=0x0, duration=105.975s, table=0, n_packets=0, n_bytes=0, priority=0 actions=CONTROLLER
:65535
#
```

شکل 30 - جدول جریان سوئیچ s1 پس از اتصال موفقیت‌آمیز به کنترلر

اکنون نوبت آنست که با اجرای دستور ping از میزبان h1 به h2 عملکرد سوئیچ یادگیرنده را ارزیابی و از صحت عملکرد آن اطمینان حاصل کنیم. پیش از اجرای دستور ping، بهتر است در هر یک از میزبان‌های h1، h2 و h3 دستور tcpdump را اجرا کنیم که بسته‌های دریافتی توسط هر میزبان را بررسی می‌کند:

Host h1:

```
tcpdump -en -i h1-eth0
```

Host h2:

```
tcpdump -en -i h2-eth0
Host h3:
tcpdump -en -i h3-eth0
```

آپشن -i برای مشخص کردن واسط به کار می‌رود. وجود h1-eth0 پس از آن، به معنای آنست که فقط بسته‌های عبوری از واسط h1-eth0 از میزبان h1 توسط دستور tcpdump مورد بررسی قرار می‌گیرند.

حال برای اجرای دستور ping کفایت به ترمینال xterm اولیه که در آن دستور mn را اجرا کردیم برویم و آنجا جلوی mininet> در ابتدای خط فرمان، دستور h2 ping -c1 h1 را وارد نماییم. با این کار، یک بسته ICMP echo request از h1 به h2 ارسال و پاسخ آن که ICMP echo reply باشد از h2 به h1 برمی‌گردد. ولی در ابتدای کار بعثت نامعلوم بودن آدرس MAC میزبان h2 برای h1، لازم است قبل از ICMP echo request، یک ARP request ارسال و همه‌پختی شود. این بسته ARP همه‌پختی شده توسط h2 و h3 دریافت شده و پاسخ آن از طرف h2 تحت عنوان ARP reply به h1 برمی‌گردد. نهایتاً توسط پیام ARP reply است که h1 از آدرس MAC میزبان h2 مطلع می‌شود و نوبت به بسته‌های ICMP می‌رسد. دقت کنید h3 صرفاً بسته ARP request را از h1 دریافت می‌کند و دخالت دیگری در این سناریو ندارد. آپشن -c1 باعث می‌شود تنها یک بسته ICMP ارسال و دریافت شود.

اکنون، مجدداً جدول جریان سوئیچ s1 را مورد بررسی قرار می‌دهیم:

```
# ovs-ofctl -O openflow13 dump-flows s1
OFPST_FLOW reply (OF1.3) (xid=0x2):
 cookie=0x0, duration=417.838s, table=0, n_packets=3, n_bytes=182, priority=0 actions=
CONTROLLER:65535
 cookie=0x0, duration=48.444s, table=0, n_packets=2, n_bytes=140, priority=1,in_port=2,dl_dst
=00:00:00:00:00:01 actions=output:1
 cookie=0x0, duration=48.402s, table=0, n_packets=1, n_bytes=42, priority=1,in_port=1,dl_dst
=00:00:00:00:00:02 actions=output:2
#
```

شکل 31 – جدول جریان سوئیچ s1 پس از انجام ping

جریان اول، همان جریان miss است؛ ولی اکنون دو جریان دیگر هم به جدول جریان s1 اضافه شده‌اند. توصیف این دو جریان بصورت زیر است:

- چنانچه روی پورت شماره 2 بسته‌ای به مقصد (dl_dst) مربوط به h1 دریافت شود (in_port=2) این بسته باید به پورت شماره 1 ارسال گردد.
- چنانچه روی پورت شماره 1 بسته‌ای به مقصد (dl_dst) مربوط به h2 دریافت شود (in_port=1) این بسته باید به پورت شماره 2 ارسال گردد.

فیلد `n_packets` برای هر جریان تعداد مطابقت‌هایی را نشان می‌دهد که از زمان ایجاد آن جریان بین آن جریان و بسته‌های عبوری از شبکه رخ داده‌اند.

با بررسی کوتاهی از `n_packets` این بخش و در واقع این فصل را به پایان می‌بریم. ابتدا برای جلوگیری از سردرگمی نام‌های زیر را برای پیام‌ها و همچنین جریان‌های رد و بدل شده در سناریوی فوق در نظر می‌گیریم تا در ادامه آن‌ها را با نام‌هایشان مورد بررسی قرار دهیم: پیام `ARP request` را `m1`، `ARP reply` را `m2`، `ICMP echo request` را `m3`، `ICMP echo reply` را `m4` در نظر می‌گیریم. ضمناً جریان `miss` را جریان `f0` و بین دو جریان توصیف شده فوق، جریان اول (ارسال به پورت 1) را `f1` و جریان دوم (ارسال به پورت 2) را `f2` در نظر می‌گیریم. نهایتاً `fm0`، `fm1` و `fm2` به ترتیب نشان‌دهنده مطابقت با جریان‌های `miss`، `f1` و `f2` می‌باشند.

با فرستادن پیام `m1`، بعلت خالی بودن جدول جریان، یک مطابقت `fm0` رخ می‌دهد. دقت کنید متد `add_flow` وقتی صدا زده می‌شود که مقصد از نوع همه‌پخشی نباشد (بعبارتی Flood رخ نداده باشد). بنابراین با عبور پیام `m1` درست است که دیکشنری `mac_to_port` رکورد مربوط به تناظر آدرس `MAC` میزبان `h1` با پورت شماره 1 سوئیچ `s1` را ایجاد می‌کند ولی بعلت همه‌پخشی و Flood، متد `add_flow` صدا زده نمی‌شود و جدول جریان کماکان خالی است (فقط جریان `miss` را دارد).

با فرستادن پیام `m2`، با توجه به این که جدول جریان کماکان خالی است، یک `miss` رخ می‌دهد که مجدداً مطابقت `fm0` است. ولی در کنترلر، چون دیکشنری `mac_to_port` رکورد مربوط به میزبان `h1` را دارد، همه‌پخشی و Flood دیگر رخ نمی‌دهد و با اجرای متد `add_flow`، جریان `f1` به جدول جریان `s1` اضافه می‌گردد.

در ادامه با پیام `m3`، با توجه به این که جدول جریان، جریان `f2` را ندارد مجدداً مطابقت `fm0` با جریان `f0` رخ می‌دهد. ولی در کنترلر چون دیکشنری `mac_to_port` رکورد مربوط به میزبان `h2` را دارد، همه‌پخشی و Flood دیگر رخ نمی‌دهد و با اجرای متد `add_flow`، جریان `f2` به جدول جریان `s1` اضافه می‌گردد.

نهایتاً پیام `m4`، با جریان `f1` مطابقت `fm1` را ایجاد می‌کند و در اینجا دیگر `Packet In` نداریم. پس به کنترلر ارجاع داده نمی‌شود و از پورت 1 سوئیچ `s1` به سمت میزبان `h1` می‌رود.

بنظر می‌رسد ایجاد جریان `f0` یا همان `miss`، ارجاع محسوب نمی‌شود ولی ایجاد جریان‌های `f1` و `f2` ارجاع هم محسوب می‌گردند. با این تفاسیر، تعداد ارجاعات جریان‌ها که با `n_packets` مشخص می‌شود برای جریان `f0` برابر 3 (3 بار `fm0`)، برای جریان `f1` برابر 2 (1 بار هنگام ایجاد و 1 بار `fm1`) و برای جریان `f2` برابر 1 (فقط 1 بار هنگام ایجاد) خواهند بود.

5 فصل پنجم: پیاده‌سازی و اجرای monitor

5.1 بررسی و اجرای کد monitor

کد monitor در فایل simple_monitor_13.py نوشته شده است که متن آن در ذیل آمده است:

```
# Copyright (C) 2016 Nippon Telegraph and Telephone Corporation.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.

from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import CONFIG_DISPATCHER, MAIN_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet

from operator import attrgetter

from ryu.app import simple_switch_13
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.lib import hub
import json
import logging
import requests
from logging import Logger

'''
class SimpleMonitor13(simple_switch_13.SimpleSwitch13):
    url_port = 'http://monitorproj.test/portstats/'
    url_flow = 'http://monitorproj.test/flowstats/'
    def __init__(self, *args, **kwargs):
        super(SimpleMonitor13, self).__init__(*args, **kwargs)

        self.datapaths = {}
```

```

logging.basicConfig(filename='/home/alireza-a/monitor.log',encoding='utf-8')
fh = logging.FileHandler('/home/alireza-a/monitor.log')
fh.setLevel(logging.DEBUG)
self.logger.addHandler(fh)

self.monitor_thread = hub.spawn(self._monitor)

@set_ev_cls(ofp_event.EventOFPStateChange,
            [MAIN_DISPATCHER, DEAD_DISPATCHER])
def _state_change_handler(self, ev):
    datapath = ev.datapath
    if ev.state == MAIN_DISPATCHER:
        if datapath.id not in self.datapaths:
            # self.logger.debug('register datapath: %016x', datapath.id)
            self.datapaths[datapath.id] = datapath
    elif ev.state == DEAD_DISPATCHER:
        if datapath.id in self.datapaths:
            # self.logger.debug('unregister datapath: %016x', datapath.id)
            del self.datapaths[datapath.id]

def _monitor(self):
    while True:
        for dp in self.datapaths.values():
            self._request_stats(dp)
            hub.sleep(10)

def _request_stats(self, datapath):
    # self.logger.debug('send stats request: %016x', datapath.id)
    ofproto = datapath.ofproto
    parser = datapath.ofproto_parser

    req = parser.OFPFlowStatsRequest(datapath)
    datapath.send_msg(req)

    req = parser.OFPPortStatsRequest(datapath, 0, ofproto.OFPP_ANY)
    datapath.send_msg(req)

@set_ev_cls(ofp_event.EventOFPFlowStatsReply, MAIN_DISPATCHER)
def _flow_stats_reply_handler(self, ev):
    body = ev.msg.body
    for stat in sorted([flow for flow in body if flow.priority == 1],
                      key=lambda flow: (flow.match['in_port'],
                                         flow.match['eth_dst'])):
        # Flow Stats in JSON
        monitor_dict = {
            "datapath": str(ev.msg.datapath.id),
            "in_port": str(stat.match['in_port']),
            "eth_dst": stat.match['eth_dst'],
            "out_port": str(stat.instructions[0].actions[0].port),
            "packets": str(stat.packet_count),
            "bytes": str(stat.byte_count)
        }
        x = requests.post(self.url_flow, data=monitor_dict)
        self.logger.info(x.text)
        self.logger.info('%s', monitor_dict)

```

```

self.logger.info('%s', json.dumps(monitor_dict))

@set_ev_cls(ofp_event.EventOFPPortStatsReply, MAIN_DISPATCHER)
def _port_stats_reply_handler(self, ev):
    body = ev.msg.body
    for stat in sorted(body, key=attrgetter('port_no')):
        # Port Stats in JSON
        monitor_dict={"datapath" : ev.msg.datapath.id,
                     "port" : str(stat.port_no),
                     "rx_pkts": str(stat.rx_packets),
                     "rx_bytes": str(stat.rx_bytes),
                     "rx_error": str(stat.rx_errors),
                     "tx_pkts": str(stat.tx_packets),
                     "tx_bytes": str(stat.tx_bytes),
                     "tx_error": str(stat.tx_errors)}
        x = requests.post(self.url_port, data=monitor_dict)
        self.logger.info(x.text)
        self.logger.info('%s', monitor_dict)

```

اکنون به توضیح بخش‌هایی از کد می‌پردازیم. پیش‌تر گفته بودیم با استفاده از سوئیچ یادگیرنده، اقدام به پیاده‌سازی monitor می‌کنیم. در کد پایتون، این کار را با استفاده از مفهوم ارث‌بری انجام داده‌ایم. عبارت `class SimpleMonitor13(simple_switch_13.SimpleSwitch13)` باعث می‌شود که کلاس `SimpleMonitor13` که در آن عملکرد monitor پیاده‌سازی شده است، از کلاس `SimpleSwitch13` ارث‌بری کند. با این کار، متغیرها و متدهای سوئیچ یادگیرنده که در فایل `simple_switch_13.py` و کلاس `SimpleSwitch13` قابل دسترسی و انجام بودند در monitor نیز قابل دسترسی و اجرا خواهند شد. حال با تعریف متغیرها و پیاده‌سازی متدهای جدید در `SimpleMonitor` باعث می‌شویم monitor در سلسله مراتب ارث‌بری نوعی سوئیچ یادگیرنده باشد که علاوه بر رفتار سوئیچ یادگیرنده، کار مانیتورینگ محیط‌های مجازی را نیز انجام می‌دهد.

در متد `_init_` و به محض نمونه‌سازی از این کلاس (که توسط کنترلر انجام می‌شود) یک رشته^{۹۴} تولید می‌گردد که به موازات برنامه اصلی، بصورت متناوب به سوئیچ‌های متصل به کنترلر، درخواست می‌فرستد تا اطلاعات آماری آن‌ها را استخراج کند. چنین کاری در پایتون راه‌های مختلفی دارد که در این پروژه از متد `hub.spawn()` جهت موازی‌سازی استفاده کرده‌ایم.

در اینجا، متدی که بموازات برنامه اصلی اجرا می‌شود متد `_monitor()` است که هر 10 ثانیه یکبار به تمام سوئیچ‌های حاضر در صفحه داده توسط متد `_request_stats()` درخواست می‌فرستد تا وضعیت آن‌ها را مانیتور کند. البته در این پروژه فقط 1 سوئیچ با نام `mybridge` مورد بررسی قرار گرفته ولی می‌توان عملکرد چند

⁹⁴ Thread

سوئیچ را با هم مورد بررسی قرار داد. بعنوان مثال می‌توان سوئیچ s1 را که در محیط mininet ایجاد شده بود نیز در کنار سوئیچ mybridge مانیتور نمود.

برای هندل کردن قطع یا وصل شدن اتصال (اتصال سوئیچ‌ها) با کنترلر، از رخداد EventOFPSStateChange استفاده شده که بهنگام تغییر وضعیت صفحه داده تولید می‌شود. همانطور که می‌بینید کماکان از دکوراتور set_ev_cls برای هندل کردن رخداد استفاده شده که اینجا هندل کردن این رخداد به کمک متد اصلی _state_change_handler() انجام گرفته است. در این متد، به کمک متغیر datapaths که از نوع دیکشنری تعریف شده یک دیکشنری از سوئیچ‌های متصل نگهداری می‌شود. با قطع یا وصل شدن هر سوئیچ از کنترلر یا به آن، یک رخداد EventOFPSStateChange تولید می‌گردد. چنانچه این رخداد، اتصال سوئیچ باشد وضعیت به MAIN_DISPATCHER تغییر پیدا کرده و در صورت عدم حضور شناسه مربوط به آن سوئیچ در datapaths، آن سوئیچ به datapaths اضافه خواهد شد. اما چنانچه این رخداد، قطع اتصال سوئیچ باشد وضعیت به DEAD_DISPATCHER تغییر پیدا کرده و شناسه آن سوئیچ از datapaths حذف خواهد شد.

در متد _request_stats() که بصورت تناوبی در متد _monitor() صدا زده می‌شود، دو نوع درخواست به سمت هر سوئیچ متصل فرستاده می‌شود [32]:

- OFPFlowStatsRequest: از سوئیچ درخواست می‌کند تا اطلاعات آماری مربوط به جریان‌ها را در پاسخ ارسال کند.

- OFPPortStatsRequest: از سوئیچ درخواست می‌کند تا اطلاعات آماری مربوط به پورت‌ها را در پاسخ ارسال کند. در این جا از OFPP_ANY استفاده شده که باعث می‌شود اطلاعات آماری تمامی پورت‌های سوئیچ در پاسخ ارسال گردد.

حال در پاسخ این دو نوع درخواست، پیام‌های FlowStatsReply و PortStatsReply تولید می‌شوند که رخدادهای متناظر با آن‌ها به ترتیب EventOFPPortStatsReply و EventOFPFlowStatsReply می‌باشند. بعلت مشابه بودن هندلر این دو رخداد با یکدیگر فقط هندلر EventOFPFlowStatsReply را مختصراً بررسی می‌کنیم که در متد _flow_stats_reply_handler پیاده‌سازی شده است.

برای نمایش خروجی، تمامی جریان‌ها غیر از جریان miss را بصورت مرتب‌شده در کنسول نمایش داده و همچنین به فرمت JSON⁹⁵ به url صفحه وب مربوط به وضعیت جریان‌ها فرستاده‌ایم. این url به همراه

⁹⁵ JavaScript Object Notation

وضعیت پورت‌ها، مربوط به واسط وب می‌باشند که نتایج کنترلر را بصورت JSON گرفته در جدول نمایش می‌دهند.

برای هر جریان، مهم‌ترین فیلدها را برای نمایش انتخاب کرده‌ایم. نام هر فیلد به اندازه کافی گویای مطالب هست؛ بنابراین از توضیحات بیش از اندازه خودداری می‌کنیم. مطلبی که گفتن آن شاید خالی از لطف نباشد آنست که برای مرتب‌سازی در پایتون، جریان‌ها بخودی خود ترتیب خاصی ندارند. در چنین شرایطی که اعضای یک لیست بخودی خود قابل مرتب‌سازی نیستند، یک راه‌حل آن است که تابعی بنویسیم که هر عضو از آن لیست را به موجودیتی از نوع قابل مقایسه تبدیل کند. سپس برای مقایسه اعضای لیست، از مقایسه موجودیت‌های نگاشت شده توسط آن تابع استفاده کنیم که نقش کلید مرتب‌سازی را ایفا می‌کنند. یکی از راه‌های انجام چنین کاری در پایتون (و برخی زبان‌های دیگر برنامه‌نویسی) از طریق Lambda Calculus می‌باشد. در اینجا در قسمت key مربوط به متد sorted() چنین تابعی با Lambda Calculus مشخص شده است. این تابع، هر جریان از لیست جریان‌ها را گرفته و به ازای آن یک دوتایی مرتب بصورت (a,b) برمی‌گرداند که در آن a پورت ورودی و b آدرس MAC مربوط به مقصد می‌باشد. با توجه به اینکه چنین دوتایی مرتبی قابلیت مرتب‌سازی دارد (ابتدا به ترتیب مؤلفه اول یا همان شماره پورت، و سپس در شماره پورت‌های یکسان به ترتیب مؤلفه دوم یا همان آدرس MAC) لیست جریان‌ها را بدین‌صورت مرتب می‌کنیم.

پیش از اجرای کنترلر، لازم است url های مربوط به واسط وب پروژه را در مرورگر سیستم میزبان (در اینجا فایرفاکس) وارد کرده آن صفحات را از قبل بصورت آماده و بارگذاری شده داشته باشیم. چرا که نتایج نهایی مدنظر در این پروژه بصورت جدول در آن صفحات ایجاد می‌گردند. این صفحات، هر 5 ثانیه یکبار بارگذاری مجدد می‌شوند.

5.1.1 اجرا برای مانیتورینگ ترافیک ماشین‌های مجازی

در بخش قبل، سوئیچ s1 از محیط mininet را نیز به کنترلر وصل کرده بودیم. پس چنانچه کنترلر را اجرا کنیم، کنترلر به مانیتورینگ توأمان سوئیچ‌های s1 و mybridge می‌پردازد. با توجه به این که سوئیچ s1 در محیط mininet بیشتر جهت یادگیری و تست اولیه سوئیچ یادگیرنده ایجاد شده بود، حضور نتایج مانیتورینگ سوئیچ s1 در کنار سوئیچ mybridge که سوئیچ اصلی به کار رفته در پروژه می‌باشد باعث شلوغی بیش از اندازه و سردرگمی بی‌فایده می‌گردد. توصیه می‌شود جهت جلوگیری از سردرگمی ابتدا اتصال سوئیچ s1 را با کنترلر قطع نموده سپس به اتصال کنترلر با سوئیچ mybridge بپردازیم.

دقت کنید کنترلر در اینجا، همان monitor می‌باشد که برنامه پایتون مربوط به آن در فایل simple_monitor_13.py بررسی شد. کنترلر وظیفه مانیتورینگ بین ماشین‌های مجازی را بعهده دارد. ابتدا با دستور زیر، ارتباط سوئیچ s1 را با کنترلر قطع می‌کنیم:

```
ovs-vsctl del-controller s1
```

برای اتصال کنترلر به سوئیچ mybridge که در سیستم میزبان قرار دارد دستور زیر را وارد می‌کنیم:

```
ovs-vsctl set-controller mybridge tcp:<Monitor IP>
```

بعلت سطح بالا نبودن سیستم، برای مانیتور که همان monitor یا کنترلر است یک ماشین مجازی دیگر در نظر نگرفته‌ایم. بنابراین در این پروژه IP سیستمی که مانیتور روی آن اجرا می‌شود همان localhost با آدرس 127.0.0.1 می‌باشد. چنانچه مانیتور روی ماشین مجازی یا سیستم جداگانه‌ای اجرا می‌شد آدرس IP آن ماشین مجازی یا سیستم را در <Monitor IP> وارد می‌کردیم.

با دستور زیر کد پایتونی که برای کنترلر Ryu ایجاد کرده‌ایم اجرا می‌نماییم:

```
ryu-manager -verbose <Absolute Path to Ryu>/ryu/ryu/app/simple_monitor_13.py
```

برای سادگی بیشتر، فایل simple_monitor_13.py را در همان دیرکتوری app از کنترلر ryu نگه می‌داریم که با نصب کنترلر ایجاد شده است. در این دیرکتوری برخی از برنامه‌های از پیش نوشته شده کنترلر ryu قرار دارد که برنامه simple_switch_13.py که در رابطه با سوئیچ یادگیرنده در فصل قبل بررسی شد یکی از آنها می‌باشد.

حال، چنانچه اتصالات مربوط به ماشین‌های مجازی VM-A و VM-B مطابق بخش 4.1.2 انجام شده باشد، با دستور ifconfig در هر یک از ماشین‌های مجازی، می‌توان آدرس IP آن ماشین مجازی را استخراج کرد. روند کار برای ماشین مجازی VM-B در شکل قابل مشاهده است:

```

Mininet-B [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
mininet@mininet-vm:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 08:00:27:a4:54:fc
          inet addr:192.168.1.7  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:368168  errors:0  dropped:0  overruns:0  frame:0
          TX packets:117491  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:27086010 (27.0 MB)  TX bytes:11378654 (11.3 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0  errors:0  dropped:0  overruns:0  frame:0
          TX packets:0  errors:0  dropped:0  overruns:0  carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

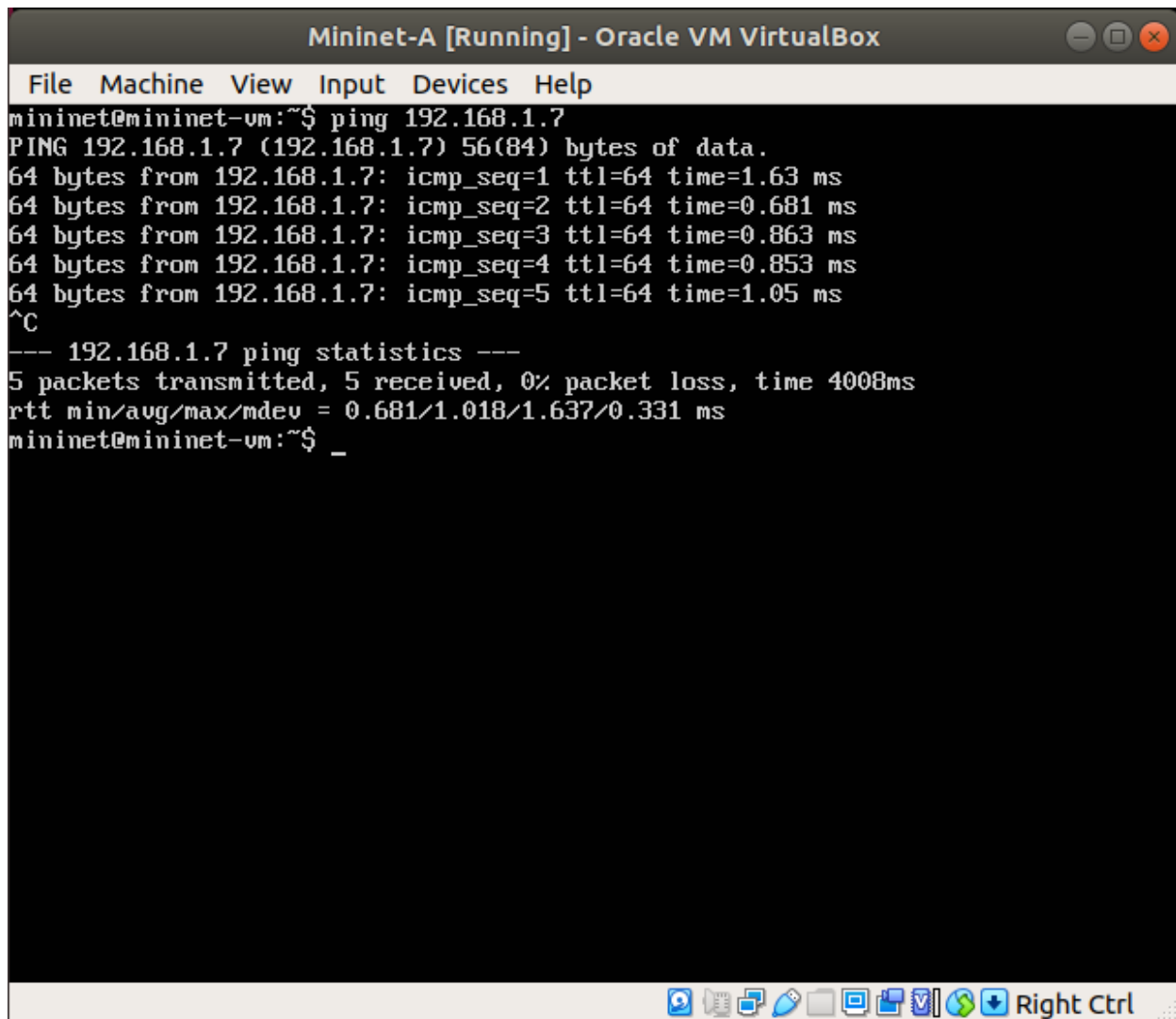
mininet@mininet-vm:~$

```

شکل 32 - خروجی دستور ifconfig برای استخراج آدرس IP ماشین مجازی VM-B

همانطور که از شکل نیز پیداست، در قسمت eth0 آدرس IP مربوط به VM-B برابر 192.168.1.7 مشخص شده است. اکنون از سمت ماشین مجازی VM-A، ماشین مجازی VM-B را ping می‌کنیم. با این کار، دائماً بسته‌های ICMP echo request از VM-A به VM-B فرستاده می‌شوند که جواب آن‌ها نیز بصورت ICMP echo reply به VM-A برمی‌گردد.

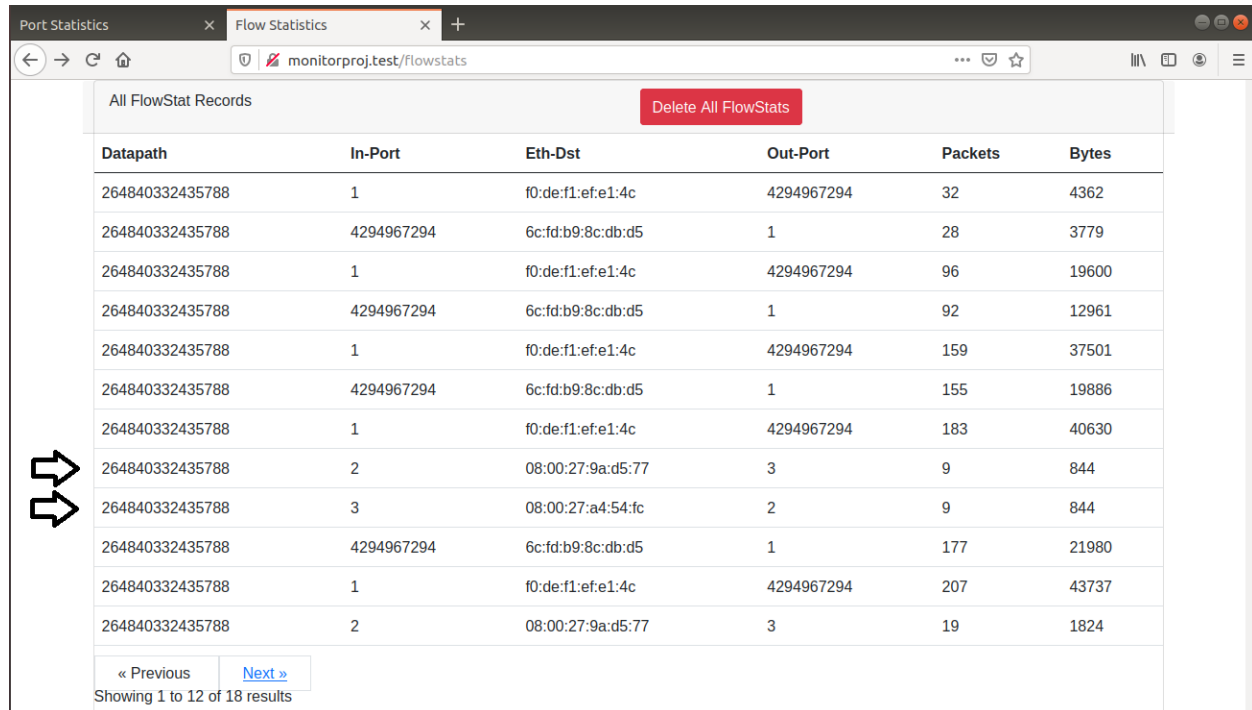
نهایتاً پس از دریافت تعدادی از ICMP echo reply ها، با فشردن ترکیب Ctrl + C و ارسال سیگنال وقفه، به کار دستور ping خاتمه می‌دهیم. خروجی، مشابه شکل زیر خواهد شد:



```
Mininet-A [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
mininet@mininet-vm:~$ ping 192.168.1.7
PING 192.168.1.7 (192.168.1.7) 56(84) bytes of data.
64 bytes from 192.168.1.7: icmp_seq=1 ttl=64 time=1.63 ms
64 bytes from 192.168.1.7: icmp_seq=2 ttl=64 time=0.681 ms
64 bytes from 192.168.1.7: icmp_seq=3 ttl=64 time=0.863 ms
64 bytes from 192.168.1.7: icmp_seq=4 ttl=64 time=0.853 ms
64 bytes from 192.168.1.7: icmp_seq=5 ttl=64 time=1.05 ms
^C
--- 192.168.1.7 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4008ms
rtt min/avg/max/mdev = 0.681/1.018/1.637/0.331 ms
mininet@mininet-vm:~$ _
```

شکل 33 - خروجی دستور ping و خاتمه آن توسط وقفه

اکنون مدتی صبر می‌کنیم. یادآوری می‌کنیم عمل مانیتورینگ هر 10 ثانیه یکبار انجام می‌شود. پس از گذشت مدتی، صفحه وب مربوط به اطلاعات آماری جریان‌ها بصورت زیر در خواهد آمد:



Datapath	In-Port	Eth-Dst	Out-Port	Packets	Bytes
264840332435788	1	f0:de:f1:ef:e1:4c	4294967294	32	4362
264840332435788	4294967294	6c:fd:b9:8c:db:d5	1	28	3779
264840332435788	1	f0:de:f1:ef:e1:4c	4294967294	96	19600
264840332435788	4294967294	6c:fd:b9:8c:db:d5	1	92	12961
264840332435788	1	f0:de:f1:ef:e1:4c	4294967294	159	37501
264840332435788	4294967294	6c:fd:b9:8c:db:d5	1	155	19886
264840332435788	1	f0:de:f1:ef:e1:4c	4294967294	183	40630
264840332435788	2	08:00:27:9a:d5:77	3	9	844
264840332435788	3	08:00:27:a4:54:fc	2	9	844
264840332435788	4294967294	6c:fd:b9:8c:db:d5	1	177	21980
264840332435788	1	f0:de:f1:ef:e1:4c	4294967294	207	43737
264840332435788	2	08:00:27:9a:d5:77	3	19	1824

شکل 34 - صفحه وب مربوط به اطلاعات آماری جریان‌ها در محیط ماشین مجازی

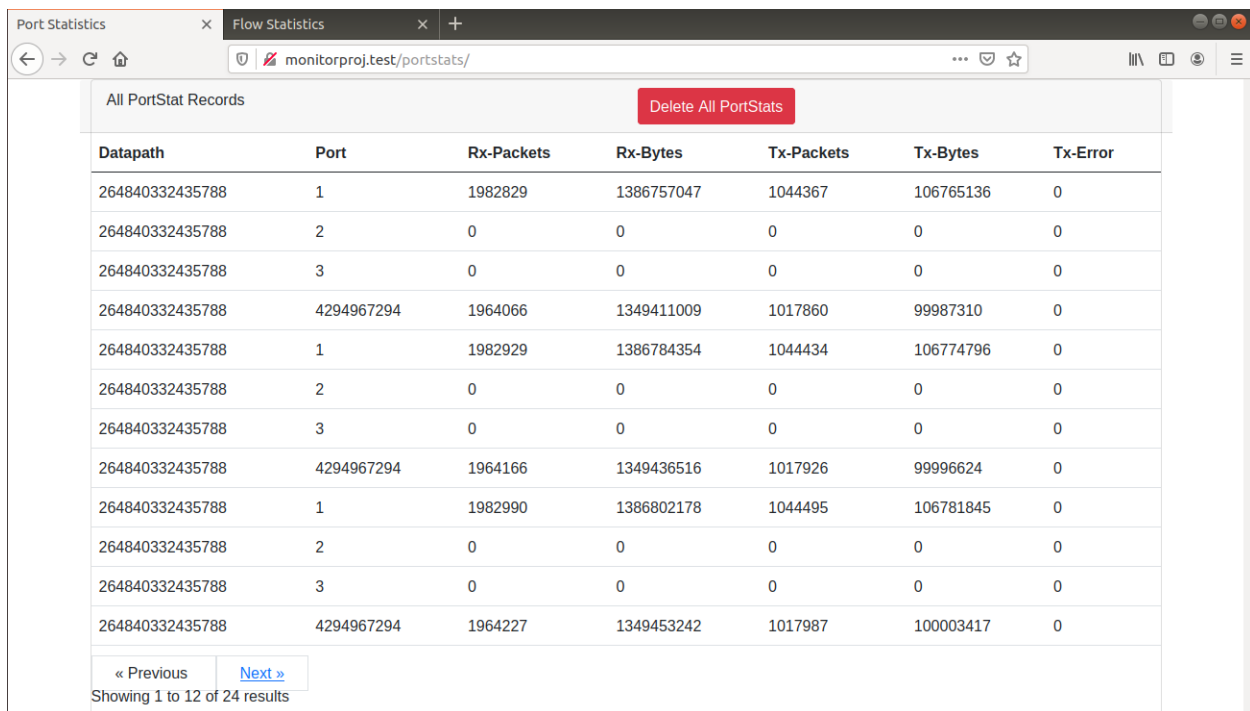
درباره این شکل ذکر نکاتی حائز اهمیت است. بنظر می‌رسد علاوه بر ترافیک مربوط به ping، ترافیک‌های دیگری هم بین ماشین‌های مجازی در virtualbox رد و بدل می‌شوند زیرا در حالتی که هنوز ping نگرفته‌ایم نیز سوئیچ mybridge جریان‌هایی غیر از جریان miss دارد. اجرای مجدد این سناریو در محیط mininet در بخش بعد اثباتی بر این مدعاست.

رکوردهای جدول از هر in_port=1 تا in_port=1 بعدی بیانگر یک بار مانیتورینگ اطلاعات آماری جریان‌های شبکه است. دقت کنید رکوردهای جدول (همان جریان‌ها) برای هر سوئیچ (البته اینجا تنها یک سوئیچ موجود است) ابتدا به ترتیب in_port و سپس به ترتیب eth_dst مرتب شده‌اند.

پورت 2 از سوئیچ متناظر با ماشین مجازی VM-B و پورت 3 از سوئیچ متناظر با ماشین مجازی VM-A می‌باشد. این استدلال بر طبق آدرس MAC صورت گرفته است. چرا که در جریان‌ها جایی که آدرس MAC مقصد 08:00:27:a4:54:fc بوده بسته از پورت 2 خارج شده است. حال آن که این آدرس MAC همانطور که در شکل مربوط به ifconfig در خروجی این دستور دیدیم مربوط به VM-B بوده است.

از شکل معلوم می‌شود تا زمانیکه ping را انجام نداده بودیم، جریان‌های موجود در جدول جریان سوئیچ mybridge صرفاً ناشی از ترافیک‌های جانبی بین ماشین‌های مجازی و VirtualBox یا سیستم میزبان ایجاد شده بودند. ولی پس از ping کردن جریان‌های دیگری نیز به جریان‌های موجود اضافه می‌شوند. نخستین دفعه ایجاد این جریان‌های جدید ناشی از ping را در شکل با فلش مشخص کرده‌ایم. طبیعتاً دو جریان وجود خواهند داشت که بالایی مربوط به فرستادن بسته به پورت 3 و VM-A و پایینی مربوط به فرستادن بسته به پورت 2 و VM-B می‌باشد.

نهایتاً صفحه وب مربوط به اطلاعات آماری پورت‌ها نیز بصورت زیر خواهد بود:



The screenshot shows a web browser window with the address `monitorproj.test/portstats/`. The page title is "Port Statistics" and it contains a table of network statistics. The table has columns: Datapath, Port, Rx-Packets, Rx-Bytes, Tx-Packets, Tx-Bytes, and Tx-Error. There are 12 rows of data. A "Delete All PortStats" button is visible at the top right of the table. Navigation links "« Previous" and "Next »" are at the bottom left, along with the text "Showing 1 to 12 of 24 results".

Datapath	Port	Rx-Packets	Rx-Bytes	Tx-Packets	Tx-Bytes	Tx-Error
264840332435788	1	1982829	1386757047	1044367	106765136	0
264840332435788	2	0	0	0	0	0
264840332435788	3	0	0	0	0	0
264840332435788	4294967294	1964066	1349411009	1017860	99987310	0
264840332435788	1	1982929	1386784354	1044434	106774796	0
264840332435788	2	0	0	0	0	0
264840332435788	3	0	0	0	0	0
264840332435788	4294967294	1964166	1349436516	1017926	99996624	0
264840332435788	1	1982990	1386802178	1044495	106781845	0
264840332435788	2	0	0	0	0	0
264840332435788	3	0	0	0	0	0
264840332435788	4294967294	1964227	1349453242	1017987	100003417	0

شکل 35 – صفحه وب مربوط به اطلاعات آماری پورت‌ها در محیط ماشین مجازی

همانطور که در شکل می‌بینید 4 پورت روی سوئیچ mybridge وجود دارند. متأسفانه برخلاف انتظار روی پورت‌های 2 و 3 سوئیچ که به ترتیب متناظر با VM-B و VM-A می‌باشند هیچ ترافیکی ثبت نشده است. بنظر می‌رسد این نتیجه دور از انتظار، باز هم ناشی از پروتکل‌های پیاده‌سازی حالت Bridged Networking در VirtualBox باشد. یک توجیه آنست که تمامی ترافیک‌های عبوری در VirtualBox از دید سیستم میزبان، همگی به چشم ترافیک عبوری از یک واسط و یک پورت دیده می‌شوند که باعث می‌شود ترافیک پورت‌های 2 و 3 نیز روی پورت 1 فوروارده شود. اجرای مجدد این سناریو در محیط mininet در بخش بعد اثباتی بر این مدعاست.

در میزبان، می‌توان با دستور `ovs-ofctl dump-flows mybridge` جدول جریان مربوط به سوئیچ `mybridge` را مشاهده نمود:

```
root $ ovs-ofctl -O OpenFlow13 dump-flows mybridge
cookie=0x0, duration=95.284s, table=0, n_packets=935, n_bytes=400427,
priority=1,in_port=enp0s25,dl_dst=f0:de:f1:ef:e1:4c actions=LOCAL
cookie=0x0, duration=95.280s, table=0, n_packets=980, n_bytes=144737,
priority=1,in_port=LOCAL,dl_dst=6c:fd:b9:8c:db:d5 actions=output:enp0
s25
cookie=0x0, duration=90.812s, table=0, n_packets=44, n_bytes=4274, pr
iority=1,in_port=vport2,dl_dst=08:00:27:9a:d5:77 actions=output:vport1
cookie=0x0, duration=89.813s, table=0, n_packets=44, n_bytes=4274, pr
iority=1,in_port=vport1,dl_dst=08:00:27:a4:54:fc actions=output:vport2
cookie=0x0, duration=37.685s, table=0, n_packets=97, n_bytes=29985, p
riority=0 actions=CONTROLLER:65535
root $
```

شکل 36 – جدول جریان سوئیچ `mybridge`

5.1.2 اجرا در محیط `mininet`

این بار بر خلاف دفعه قبل، ابتدا اتصال سوئیچ `mybridge` را از کنترلر `Ryu` قطع می‌نماییم. چرا که جهت جلوگیری از سردرگمی می‌خواهیم تنها سوئیچ متصل به کنترلر، سوئیچ `s1` باشد. حال با دستور زیر توپولوژی مربوطه را در محیط `mininet` ایجاد می‌کنیم:

```
mn -topo single,2 -mac -switch ovsk -controller remote -x
```

تنها فرق آن با دستور ایجاد توپولوژی در فصل قبل، این است که بجای `single,3`، از `single,2` استفاده کرده‌ایم که فقط 2 میزبان ایجاد می‌کند. ایجاد میزبان سوم بلااستفاده در مانیتورینگ کاری اضافی است.

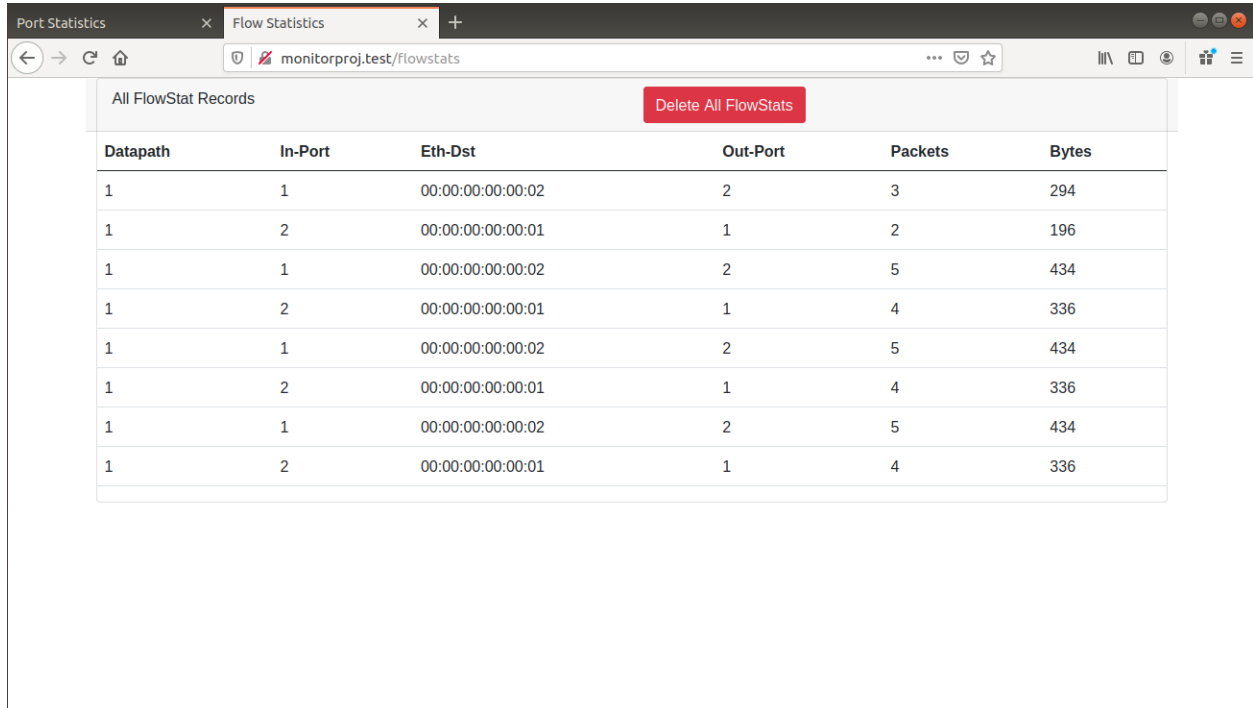
پیش از اجرای کنترلر، از صفحات وب مربوط به اطلاعات آماری جریان‌ها و همچنین اطلاعات آماری پورت‌ها، به ترتیب روی `Delete All FlowStats` و `Delete All PortStats` کلیک می‌کنیم که رکوردهای قبلی این دو جدول ناشی از اجرای `monitor` بین ماشین‌های مجازی پاک شوند. اکنون، دو جدول خالی داریم که با اجرای کنترلر در محیط `mininet` جداول آن از نو شروع به پر شدن می‌کنند.

مجدداً به ازای هریک از میزبان‌ها، سوئیچ و کنترلر یک ترمینال جداگانه `xterm` ایجاد می‌شود. حال در ترمینال همان دستور اجرای کنترلر را وارد و اجرا می‌نماییم.

از میزبان `h1` به میزبان `h2` `ping` می‌فرستیم. برای این کار، می‌توان مستقیماً از ترمینال مربوط به `h1`، به `ping` کردن `h2` مبادرت نمود. نیز می‌توان این کار را بوسیله دستور `h2 ping -c1 h1` در ترمینال `xterm` اولیه که در آن دستور `mn` اجرا شد انجام داد. توجه داشته باشید میزبان `h1` به پورت شماره 1 از سوئیچ `s1` متصل است و

دارای آدرس MAC 00:00:00:00:00:01 می‌باشد. میزبان h2 به پورت شماره 2 از سوئیچ s1 متصل بوده دارای آدرس MAC 00:00:00:00:00:02 می‌باشد.

صفحه وب مربوط به اطلاعات آماری جریان‌ها به شکل زیر خواهد بود:



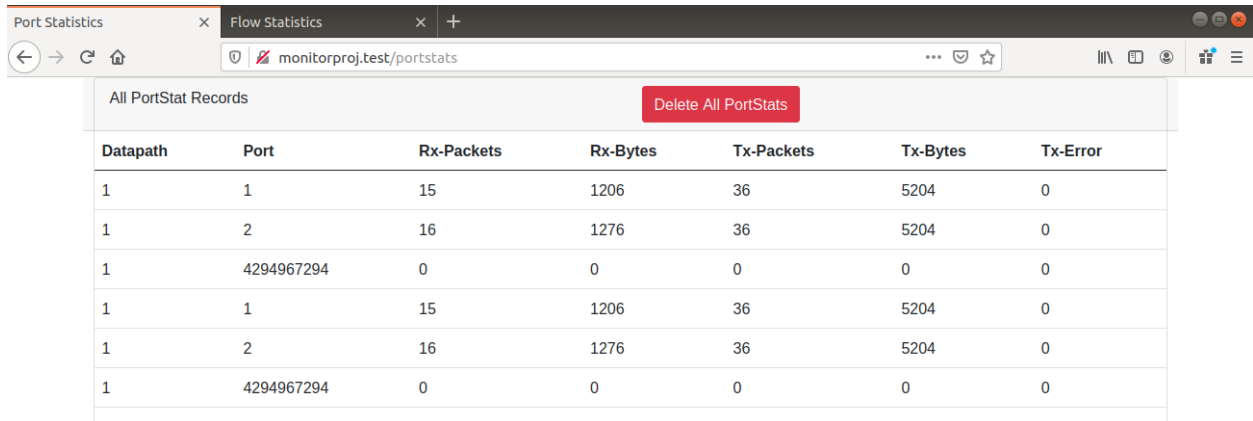
The screenshot shows a web browser window with the address bar displaying 'monitorproj.test/flowstats'. The page title is 'Flow Statistics'. Below the title bar, there is a table titled 'All FlowStat Records' with a red button labeled 'Delete All FlowStats'. The table has six columns: 'Datapath', 'In-Port', 'Eth-Dst', 'Out-Port', 'Packets', and 'Bytes'. The table contains eight rows of data.

Datapath	In-Port	Eth-Dst	Out-Port	Packets	Bytes
1	1	00:00:00:00:00:02	2	3	294
1	2	00:00:00:00:00:01	1	2	196
1	1	00:00:00:00:00:02	2	5	434
1	2	00:00:00:00:00:01	1	4	336
1	1	00:00:00:00:00:02	2	5	434
1	2	00:00:00:00:00:01	1	4	336
1	1	00:00:00:00:00:02	2	5	434
1	2	00:00:00:00:00:01	1	4	336

شکل 37 – صفحه وب مربوط به اطلاعات آماری جریان‌ها در محیط mininet

همانطور که گفته بودیم، برخلاف محیط VirtualBox در محیط mininet دیگر از ترافیک‌های اضافه بر ترافیک مربوط به ping ناشی از VirtualBox اثری نیست و جریان‌های جدول جریان سوئیچ s1، فقط همان دو جریانی هستند که در نتیجه مکانیزم ping به وجود می‌آیند. برای اطلاعات بیشتر می‌توانید به بررسی کد سوئیچ یادگیرنده در فصل قبل مراجعه کنید.

نهایتاً صفحه وب مربوط به اطلاعات آماری پورت‌ها به شکل زیر خواهد بود:



Datapath	Port	Rx-Packets	Rx-Bytes	Tx-Packets	Tx-Bytes	Tx-Error
1	1	15	1206	36	5204	0
1	2	16	1276	36	5204	0
1	4294967294	0	0	0	0	0
1	1	15	1206	36	5204	0
1	2	16	1276	36	5204	0
1	4294967294	0	0	0	0	0

شکل 38 – صفحه وب مربوط به اطلاعات آماری پورت‌ها در محیط mininet

همانطور که در این شکل می‌بینید، برخلاف محیط VirtualBox در محیط mininet هر یک از پورت‌های سوئیچ که به یک میزبان متصل است، ترافیک عبوری دارد و اینطور نیست که عدد این ترافیک‌ها 0 باشد و به پورت دیگری فوروارد شده باشند.

5.2 جمع‌بندی

در این فصل بنا بود با استفاده از سوئیچ یادگیرنده که فصل پیش کد آماده آن در کنترلر Ryu مورد بررسی و اجرا قرار گرفت، اقدام به پیاده‌سازی monitor کنیم. در ابتدا محیط مجازی‌سازی VirtualBox را بررسی کردیم و انواع حالات شبکه کردن در آن را شرح دادیم. سپس با انتخاب حالت شبکه مناسب و ایجاد اتصالات لازم بین سوئیچ و ماشین‌های مجازی و سیستم میزبان، کد پایتون مربوط به monitor را بررسی کردیم. پس از اجرای کنترلر در محیط ماشین‌های مجازی، خروجی دو جدول مربوط به اطلاعات آماری جریان‌ها و همچنین پورت‌ها، موارد غیر منتظره‌ای را نمایش داد. ادعایمان این بود که آن موارد غیر منتظره از اجرای monitor در محیط VirtualBox نشأت می‌گرفتند. برای اثبات این ادعا، یک‌بار دیگر و این بار در محیط mininet برنامه monitor را به‌همراه راهکار vTAP مربوطه اجرا کردیم و دیدیم که از بروز اتفاقات غیر منتظره جلوگیری شده است. بنابراین با اجرای monitor در دو محیط مجازی‌سازی، کارکرد آنرا امتحان کرده و نتایج جداول مربوط به اطلاعات آماری جریان‌ها و همچنین پورت‌ها را به ازای هر بار امتحان، در قالب تصاویر نشان دادیم.

6 فصل ششم: کارهای آینده

- 1- مهم‌ترین مورد از موارد استفاده^{۹۶} vTAP تقویت دفاع در برابر حملات امنیتی است. استفاده از سناریوهای ایجاد حملات ساختگی و بررسی تشخیص آن‌ها بر عهده سیستم مانیتورینگ می‌باشد. از فریم‌ورک‌های آماده جهت انجام این کار می‌توان به pybull و Suricata اشاره کرد که با استفاده از آن‌ها می‌توان امکان دفاع در برابر حملات امنیتی را به monitor موجود اضافه نمود.
- 2- در صورت استفاده از^{۹۷} OVS، DPDK در حالت کاربر کار می‌کند. این امر نسبت به حالت عادی که OVS در حالت کرنل است، باعث می‌شود لایه‌های سنگین پشته شبکه در کرنل دور زده شود و ارتباط مستقیم با سخت‌افزار مربوط به شبکه برقرار گردد. همچنین بعلت استفاده از Hugepage ها در DPDK که اندازه‌شان از 2MB تا 1GB متغیر است، تعداد memory page کمتری نسبت به حالت استاندارد (با اندازه معمولاً 4KB) مورد نیاز خواهد بود. این امر، موجب کاهش TLB^{۹۸} miss و افزایش سرعت عملکرد OVS خواهد شد.
- 3- محیط mininet برای ایجاد توپولوژی، از تجریدی به نام namespace استفاده می‌کند. Namespace قابلیت‌هایی است که باعث می‌شود هر کانتینر^{۹۹} مثل یک ماشین جداگانه عمل نماید. عبارتی منابع سراسری سیستم بصورت مجرد، بگونه‌ای بنظر میرسد که انگار پروسه‌های موجود در هر namespace هر یک نمونه‌ای ایزوله از منابع سراسری در اختیار دارد که از منابع سایر namespace ها مستقل و مجزاست. تجرید namespace در کنار مفاهیم دیگری همچون cgroup و capability پایه و اساس ساخت کانتینرها را تشکیل می‌دهد. در این پروژه، در اجرایی که در محیط mininet انجام گرفته صرفاً از namespace ها استفاده شده که حالت خاص کانتینرها محسوب می‌شود. استفاده از راهکار vTAP در محیط‌های کانتینریزاسیون^{۱۰۰} از قبیل Docker و Kubernetes و یا محیط‌های ابری می‌تواند گامی فراتر برای توسعه vTAP استفاده شده در محیط مجازی تلقی گردد.

^{۹۶} Use-Cases^{۹۷} Data Plane Development Kit^{۹۸} Translation Lookaside Buffer^{۹۹} Container^{۱۰۰} Containerization

7 منابع و مراجع

- [1] "https://en.wikipedia.org/wiki/Software-defined_networking," [Online].
- [2] G. Bouchard, "What Are Network TAPs? And Why Do We Need Them?," Profitap, 10 February 2020. [Online]. Available: <https://insights.profitap.com/what-are-network-taps>.
- [3] J. Wade, "Greenfield Vs. Brownfield Software Development. What's The Difference?," 27 September 2018. [Online]. Available: <https://synoptek.com/insights/it-blogs/greenfield-vs-brownfield-software-development/>.
- [4] J. Casey, "Introduction to SDN & OpenFlow," INE, [Online]. Available: <https://my.ine.com/Networking/courses/7a55a73d/introduction-to-sdn-and-openflow>.
- [5] "What is network service chaining?," Spectrum ENTERPRISE, [Online]. Available: <https://enterprise.spectrum.com/support/faq/network/what-is-network-service-chaining.html>.
- [6] S. Raynovich, "What Is Network Service Chaining," 11 February 2016. [Online]. Available: <https://www.sdxcentral.com/networking/virtualization/definitions/what-is-network-service-chaining/>.
- [7] R. Dua, V. Kohli and S. K. Konduri, "Overlay networks and underlay networks," in *Learning Docker Networking*, Packt Publishing.
- [8] D. Mahler, "Introduction to Cloud Overlay Networks - VXLAN," 3 Jun 2014. [Online]. Available: https://www.youtube.com/watch?v=Jqm_4TMmQz8.
- [9] J. Casey and A. Sprintson, "Applications of Software-Defined Networking (SDN) in Power System Communication Infrastructure: Benefits and Challenges," in *PSERC*, 2015.
- [10] "The Most Widely Deployed Open Source Cloud Software in the World," [Online]. Available: <https://www.openstack.org/>.
- [11] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg and S. Azodolmolky, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, 2015.
- [12] "Mask (computing)," [Online]. Available: [https://en.wikipedia.org/wiki/Mask_\(computing\)](https://en.wikipedia.org/wiki/Mask_(computing)).
- [13] "Classification," Flowgrammable, [Online]. Available: <http://flowgrammable.org/sdn/openflow/classifiers/>.
- [14] P. Wette, "Testing SDN behavior with Mininet," 2014. [Online]. Available:

-
- <https://www.linux-magazine.com/Issues/2014/162/Mininet>.
- [15] "VirtualBox," Oracle, [Online]. Available: <https://www.virtualbox.org/>.
- [16] "Chapter 6. Virtual Networking," Oracle, [Online]. Available: <https://www.virtualbox.org/manual/ch06.html#networkingmodes>.
- [17] S. Simpson, "What is VirtualBox?," Lynda, 17 November 2017. [Online]. Available: <https://www.lynda.com/Linux-tutorials/What-VirtualBox/597026/678851-4.html>.
- [18] D. Mahler, "Introduction to Open vSwitch (OVS)," Youtube, 15 December 2013. [Online]. Available: <https://www.youtube.com/watch?v=rYW7kQRyUvA&t=6s>.
- [19] "Bridged Mode not working," Oracle, 10 Jun 2020. [Online]. Available: <https://forums.virtualbox.org/viewtopic.php?f=8&t=98508>.
- [20] "ioctl (TUNSETIFF) : device or resource busy," [Online]. Available: <https://stackoverflow.com/questions/37026255/ioctl-tunsetiff-device-or-resource-busy?rq=1>.
- [21] "Ubuntu, remove network TAP interface," [Online]. Available: <https://stackoverflow.com/questions/17529345/ubuntu-remove-network-tap-interface>.
- [22] "Production Quality, Multilayer Open Virtual Switch," [Online]. Available: <https://www.openvswitch.org/>.
- [23] R. Nathuji, "Introduction to Open vSwitch," INE, [Online]. Available: <https://my.ine.com/Networking/courses/f7eabce2/introduction-to-open-vswitch>.
- [24] "Mininet," 2018. [Online]. Available: <http://mininet.org/>.
- [25] A. Pashamokhtari, "Optimization and Implementation of quality of service features for VPLS in SDN environment," B.S. Thesis, Department of Computer Engineering and Information Technology, Amirkabir Univ., Tehran, 2018.
- [26] "veth(4) — Linux manual page," 1 November 2020. [Online]. Available: <https://man7.org/linux/man-pages/man4/veth.4.html>.
- [27] "Switching Hub," [Online]. Available: https://osrg.github.io/ryu-book/en/html/switching_hub.html.
- [28] "PacketIn," Flowgrammable, [Online]. Available: <http://flowgrammable.org/sdn/openflow/message-layer/packetin/>.
- [29] "FeatureReq - FeatureRes," Flowgrammable, [Online]. Available: <http://flowgrammable.org/sdn/openflow/message-layer/feature/>.
- [30] "Ryu application API," [Online]. Available: https://ryu.readthedocs.io/en/latest/ryu_app_api.html.
- [31] "State Machine," Flowgrammable, [Online]. Available: <http://flowgrammable.org/sdn/openflow/state-machine/>.
- [32] "Traffic Monitor," [Online]. Available: https://osrg.github.io/ryu-book/en/html/traffic_monitor.html.
