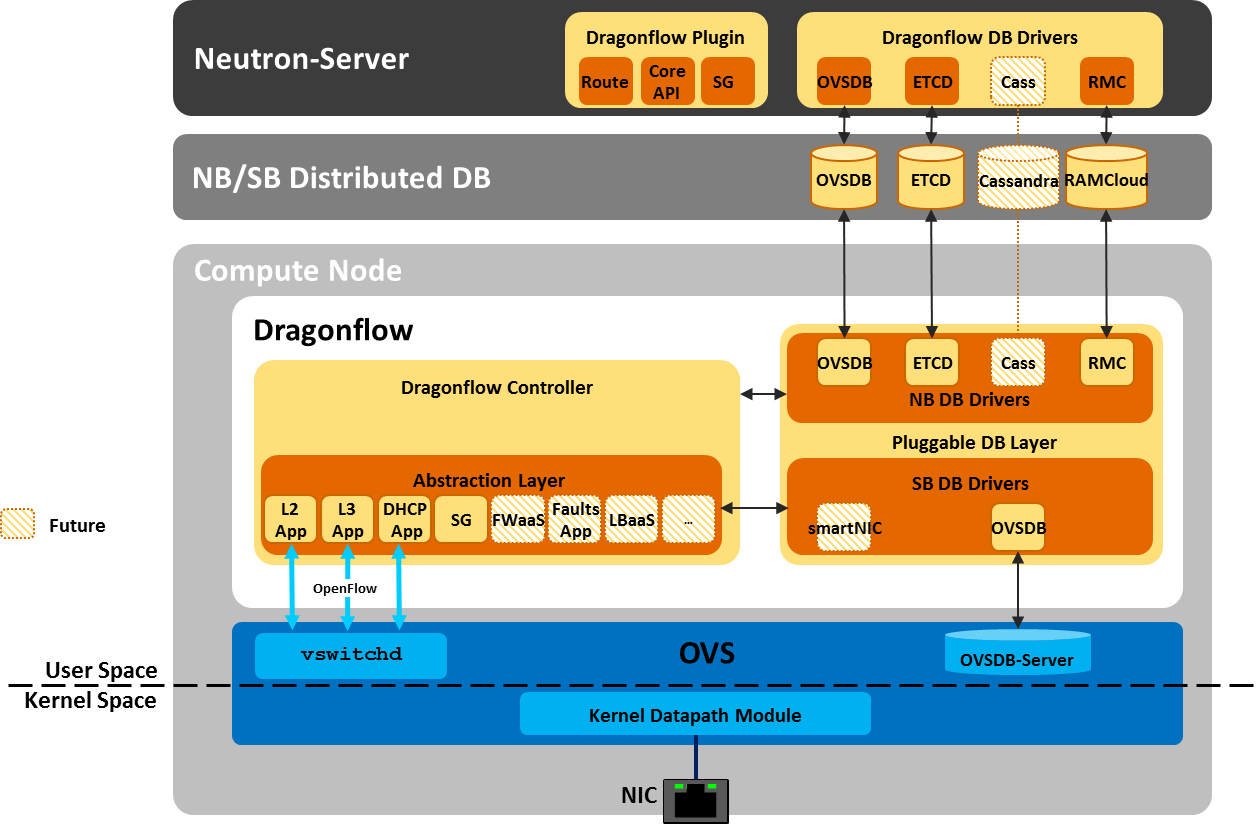
### DRAGONFLOW

## 1. Lý thuyết về Dragonflow

### *1.1. Distributed Dragonflow*

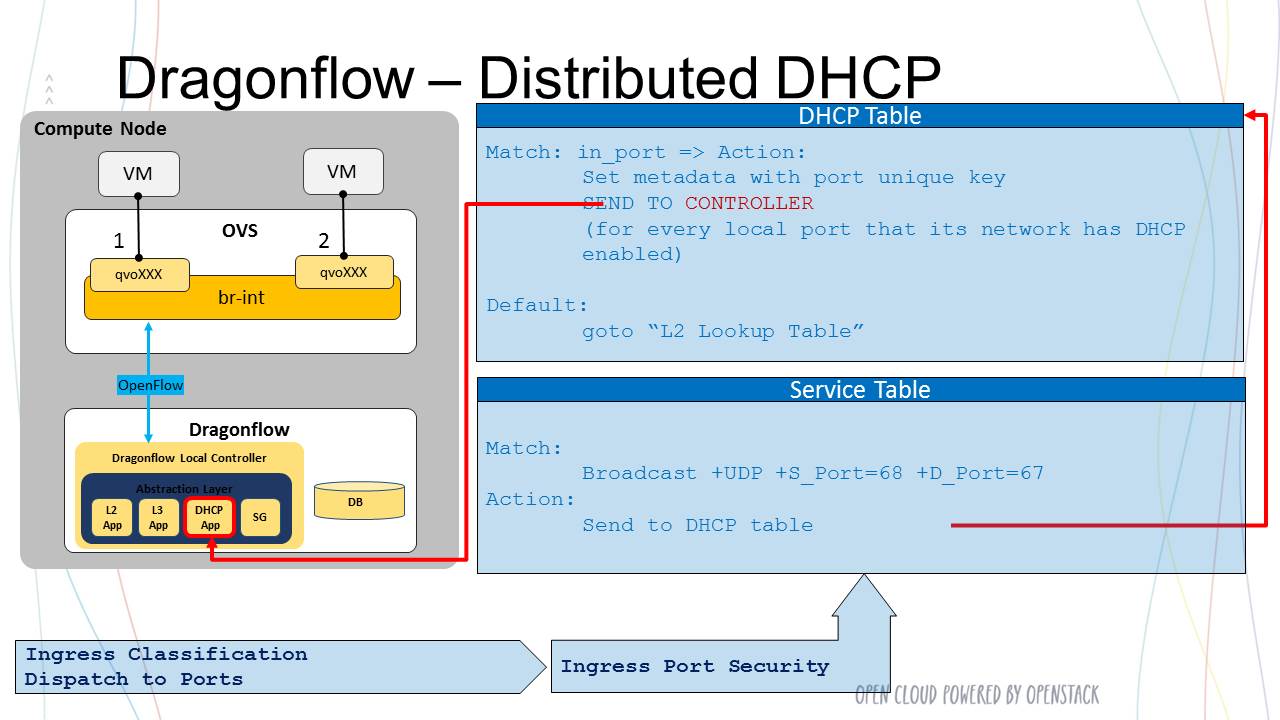
* Dragonflow là distributed SDN controller cho OpenStack Neutron hỗ trợ switching, routing, DHCP phân tán. Dragonflow được thiết kế để hỗ trợ môi trường triển khai quy mô lớn, tập trung vào latency và hiệu năng.
* Một số điểm chính trong thiết kế của Dragonflow:
* Cung cấp Pluggable database để đáp ứng với nhiều môi trường với những yêu cầu khác nhau về quy mô, hiệu năng, latency.
* Đồng bộ hóa policy, topology tới các Compute Node.
* Local Dragonflow controller sử dụng reactive model (sẽ giải thích trong các mục sau)
* Kiến trúc tổng quan:



* Dragonflow bao gồm một local controller chạy trên mỗi Compute Nodes khi triển khai. Các controller này sẽ đồng bộ network topology và policy sử dụng pluggable database. Sau đó Controller sẽ map các policy thành các OpenFlow flows sử dụng local Dragonflow applications (L2 app, L3 app, DHCP app, Security Group) để tương tác với Open vSwitch bridge cục bộ.
* Database được populate bởi Dragonflow Neutron plugin, từ đó chuyển đổi neutron API sang data model của Dragonflow.
* Một số tính năng mà Dragonflow đã hỗ trợ:
* L2 core API, IPv4, IPv6: hỗ trợ GRE/VxLAN/Geneve tunneling.
* Distributed virtual router L3
* Distributed DHCP
* Pluggable distributed database: etcd, ovsdb, redis, rethinkdb, ramcloud, cassandra.

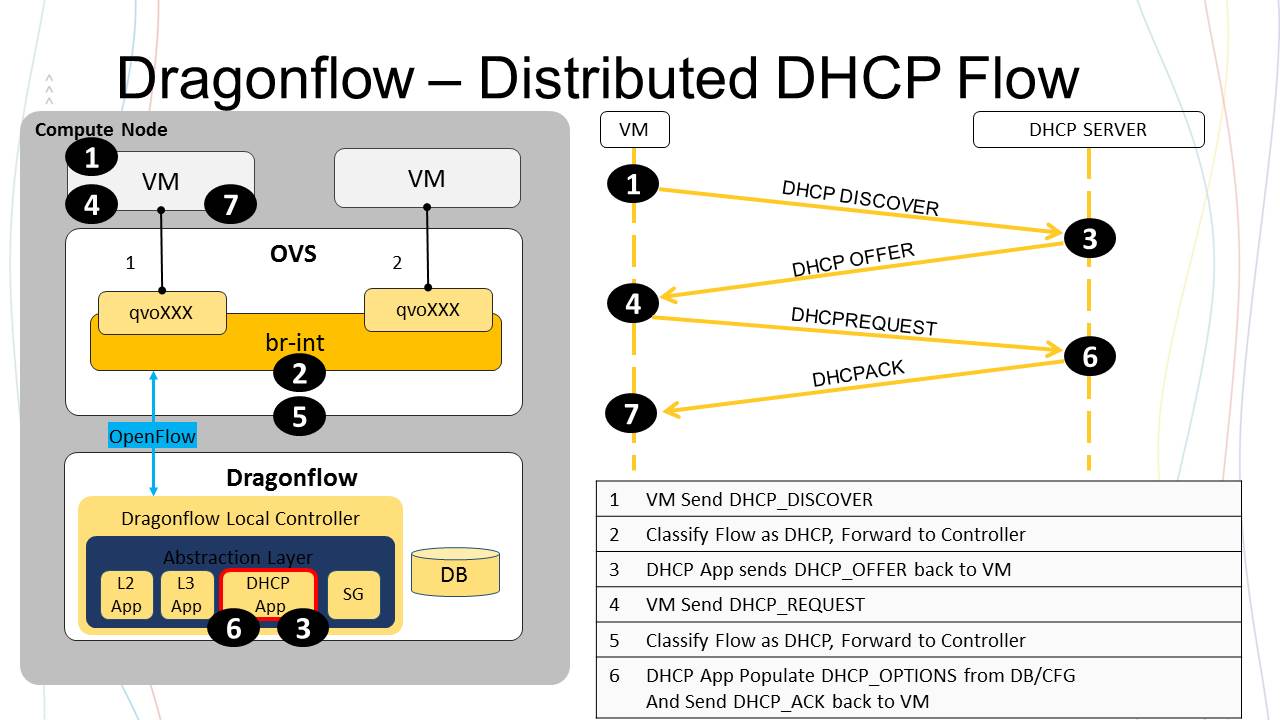
### *1.2. Distributed DHCP*

* Một số hạn chế trong triển khai DHCP hiện tại của neutron:
* DHCP server được thiết kế sử dụng dnsmasq chạy trong một namespace trên network-node đối với mỗi tenant subnet mà kích hoạt DHCP (lúc khởi tạo subnet). Trong trường hợp để đạt tính sẵn sàng cao, nhiều dnsmasq servers sẽ được triển khai trên nhiều network nodes. Tuy nhiên hiện tại dhcp agent có thể cài đặt phân tán trên các Compute Node thay vì tập trung trên các network nodes.
* Xét trong trường hợp nào thì triển khai DHCP sử dụng dhcp-agent cũng có một số hạn chế như sau:
* Hạn chế trong việc quản lý và mở rộng, nghĩa là cần phải cấu hình và quản lý nhiều dnsmasq instances
* Giải pháp triển khai DHCP tập trung sẽ tạo gánh nặng cho các network node.
* Distributed DHCP trong Dragonflow:
* Dragonflow phân bố policy/cấu hình về DHCP sử dụng pluggable database. Mỗi local controller (trên các Compute Node) sẽ chuyển đổi các thao tác DHCP sang Openflow rules để xử lý. Điều này cũng tương tự với các chức năng về L3, L2, Security Group, được quản lý bằng các application tương ứng trên local Controller, và việc triển khai các thành phần như Distributed Virtual Router (L3) hay các Security Group Rule được triển khai thành các flow trên integration bridge **br-int.** Các rules này được áp dụng chỉ cho các ports cục bộ mà được attach với virtual network kích hoạt DHCP.
* Controller thiết lập flow metadata trong các DHCP rule bằng key xác định duy nhất với các port cục bộ cho phép tìm kiếm nhanh thông tin của port.
* Local DHCP app xử lý các bản tin DHCP và trả lời như một DHCP server. DHCP traffic được xử lý trực tiếp tại compute node và không bao giờ đi ra ngoài. DHCP service pipeline được đặc tả thành Dragonflow's OpenFlow pipeline như sau:



* Classification flow được thêm vào Service Table của br-int để match DHCP traffic và chuyển tiếp nó tới DHCP Table
* Trên DHCP Table, flow matching được hiện dựa vào in\_port (tap interface trên br-int) của mỗi VM kết nối với một subnet kích hoạt DHCP server
* Lưu lượng từ các VM trên các subnets không kích hoạt DHCP server sẽ chuyển tiếp tới L2 Lookup table.
* Đối với các packets match thành công tren DHCP table, port\_id sử dụng cho giá trị metadata để tìm kiếm nhanh thông tin của port trên local controller database, sau đó chuyển tiếp gói tin lên local controller.
* Tại controller, bản tin PACKET\_IN (gửi từ br-int lên) với DHCP message được chuyển tiếp sang DHCP SDN Application để xử lý.
* Với DHCP Application, bản tin DHCP\_DISCOVER được xử lý và trả lời với DHCP\_ACK mang theo DHCP\_OPTIONs lấy từ local database.

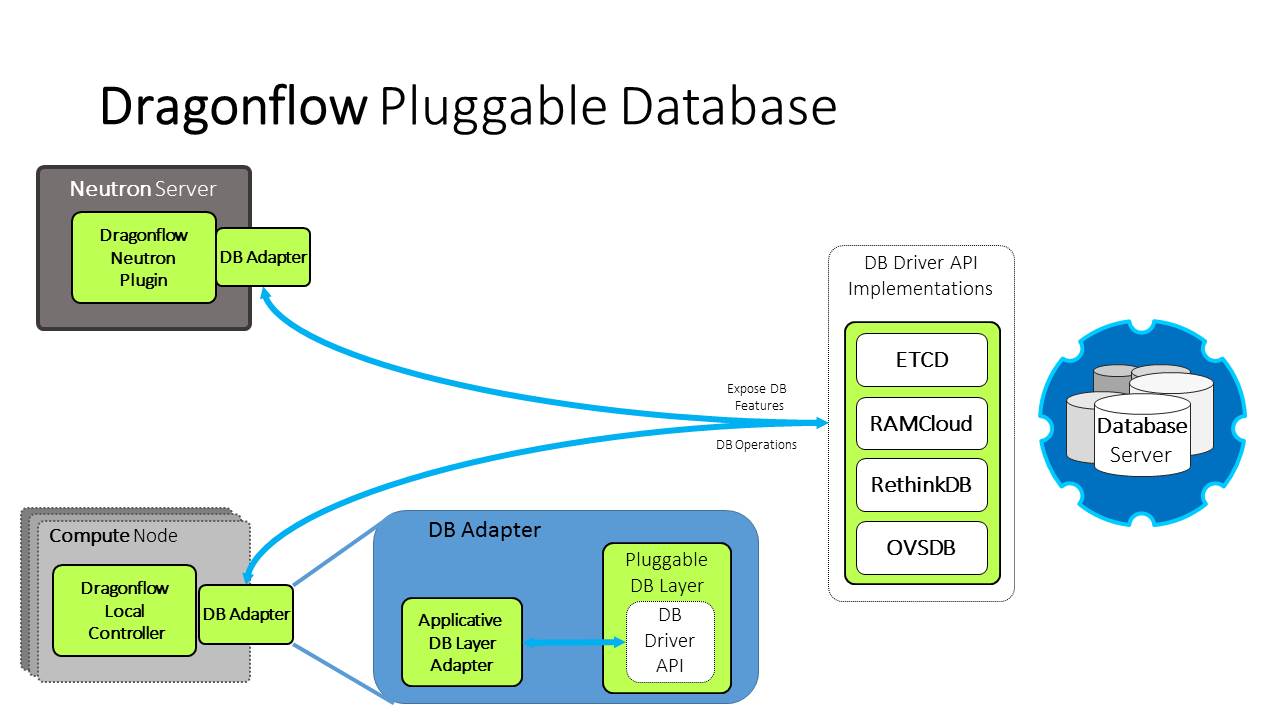
Distributed DHCP flow được thể hiện như hình bên dưới:



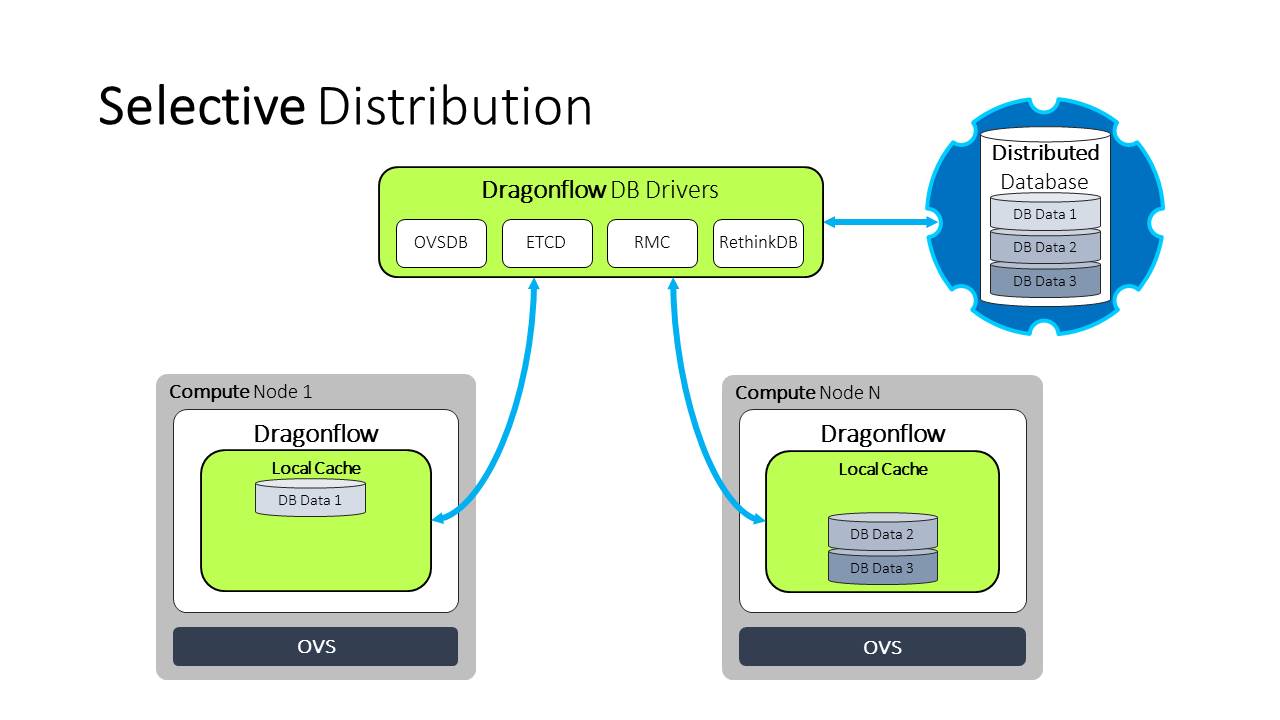
* Bước 1: máy ảo gửi bản tin DHCP\_DISCOVER
* Bước 2: bridge br-int phân loại lưu lượng, thiết lập flow cho DHCP, chuyển tiếp lên controller xử lý
* Bước 3: DHCP app trên Dragonflow gửi bản tin DHCP\_OFFER trở lại máy ảo để trả lời
* Bước 4: máy ảo gửi bản tin DHCP\_REQUEST
* Bước 5: bridge br-int phân loại lưu lượng thành flow DHCP rồi chuyển tiếp lên controler
* Bước 6: DHCP app trên controller lấy thông tin từ database để có được DHCP options đưa vào bản tin DHCP\_ACK, trả lại cho máy ảo.

### *1.3. Pluggable database*

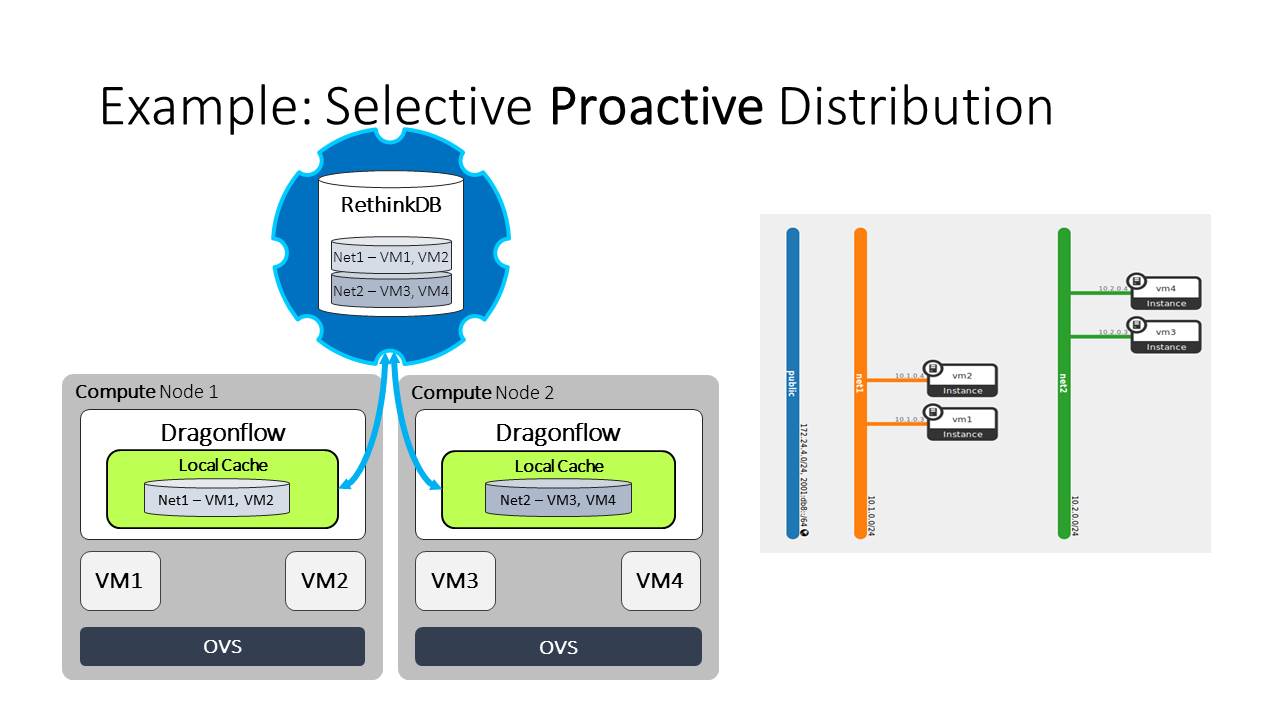
* Database framework là mô hình để đồng bộ về network policy và topology giữa Central Management Store (trên các OpenStack Controller Node) với các local SDN controller (trên các OpenStack Compute Node) và do đó kiểm soát được hiệu năng, latency và quy mô của môi trường triển khai Dragonflow.
* Pluggable database layer cho phép người vận hành linh hoạt trong việc lựa chọn và thay đổi giải pháp database phù hợp với môi trường của riêng mình.
* Dưới đây là kiến trúc pluggable database:



* Mode của database:
* **Full proactive**: ở mode này, tất cả dữ liệu của database (policy và topology) được đồng bộ trên toàn bộ các local Dragonflow controllers (trên mỗi compute node). Dragonflow lưu giữ toàn bộ dữ liệu đã được đồng bộ từ database vào local in-memory cache cho phép tìm kiếm nhanh.
* **Selective Proactive**: ở mode này, dragonflow chỉ cần đồng bộ dữ liệu liên quan cho local-controller trên mỗi compute node, phụ thuộc vào các port cục bộ trên compute node đó.



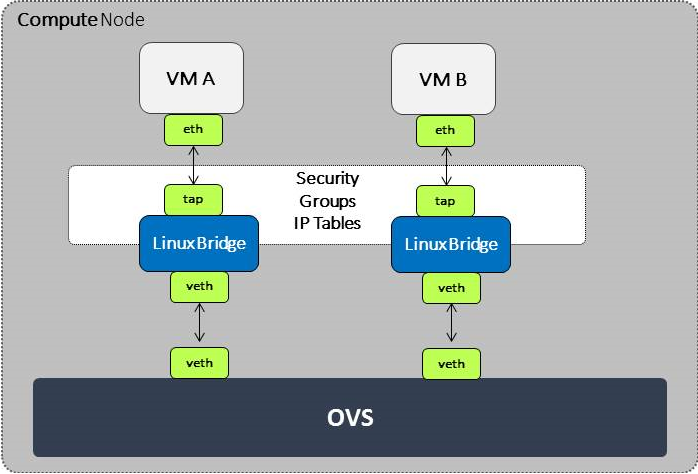
* Ví dụ như sau:



Có thể thấy trong sơ đồ ví dụ ở trên rằng, mỗi compute node chứa các VMs từ hai network cô lập với nhau như trên đồ hình mạng, nghĩa là các máy ảo trên một network không thể liên lạc với các máy ảo trên các network khác. Do đó mỗi compute node chỉ cần lấy thông tin về topology và policy liên quan tới network và các VMs đặt cục bộ trên compute node đó.

### *1.4. Security group*

* Neutron triển khai security group bằng việc thêm một linux bridge trên đường đi giữa mỗi port của instance và integration bridge br-int. Linux bridge này được cấu hình với các IP table rules để làm security group rule cho port này (Iptable rule không thể cấu hình cho Open vSwitch port).



* Trong Dragonflow, VMs được kết nối với integration bridge (ovs bridge) br-int bằng việc tận dụng Open vSwitch connection tracking (<http://openvswitch.org/support/ovscon2014/17/1030-conntrack_nat.pdf> -<https://www.youtube.com/watch?v=Wx9ZqCYW5sk>), nhờ đó các security group rules được triển khai sử dụng OVS flows và các actions tương ứng. Bên cạnh đó, việc triển khai như vậy tránh việc phải tái biên dịch các rules tới iptables chains pipeline mỗi khi thay đổi cấu hình của VM port (thêm/cập nhật/xóa). Việc thay đổi security group rules sẽ chỉ cập nhật một số lượng nhỏ các flows.
* Security group pipeline:
* Egress:



* Ingress:



### *1.5. Publish-subscribe*

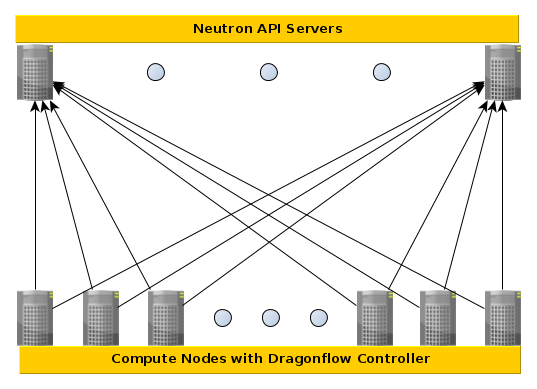
Dragonflow triển khai mô hình publish-subscribe dựa trên pluggable DB driver để lắng nghe các sự kiện thay đổi cấu hình. Luồng hoạt động của mô hình như sau:

* Local controller đọc database va đồng bộ toàn bộ cấu hình cần thiết.
* Sau đó, nó đợi thông báo thay đổi bằng việc gọi một DB driver API đặc biệt với phương thức callback.

Sau đó DB driver chịu trách nhiệm cập nhật controller khi có bất kì thay đổi nào về cấu hình. Nếu DB driver không hỗ trợ mô hình publish-subscribe, local controller tiếp tục thăm dò DB và tìm kiếm sự thay đổi bằng việc so sánh DB với local cache. DB driver hỗ trợ publish-subscribe hiện tại có redis.

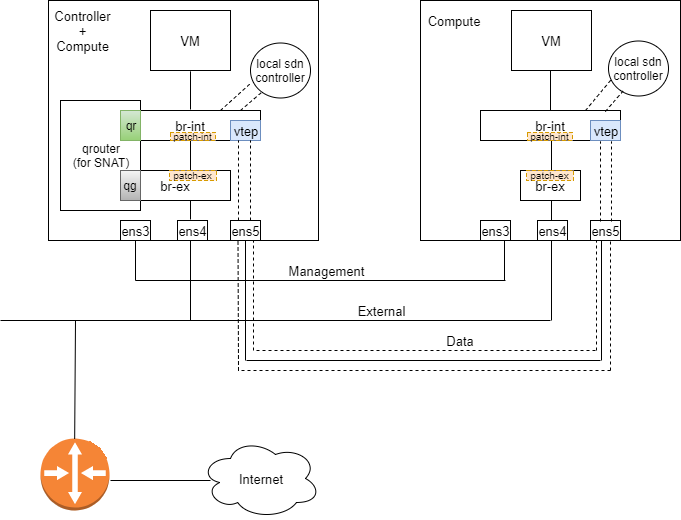


Ở các bản phát hành gần đây, Dragonflow có sự thay đổi về mô hình publish-subscribe, bằng việc phát triển một mô hình pub/sub tích hợp với Dragonflow theo tiêu chí pluggable. Trong mô hình này sẽ có nhiều neutron API server đóng vai trò là publisher. Mỗi local controller trên các compute node sẽ đăng ký như là subscriber tới mỗi neutron API server.



## 2. Cài đặt Dragonflow

### *2.1. Mô hình cài đặt*



* Mô hình cài đặt ở đây gồm hai node:
* **Controller Node**: đóng vai trò của cả Controller và Compute trong OpenStack với hầu hết các dịch vụ cơ bản (Glance, Nova, Neutron, Cinder, Horizon). Trong đó nova ngoài các dịch vụ thông thường cài đặt cho controller có cài thêm cả dịch vụ nova-compute đóng vai trò như một node Compute. Về neutron, thay vì cài đặt l3 agent, open vswitch agent, dhcp agent và metadata agent như cách cài đặt truyền thống, DragonFlow được cài đặt và kích hoạt các dịch vụ để thay thế bao gồm: df-controller, df-metadata, df-l3-agent, df-publisher. Ngoài ra còn redis cũng được cài đặt đóng vai trò là database phân tán để lưu trữ và truyền cấu hình về topology giữa các compute node với các controller node. Redis trên Controller cũng cung cấp cả publisher service trong mô hình publisher/subscriber khi sử dụng Dragonflow.
* **Compute Node**: cài đặt df-controller, df-metadata, nova-compute. Compute node cũng cài đặt Redis với vai trò làm database lưu giữ các thông tin cần thiết liên quan tới network ảnh hưởng tới các instance đặt tại compute node đó, ngoài ra cũng đóng vai trò là subscriber trong mô hình publichser/subscriber khi sử dụng Dragonflow.
* Networking: mỗi node gồm 3 card mạng kết nối vào các dải mạng đóng vai trò riêng biệt.
* Management Network – 10.10.0.0/24
* External Network – 172.16.1.0/24
* Data Network – 10.10.1.0/24: mang lưu lượng East-West giữa các instance trên các Compute Node và lưu lượng SNAT khi máy ảo trên Compute Node sử dụng fixed IP kết nối ra internet (đi qua qrouter trên Controller node).
* DevStack setting:
* local.conf (controller): 
* local.conf (compute1):



### *2.2. Luồng đi dữ liệu trong các kịch bản*

* Các network, subnet, router và instance sử dụng trong các phân tích bên dưới như sau:
* Các network, subnet:

|  |  |  |  |
| --- | --- | --- | --- |
| Network | Subnet | Address | Gateway |
| public | public-subnet | 172.16.1.0/24 | 172.16.1.1 |
| private | private-subnet | 10.0.0.0/26 | 10.0.0.1 (trên router1) |
| private-1 | private1-subnet | 172.17.1.0/24 | 172.17.1.1 (trên router1) |

* Router: router1, public gateway trên mạng public.
* Các instance:

|  |  |  |  |
| --- | --- | --- | --- |
| Instance | Fixed IP | Floating IP | Compute host |
| df-test-instance-1 | 10.0.0.11 | 172.16.1.55 | controller |
| df-test-instance-2 | 10.0.0.13 | 172.16.1.57 | compute1 |
| df-test-instance-3 | 10.0.0.15 | 172.16.1.58 | compute1 |
| df-test-instance-4 | 172.17.1.8 | 172.16.1.60 | compute1 |

* **Kịch bản 1**: Lưu lượng North-South sử dụng Floating IP. Trường hợp này test với **df-test-instance-4**:
* Kiểm tra interface của instance:



* Kiểm tra tap interface nối với instance trên bridge br-int:



Như vậy instance 4 gắn vào OpenFlow port 9 trên bridge br-int.

* Ping từ instance ra google.com, rồi bắt gói tin trên tap interface này sử dụng tcpdump:



* Trước hết xem qua OpenFlow entry trên br-int mà các actions liên quan tới local SDN Controller:



Có thể quan sát thấy flow entry thứ ba áp dụng cho các gói tin mà br-int nhận được trên OpenFlow port 9. Ta thấy rằng có hai packets match flow này, chính là các bản tin ARP và ICMP khi tới OpenFlow port 9 ở thời điểm ban đầu mà chưa được chỉ ra phải xử lý như thế nào tại br-int, do đó br-int phải liên lạc với local SDN controller. Sau bước này, SDN controller sẽ cài đặt các flow lên br-int để xử lý các gói tin này theo pipeline.

* Kiểm tra flow entry áp dụng cho OF port 9 trên table 0 (flow table đầu tiên mọi gói tin đều phải đi qua):



Chú ý giá trị metadata được thiết lập bằng 0x367 để trao đổi các packet tới port 9 qua các flow table. Hành động cuối cùng là gửi tới flow table 5.

* Kiểm tra flow table 5:



Bản tin cần quan tâm ở đây là ICMP, như vậy sẽ match với flow entry đầu tiên (địa chỉ MAC nguồn và IP nguồn match với gói tin đi từ instance 4, và có độ ưu tiên cao hơn flow entry cuối cùng). Hành động là gửi sang flow table 10.

* Kiểm tra flow table 10 (**egress connection tracking table**):



Quan sát thấy rằng **actions=ct(table=15,zone=OXM\_OF\_METADATA[0..15]),** điều này được hiểu như sau: local port của instance 4 được gắn kết với security group (được chỉ ra khi khởi tạo instance), do đó Dragonflow controller sẽ thiết lập flow cho các gói tin đi từ instance ra ngoài vào một flow table, table này là **egress security group table**, là table triển khai security group rules, ở đây là table 15.

* Kiểm tra flow table 15:



Dưới góc độ của instance port, khi port của instance được áp dụng security group, Dragonflow controller sẽ thiết lập các OpenFlow flows trên egress security group table 15 với mục đích chuyển đổi các security group rule sang OpenFlow flows. Flow này mang theo trường match với instance port ID (giá trị với lưu lượng egress sẽ bằng số hiệu OpenFlow port trên OVS bridge, ở đây là 9, còn với lưu lượng ingress sẽ bằng giá trị tương ứng ghi trong thanh ghi 7 **reg7** ).

Bên cạnh đó, Dragonflow controller cũng thiết lập flow match với trường chứa conjunction id khớp với security group, và actions chỉ ra sẽ là một connection tracking action để ký thác connection track entries và gửi packets tới flow table tiếp theo (là flow entry thứ ba liệt kê phía trên). Flow table tiếp theo được chỉ ra là table 20.

* Flow table 20:



Bản tin ICMP v4 match với flow entry cuối cùng, nghĩa là được gửi sang table 55 để xử lý tiếp.

* Flow table 55:



Chú ý flow entry thứ hai sử dụng để học MAC và IP của instance 4 dùng cho gói tin ICMP response. Flow entry thứ tư match với ICMP packet do khớp với giá trị metadata = 0x367 được thiết lập ban đầu. Packet được chuyển tiếp sang table 60 xử lý.

* Flow table 60:



Packet match với địa chỉ IP nguồn và medatata, được chuyển tiếp sang table 70 để xử lý tiếp.

* Flow table 70:



Quan sát thấy packet match với flow entry cuối với cùng giá trị metadata và địa chỉ IP nguồn, hành động là thay đổi địa chỉ MAC nguồn sang **fa:16:3e:df:bc:9b**, địa chỉ MAC đích sang **46:b9:a0:6c:04:4e**, và địa chỉ IP nguồn sang 172.16.1.60. Địa chỉ IP 172.16.1.60 chính là floating IP gán cho instance 4, còn địa chỉ MAC kia tương ứng với IP này để phục vụ cho ARP response. Thử kiểm tra với openstack command:



Địa chỉ MAC đích kia là địa chỉ của internal port br-ex trên external bridge br-ex của Compute Node. Thử kiểm tra:

****

Hành động cuối là chuyển tiếp sang table 80.

* Flow table 80:

****

Hành động duy nhất là đẩy gói ra khỏi Openflow port 1. Xem thử Openflow port 1:

****

Như vậy OpenFlow port 1 là patch port **pach-int** là port đầu cuối trên br-int để kết nối với external bridge **br-ex.** Có thể kiểm tra lại như sau:

****

Như vậy gói tin với địa chỉ MAC đích là MAC của internal port br-ex sẽ theo kết nối của cặp patch-port **patch-int – patch-ex** đi tới external bridge br-ex. Xem flow entry trên br-ex:

****

Action normal ở đây cho biết br-ex thực hiện L2 forwarding như bình thường, nghĩa là gửi gói tin qua external interface ens4 trên Compute Node đi ra ngoài.

* **Kịch bản 2**: Lưu lượng North-South sử dụng fixed IP. Trường hợp này vẫn test với df-test-instance-4, tuy nhiên trước khi tìm hiểu, detach floating IP khỏi instance này. Pipeline xử lý gói tin ở các bước xử lý đầu giống với kịch bản 1, nghĩa là theo thứ tự xử lý: **table 0 (classifier) – table 5 – table 10 (egress connection tracking table) – table 15 (egress security group table) – table 20 – table 55 – table 60**. Sự khác biệt bắt đầu từ table 60:
* Kiểm tra table 60:

****

Bản tin ICMP match với flow entry thứ hai với giá trị **0xa5f** được load vào thanh ghi REG7 đi sang flow table 75.

* Flow table 75:

****

Packet match với flow entry thứ hai do khớp giá trị reg7, hành động ở đây là thiết lập giá trị tunnel ID bằng 0x14 (VNI=20) và gửi qua OpenFlow port 4 ra ngoài. Xem thử OpenFlow port 4:

****

Như vậy packet sẽ được gửi qua tunnel enpoint trên br-int là vxlan-vtp hay OpenFlow port 4. Packet từ OpenFlow port 4 sẽ qua DATA interface của Compute Node là **ens5** sang Controller Node để định tuyến đi ra ngoài. Thử bắt gói tin trên **ens5:**

****

Quan sát thấy rằng có các bản tin ICMP echo request và response giữa google (216.58.197.110) và instance 4 (172.17.1.8) được đóng gói trong các bản tin ethernet với VXLAN ID hay VNI = 20 đã thiết lập ở trên. Các bản tin ethernet này là bản tin trao đổi giữa Compute Node và Controller, thể hiện qua địa chỉ outer IP nguồn và đích là 10.10.1.220 (địa chỉ trên DATA network của Controller) và 10.10.1.219 (địa chỉ trên DATA network của Compute).

Từ đây chuyển sang Controller Node để kiểm tra.

* Trên Controller Node, các bản tin đang xét sẽ đi vào từ DATA interface là ens5 tương tự như bên Compute Node rồi chuyển tiếp tới tunnel endpoint của br-int trên Controller Node. Xem thông tin tunnel endpoint này:

****

Như vậy tunnel endpoint này tương ứng với OpenFlow port 3.

* Theo dõi bản tin trên pipeline của br-int, bắt đầu từ các flow entry áp dụng cho OpenFlow port 3 trên table 0:

****

Packet match với flow entry thứ hai với VNI=0x14 (20), giá trị medata được thiết lập bằng 0x367 và chuyển tiếp sang table 100.

* Flow table 100:

****

Packet match với flow entry thứ hai do khớp giá trị metadata và địa chỉ MAC đích. Hành động được chỉ ra là load giá trị 0xa5f vào thanh ghi REG7 chuyển tiếp sang table 105.

* Flow table 105:

****

Packet chỉ đơn giản được chuyển tiếp sang table 115.

* Flow table 115:

****

Packet match với flow entry thứ ba do khớp giá trị thanh ghi REG7. Hành động là gói tin được đẩy ra khỏi port 249.

Xem thông tin port 249:

****

Đây chính là gateway trên router1 của subnet 172.17.1.0/24.

* Xem thông tin gateway này trên router namespace:

****

* Kiểm tra thông tin định tuyến trên router này:

****

Quan sát thấy default route là 172.16.1.1 qua cổng **qg-64cf26ac-5b**. Đây là cổng nằm trên external bridge br-ex. Xem thông tin cổng này: 

Kiểm tra thêm SNAT rule:



Quan sát thấy rằng lưu lượng đi ra khỏi cổng **qg-64cf26ac-5b** sẽ được áp dụng SNAT rule, chuyển đổi địa chỉ nguồn sang 172.16.1.53 là địa chỉ trên mạng provider của router.

* Xem thông tin port này trên br-ex:



Xem thêm flow entry trên br-ex:



Các packet trên br-ex được thực hiện L2 forwarding bình thường, chú ý rằng br-ex cũng kết nối với external interface là ens4 cho kết nối ra internet. Do đó lưu lượng sẽ qua br-ex tới ens4 rồi ra ngoài.

* **Kịch bản 3**: Lưu lượng East-West giữa hai instance trên cùng một network, cùng một Compute host. Kịch bản này tìm theo lưu lượng giữa hai instance 2 và instance 3.
* Xem thông tin hai instance:



* Kiểm tra tap interface kết nối với instance 2:



* Ping từ instance 2 sang instance 3. Bắt gói tin trên tap interface mà instance 2 kết nối với:



* Xem pipeline xử lý gói tin trên br-int ứng với port 11, bắt đầu từ table 0:



Thiết lập metadata = 0x1 rồi chuyển gói tin sang table5.

* Flow table 5: gói tin match với flow entry đầu tiên, chuyển tiếp sang table 10 xử lý:



* Flow table 10: gói tin được chuyển tiếp sang **egress security group table** **15**:



* Security group rule được thực hiện tại table 15 rồi chuyển tiếp gói tin sang table 20:



* Flow table 20: gói tin match với flow entry cuối cùng, chuyển tiếp sang table 55:



* Flow table 55: gói tin match với flow entry thứ hai với metadata = 0x1 và địa chỉ MAC đích là địa chỉ MAC của instance 3. Hành động áp dụng cho gói tin ở đây là load giá trị **0xa5d** vào thanh ghi REG7 và chuyển tiếp sang table 75 xử lý:



* Flow table 75: gói tin match với flow entry đầu tiên với giá trị **reg7=0xa5d** và được chuyển tiếp sang table 105:



* Flow table 105: đây là **ingress connection tracking** table gói tin match với flow entry đầu tiên với reg7=0xa5d và được chuyển tiếp sang table 110:



* Flow table 110: đây là **ingress security group table** triển khai ingress security group rule áp dụng lên gói tin. Sau khi xử lý, gói tin được chuyển tiếp sang table 115:



* Flow table 115: gói tin match với flow entry đầu tiên với **reg7=0xa5d**, hành động cuối cùng là đẩy gói tin ra khỏi cổng 10:



* Kiểm tra OpenFlow port 10 trên br-int, thấy rằng đây là tap interface kết nối với instance 3:



* **Kịch bản 4**: Lưu lượng East-West giữa hai instance trên cùng một network, khác Compute host. Kịch bản này tìm theo lưu lượng giữa hai instance 1 và instance 2:
* Lấy thông tin các interface của hai instance:



* Lấy thông tin tap interface mà instance 1 (trên controller) và instance 2 (trên compute) kết nối tới:





* Ping từ instance 1 sang instance 2, trên Controller Node, bắt gói tin trên tap interface mà instance 1 kết nối tới (OpenFlow port 250):



* Trên Controller Node, bắt đầu xem gói tin nhận được trên OpenFlow port 250 đi qua pipeline của bridge br-int như thế nào. Trước hết liệt kê các flow entry áp dụng cho các gói tin nhận được trên OpenFlow port 250. Flow entry cuối cùng đưa ra hành động mặc định cho biết br-int liên lạc với local controller để xử lý gói tin:



* Tương tự như các kịch bản trên, gói tin nhận được trên tap interface ở thời điểm ban đầu sẽ được xử lý đi qua các flow table với thứ tự sau: **table 0 (classifier) – table 5 – table 10 (egress connection tracking) – table 15 (egress security group) – table 20 – table 55**. Metadata để trao đổi giữa các flow table được thiết lập ở thời điểm ban đầu là 0x1.













Tại flow table 55, gói tin match với flow entry thứ tư như chỉ ra trên đây, hành động thực thi là load giá trị **0xa5b** vào thanh ghi **reg7** và chuyển tiếp sang table 75 để xử lý.

* Flow table 75: gói tin match với flow entry thứ ba với giá trị **reg7=0xa5b** như thiết lập ở bước trước. Hành động được áp dụng là load giá trị tunnel ID bằng 0x46 (70) vào trường thông tin tun\_id, rồi đẩy ra khỏi OpenFlow port 3 của br-int.



* Kiểm tra OpenFlow port 3, thấy rằng đây là tunnel ID trên br-int của Controller Node, từ đây gói tin sẽ đi qua DATA interface là ens5 sang Compute Node 1, nơi chứa instance 2:



Bắt gói tin nhận được trên ens5, thấy rằng có các bản tin ethernet được đóng gói VXLAN với VXLAN ID hay VNI=70 như đã thiết lập ở bước trên. Các bản tin này có outer IP là địa chỉ trên mạng DATA của Controller Node và Compute Node (**10.10.1.220** và **10.10.1.219**) và inner IP là địa chỉ trên mạng private mà hai instance kết nối tới (**10.0.0.11** và **10.0.0.13**).



* Các bước tiếp theo thực hiện trên Compute Node. Xác nhận tunnel endpoint trên br-int tương ứng với OpenFlow port 4:



* Xem xét pipeline bắt đầu từ table 0, gói tin match với entry bên dưới với tunnel ID = 70, in\_port là OpenFlow port 4. Giá trị medata được thiết lập bằng 0x1, hành động cuối cùng là chuyển tiếp sang table 100.



* Flow table 100: gói tin match với flow entry đầu tiên được lọc ra như bên dưới, hành động là load giá trị 0xa5b vào thanh ghi reg7 và chuyển tiếp sang table 105.



* Flow table 105, các bước tiếp theo giống với kịch bản thứ ba: gói tin chuyển tiếp sang **ingress security group table** 110 để áp dụng các security group rule lên gói tin, sau đó chuyển tiếp sang table 115, hành động cuối cùng trên table 115 là đẩy gói tin ra khỏi cổng 11 chính là tap interface kết nối với instance 2.









* **Kịch bản 5**: Lưu lượng East-West giữa hai instance trên hai network có gateway trên cùng một router, cùng một Compute host. Kịch bản này áp dụng với instance 2 (thuộc private network) và instance 4 (thuộc private-1 network) trên Compute Node. Trước tiên lấy thông tin các interface mà các instance kết nối tới.



* Ping từ instance 2 sang instance 4 rồi bắt gói tin nhận được trên tap interface mà instance 2 kết nối tới:





Thấy rằng các bản tin ICMP echo request nhận được trên tap interface nối với instance 2 (Openflow port 11) có địa chỉ MAC nguồn khớp với địa chỉ MAC của instance 2, tuy nhiên địa chỉ MAC đích là **fa:16:3e:3c:b1:e2** lại khác với địa chỉ MAC của instance 4. Đây chính là địa chỉ gateway của subnet **private-subnet** trên **router1 (10.0.0.1),** subnet mà instance 2 kết nối vào. Kiểm tra gateway này trên router namespace thuộc Controller Node như sau:



* Các bước đầu tiên của pipeline xử lý gói tin từ instance 2 vẫn là: table 0 – table 5 – table 10 – table 15 – table 20 – table 55 với giá trị metadata khởi tạo là 0x1.











Tại table 55, gói tin match với flow entry thứ tư chỉ ra dưới dây do khớp giá trị metadata và địa chỉ MAC đích chỉ ra bởi tcpdump, hành động cuối là chuyển tiếp gói tin qua flow table 60.



* Flow table 60 triển khai với chức năng của distributed virtual router. Gói tin đang xét tới đây match với flow entry thứ hai với địa chỉ IP đích thuộc dải 172.17.1.0/24, hành động là đổi địa chỉ MAC nguồn sang fa:16:3e:5a:4a:28, thiết lập lại **metadata=0x367** rồi chuyển sang table 65.



Quan sát lại thông tin router namespace thu được ở bước trước, thấy rằng **fa:16:3e:5a:4a:28**  chính là địa chỉ MAC của gateway cho subnet **private1-subnet (172.17.1.1),** subnet mà instance 4 kết nối tới.

* Flow table 65: gói tin match với flow entry thứ hai do khớp với metadata thiết lập lại là 0x367 và địa chỉ IP đích, hành động thực thi ở đây là đổi địa chỉ MAC đích của gói tin sang **fa:16:3e:90:04:16** là địa chỉ MAC của instance 4 (thay cho địa chỉ MAC của gateway của subnet 10.0.0.0/26), load giá trị **0xa60** vào thanh ghi **reg7** và cuối cùng là chuyển tiếp sang table 75.



* Flow table 75: gói tin match với flow entry thứ 4, hành động là chuyển tiếp sang table 105.



* Flow table 105 (**ingress connection tracking**): gói tin match với flow entry thứ hai được chuyển tiếp sang table 110.



* Tại flow table 110 (ingress security group), gói tin được áp dụng các security group rule (đã chuyển đổi sang flow entry), rồi chuyển tiếp qua table 115.



* Gói tin match với flow entry thứ hai liệt kê dưới đây (reg7=0xa60), và được đẩy ra khỏi port 12, chính là tap interface nối với instance 4.





* **Kịch bản 6**: Lưu lượng East-West giữa hai instance trên hai network có gateway trên cùng một router, khác Compute host. Kịch bản này thực hiện với hai instance là instance 1 và instance 4. Trước hết xem lại interface trên các instance này.



* Trên Controller Node, bắt gói tin trên tap interface nối với instance 1, địa chỉ MAC đích lúc này.



* Pipeline các bước đầu tiên tương tự kịch bản 5: **table 0 – table 5 – table 10 – table 15 – table 20 – table 55 – table 60 – table 65 – table 75**, với giá trị metadata ban đầu 0x1, sau đó tại **distributed virtual router table** 60 đổi sang 0x367.













Trên table 60, đổi địa chỉ MAC nguồn sang địa chỉ MAC của gateway cho subnet 172.17.1.0/24 trên router1.



Trên table 65, đổi địa chỉ MAC đích sang địa chỉ MAC của instance 4.



* Tại table 75, gói tin match với flow entry với giá trị reg7=0xa60 được thiết lập tại table 65. Hành động cuối cùng là load giá trị tunnel ID 0x14 (20) vào trường tun\_id, rồi đẩy gói tin ra khỏi OpenFlow port 3.



OpenFlow port 3 chính là tunnel endpoint trên Controller Node. Từ đây gói tin chuyển tiếp qua DATA interface ens5 của Controller Node sang Compute Node theo đường DATA tunnel.



* Các bước tiếp sau đây thực hiện trên Compute Node. Đầu tiên xem lại VXLAN tunnel endpoint và theo dấu pipeline của gói tin nhận trên OpenFlow port tương ứng.





Gói tin được thiết lập metadata bằng 0x367 và chuyển tiếp sang table 100.

* Tại table 100, gói tin match flow entry đầu tiên sẽ được load giá trị 0xa60 vào thanh ghi reg7 và chuyển tiếp sang table 105. (flow entry thứ hai dành cho bản tin broadcast nên không match).



* Flow table 105 (**ingress connection tracking**) Flow entry match với gói tin dưới đây chuyển tiếp gói tin sang table 110.



* Security group rule áp dụng cho gói tin trên **ingress security group table** 110, chuyển tiếp sang table 115.

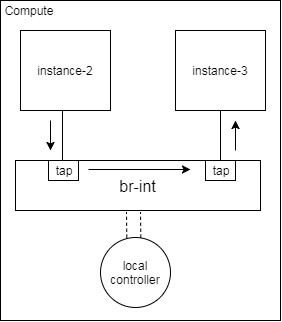


* Trên table 115, gói tin bị đẩy ra khỏi OpenFlow port 12, chính là tap interface kết nối với instance 4.

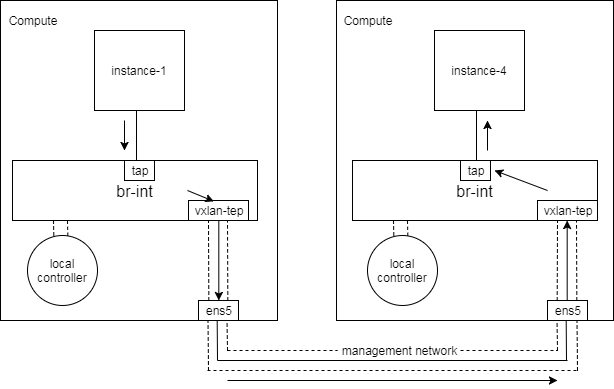




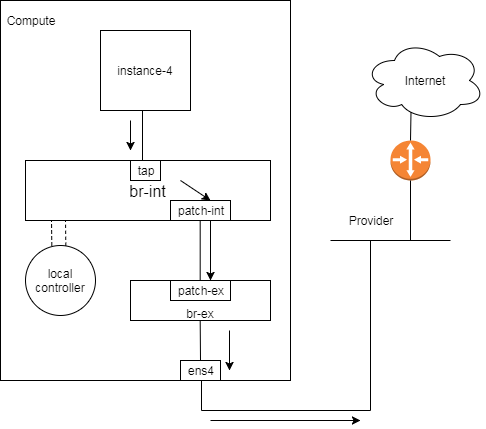
* Tổng quát
* Lưu lượng East-West giữa các instance cùng Compute Node: do hai trường hợp các instance cùng subnet hoặc khác subnet, sự khác biệt nằm ở pipeline xử lý tại integration bridge br-int nên nhìn ở mức tổng quát, lưu lượng sẽ như hình dưới. Trường hợp khác Compute Node cũng khái quát tương tự.



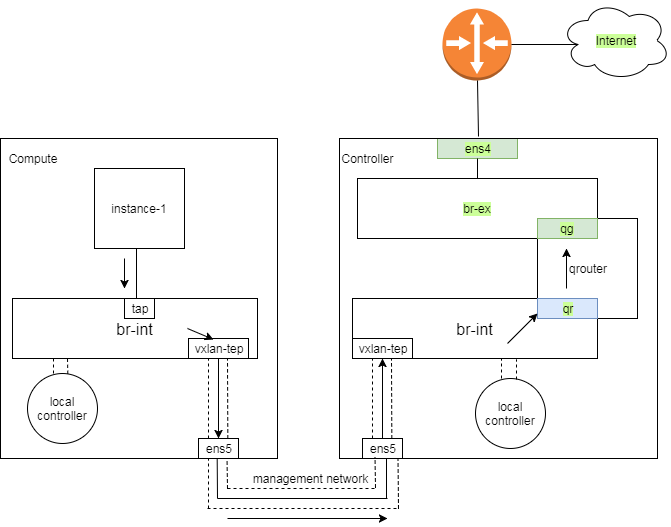
* Lưu lượng East-West giữa các instance khác Compute Node:



* Lưu lượng North-South sử dụng Floating IP:

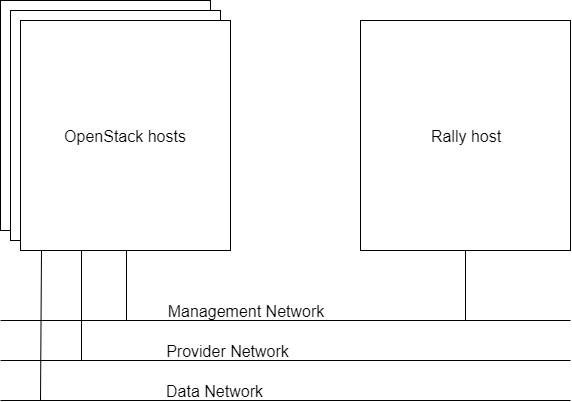


* Lưu lượng North-South sử dụng Fixed IP:



## 3. Benchmark Dragonflow sử dụng Rally

### *3.1. Mô hình cài đặt*



* Cài đặt rally trên host riêng kết nối vào mạng Management Network trên hệ thống OpenStack.
* Cài đặt rally theo hướng dẫn:

[http://rally.readthedocs.io/en/latest/install\_and\_upgrade/install.html#automated-installation](http://rally.readthedocs.io/en/latest/install_and_upgrade/install.html%23automated-installation)

### *3.2. Kịch bản benchmark*

|  |  |
| --- | --- |
| Task | Chi tiết |
| NeutronNetworks.create\_and\_delete\_floating\_ips | Lặp lại 10 lần, chạy đồng thời 5 kịch bản lặp |
| NeutronNetworks.create\_and\_delete\_networks | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronNetworks.create\_and\_delete\_ports | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronNetworks.create\_and\_delete\_routers | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronNetworks.create\_and\_delete\_subnets | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronNetworks.create\_and\_list\_floating\_ips | Lặp lại 10 lần, chạy đồng thời 5 kịch bản lặp |
| NeutronNetworks.create\_and\_list\_networks | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronNetworks.create\_and\_list\_ports | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronNetworks.create\_and\_list\_routers | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronNetworks.create\_and\_list\_subnets | Lặp lại 10 lần, chạy đồng thời 5 kịch bản lặp |
| NeutronNetworks.create\_and\_update\_networks | Lặp lại 10 lần, chạy đồng thời 5 kịch bản lặp |
| NeutronNetworks.create\_and\_update\_ports | Lặp lại 10 lần, chạy đồng thời 5 kịch bản lặp |
| NeutronNetworks.create\_and\_update\_routers | Lặp lại 10 lần, chạy đồng thời 5 kịch bản lặp |
| NeutronNetworks.create\_and\_update\_subnets | Lặp lại 10 lần, chạy đồng thời 5 kịch bản lặp |
| NeutronSecurityGroup.create\_and\_delete\_security\_groups | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronSecurityGroup.create\_and\_list\_security\_groups | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NeutronSecurityGroup.create\_and\_update\_security\_groups | Lặp lại 30 lần, chạy đồng thời 10 kịch bản lặp |
| NovaServers.boot\_and\_associate\_floating\_ip | Lặp lại 5 lần |
| NovaServers.boot\_and\_delete\_multiple\_servers | Lặp lại 2 lần, mỗi lần 4 server |
| NovaServers.boot\_and\_delete\_server | Lặp lại 10 lần, chạy đồng thời 2 kịch bản lặp |

* Kịch bản benchmark:

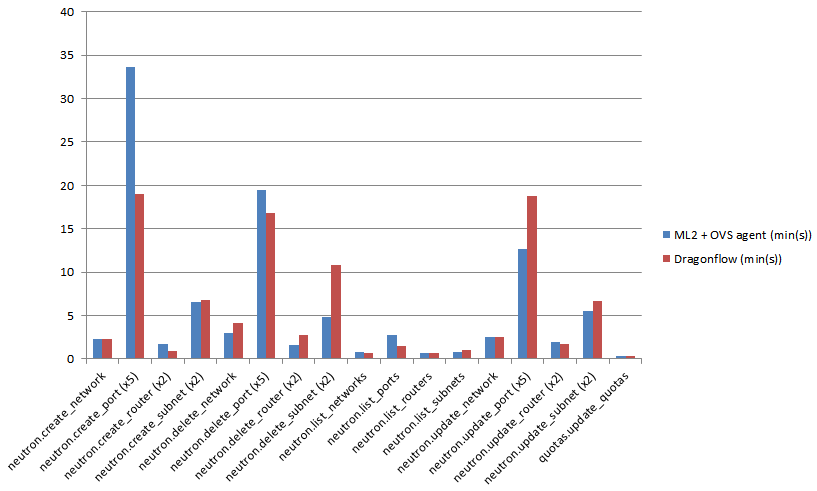


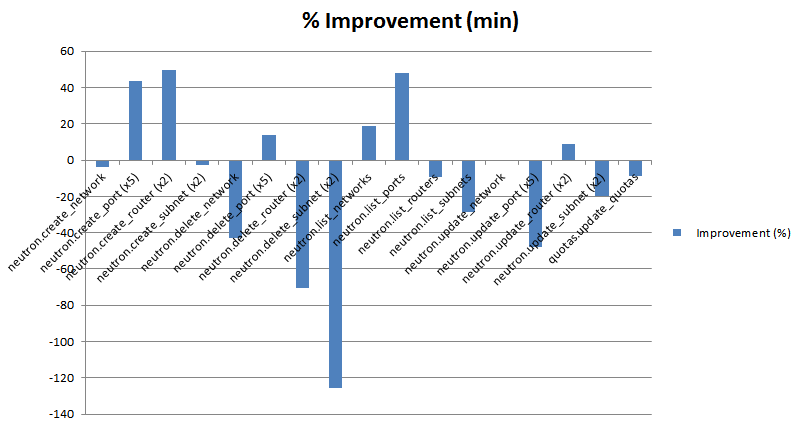
### *3.3. Kết quả benchmark và so sánh với hướng tiếp cận không sử dụng SDN*

* Trường hợp nhanh nhất:

|  |  |  |  |
| --- | --- | --- | --- |
| API | ML2 + OVS agent (min(s)) | Dragonflow  (min(s)) | % Improvement |
| neutron.create\_network | 2.267 | 2.35 | -3.66122629 |
| neutron.create\_port (x5) | 33.617 | 18.956 | 43.61186305 |
| neutron.create\_router (x2) | 1.739 | 0.873 | 49.79873491 |
| neutron.create\_subnet (x2) | 6.582 | 6.747 | -2.506836828 |
| neutron.delete\_network | 2.94 | 4.2 | -42.85714286 |
| neutron.delete\_port (x5) | 19.457 | 16.8 | 13.65575371 |
| neutron.delete\_router (x2) | 1.616 | 2.759 | -70.73019802 |
| neutron.delete\_subnet (x2) | 4.822 | 10.879 | -125.6117793 |
| neutron.list\_networks | 0.862 | 0.7 | 18.79350348 |
| neutron.list\_ports | 2.804 | 1.466 | 47.71754636 |
| neutron.list\_routers | 0.667 | 0.73 | -9.445277361 |
| neutron.list\_subnets | 0.798 | 1.027 | -28.69674185 |
| neutron.update\_network | 2.558 | 2.558 | 0 |
| neutron.update\_port (x5) | 12.645 | 18.723 | -48.06642942 |
| neutron.update\_router (x2) | 1.906 | 1.738 | 8.814270724 |
| neutron.update\_subnet (x2) | 5.54 | 6.647 | -19.98194946 |
| quotas.update\_quotas | 0.356 | 0.387 | -8.707865169 |

* Quan sát trên đồ thị:

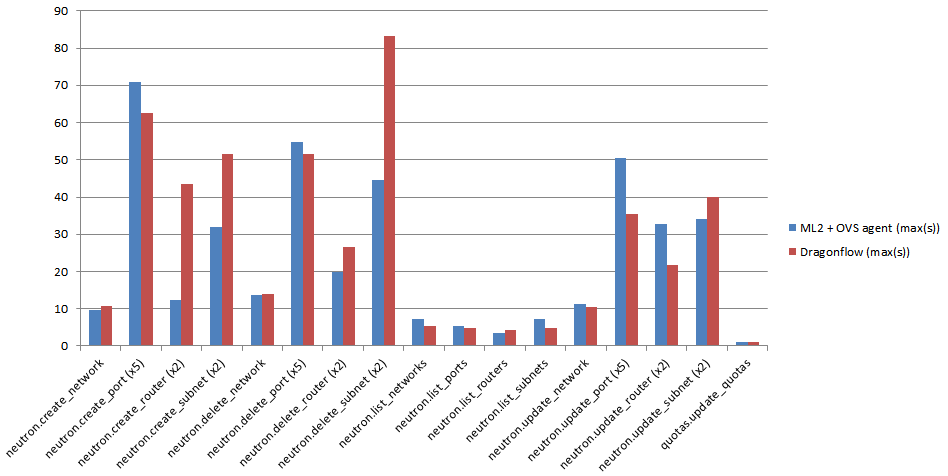


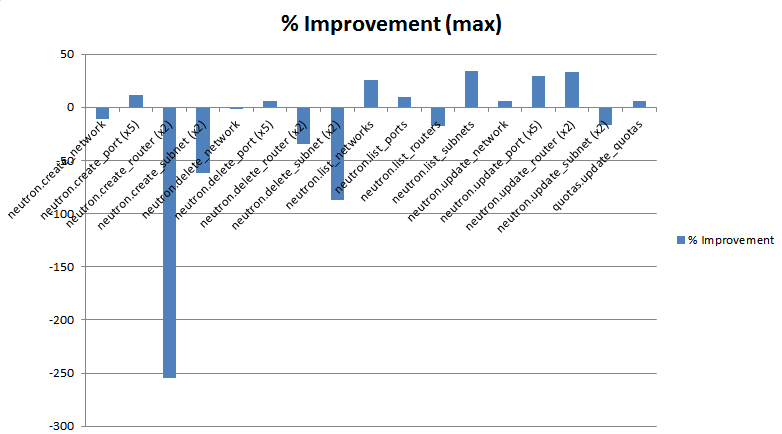


* Trường hợp lâu nhất:

|  |  |  |  |
| --- | --- | --- | --- |
| API | ML2 + OVS agent (max(s)) | Dragonflow  (max(s)) | % Improvement |
| neutron.create\_network | 9.62 | 10.718 | -11.41372141 |
| neutron.create\_port (x5) | 70.761 | 62.54 | 11.61798166 |
| neutron.create\_router (x2) | 12.262 | 43.543 | -255.1052031 |
| neutron.create\_subnet (x2) | 31.871 | 51.619 | -61.96228546 |
| neutron.delete\_network | 13.788 | 13.956 | -1.218450827 |
| neutron.delete\_port (x5) | 54.712 | 51.502 | 5.867085831 |
| neutron.delete\_router (x2) | 19.773 | 26.654 | -34.79997977 |
| neutron.delete\_subnet (x2) | 44.55 | 83.303 | -86.98765432 |
| neutron.list\_networks | 7.228 | 5.353 | 25.94078583 |
| neutron.list\_ports | 5.236 | 4.71 | 10.04583652 |
| neutron.list\_routers | 3.573 | 4.208 | -17.77218024 |
| neutron.list\_subnets | 7.244 | 4.742 | 34.53892877 |
| neutron.update\_network | 11.177 | 10.494 | 6.110763174 |
| neutron.update\_port (x5) | 50.437 | 35.464 | 29.68653964 |
| neutron.update\_router (x2) | 32.624 | 21.852 | 33.01863659 |
| neutron.update\_subnet (x2) | 34.219 | 39.925 | -16.67494667 |
| quotas.update\_quotas | 1.07 | 1.006 | 5.981308411 |

* Quan sát trên đồ thị:





* Trường hợp trung bình:

|  |  |  |  |
| --- | --- | --- | --- |
| API | ML2 + OVS agent (avg(s)) | Dragonflow  (avg(s)) | % Improvement |
| neutron.create\_network | 4.461 | 5.277 | -18.29186281 |
| neutron.create\_port (x5) | 47.942 | 43.503 | 9.259104752 |
| neutron.create\_router (x2) | 7.053 | 11.585 | -64.25634482 |
| neutron.create\_subnet (x2) | 15.508 | 29.159 | -88.02553521 |
| neutron.delete\_network | 7.722 | 9.197 | -19.1012691 |
| neutron.delete\_port (x5) | 35.414 | 31.394 | 11.35144293 |
| neutron.delete\_router (x2) | 8.977 | 10.55 | -17.52255765 |
| neutron.delete\_subnet (x2) | 19.33 | 42.926 | -122.0693223 |
| neutron.list\_networks | 3.567 | 2.899 | 18.72722175 |
| neutron.list\_ports | 4.231 | 3.582 | 15.33916332 |
| neutron.list\_routers | 1.945 | 1.928 | 0.87403599 |
| neutron.list\_subnets | 3.34 | 2.569 | 23.08383234 |
| neutron.update\_network | 5.779 | 6.302 | -9.050008652 |
| neutron.update\_port (x5) | 27.064 | 26.703 | 1.333875259 |
| neutron.update\_router (x2) | 16.724 | 11.295 | 32.46232959 |
| neutron.update\_subnet (x2) | 18.247 | 23.935 | -31.17224749 |
| quotas.update\_quotas | 0.643 | 0.694 | -7.931570762 |

* Quan sát trên đồ thị:

