Isoflat: Flat Provider Network Multiplexing and Firewalling in OpenStack Cloud

Ruipeng Zhang, Mengjun Xie, Li Yang
Department of Computer Science and Engineering
The University of Tennessee at Chattanooga
Chattanooga, Tennessee, USA
smj793@mocs.utc.edu, mengjun-xie@utc.edu, Li-Yang@utc.edu

***Abstract*—Networking is one of the key enablers of cloud computing and its security is essential for multi-tenant clouds. As a widely used open source solution to cloud computing, OpenStack allows computing resources to connect to the physical network infrastructure through provider networks for performance and reliability considerations. However, OpenStack users are stuck with either VLAN provider networks that are complex to configure and manage or flat networks that are not isolated and have the limitation on interface multiplexing. To address this problem, in this paper, we propose a new mechanism called Isoflat, which extends OpenStack's ability for creating flat provider networks with both configuration simplicity and flexible isolation capability. Our evaluation results show that a provider network with Isoflat can achieve similar network performance as a flat or VLAN provider network. Our results also show that the Isoflat firewall has much less impact on throughput performance than security group.**

I. Introduction

With the increasing popularity of big data and cloud computing, network sharing and isolation problems are receiving growing attention. It is essential for cloud service providers, especially Infrastructure-as-a-Service (IaaS) providers, to assure their tenants of relatively isolated network environments regarding data and performance, because a tenant has little control over how other tenants behave to the shared underlying physical infrastructure.

A number of existing approaches to network isolation such as VLAN (Virtual LAN) and VRF (Virtual Routing and Forwarding) Lite do not scale well in the context of commercial clouds where tens of thousands of tenants coexist [1]. They are also difficult for private or home cloud deployment because of their complex configuration or a number of requirements on networking devices. Software approaches such as overlay networking and tunneling offer a better solution to the scaling issue, but integrating them could bring technical complications and communication overhead [2]. Therefore, a scalable, flexible, lightweight, and easy-to-configure network isolation approach is more practical and acceptable for users with demands for provider network security.

OpenStack is a widely used open source solution to managing cloud computing infrastructures including data center networks. For direct access to the underlying network infrastructure, OpenStack offers flat (non-VLAN) or VLAN provider networks to its tenants. Although VLAN provider networks support network isolation, they inherit the scalability limitation and configuration complexity from VLAN. Flat provider networks, on the other hand, are easy to configure but lack isolation capability. OpenStack also poses a limitation on the number of flat provider networks that can be created over a single network interface. Usually, only one flat provider network will be created if each compute node has only one physical network interface for provider networks. It is ideal to remove that limitation and offer network isolation capability for flat provider networks while retaining their advantages.

In this paper, we propose Isoflat (Isolated flat provider network), a new mechanism for managing flat provider networks and enabling firewall function for OpenStack clouds. Isoflat tackles OpenStack's limitation on creating multiple flat provider networks on a single network interface by utilizing software switches and virtual interfaces. Isoflat's multiplexing capability lowers OpenStack's requirement on the infrastructure as fewer interfaces are needed to create the same number of provider networks. In the meantime, Isoflat supplements OpenStack's network isolation capability by introducing built-in firewall functionality to provider networks. Isoflat firewalls work in a way similar to OpenStack security group. However, its packet filtering runs close to network interfaces, which results in less overhead compared to the security group since the number of interfaces to be filtered is relatively smaller. Moreover, Isoflat can update its firewall rules when the provider network configuration changes, which reduces security risks caused by firewall configuration inconsistency. Finally, Isoflat firewall rules can only be modified by the network owner so that other tenants cannot bypass Isoflat firewalls by tampering rules as they could with their security groups.

Our evaluation results show that Isoflat's interface multiplexing only incurs negligible throughput performance overhead for TCP traffic compared to existing flat network or VLAN solutions when the provider networks share the same physical interface. In terms of firewall processing, Isoflat firewall exhibits significant throughput improvement compared to the security group.

The rest of this paper is organized as follows: We briefly introduce necessary background information in Section II. We then present the design of Isoflat in Section III and its implementation in Section IV. We detail the evaluation of Isoflat in Section V. We describe related work in Section VI and conclude this paper in Section VII.

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

II. Background

OpenStack and Neutron. OpenStack includes a series of services for creating different types of cloud resources, orchestrating cloud resources, and monitoring resource usage. As a networking service for OpenStack, Neutron provides networking infrastructure including switches, routers, and firewalls to computing devices by utilizing software-defined networking (SDN) technologies. This paper presents an extension to current Neutron capabilities for creating provider networks and setting up firewalls.

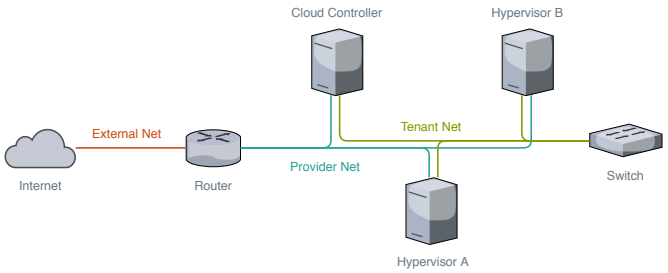

Fig. 1: Simplified view of OpenStack networks

Provider Network. There are two types of OpenStack networks managed by Neutron: provider network and tenant network. A tenant network, as its name suggests, acts within a single OpenStack tenant and is isolated from other tenants. Tenant network isolation is usually realized through overlay network protocols such as Virtual Extensible LAN (VXLAN) and Generic Routing Encapsulation (GRE).

Unlike tenant networks, OpenStack provider networks are integrated with existing physical networks. By using provider networks, computing resources can directly access physical network infrastructure for better performance and reliability. OpenStack currently supports VLAN (IEEE 802.1Q tagged) and flat (untagged) provider network types. Provider networks that directly connect to exterior network spaces also serve as external networks, whereby tenant traffic can be routed to the outside world. In addition, administrators can build self-service networks from provider networks where computing resources are only exposed externally by floating IP addresses and NAT. Figure 1 depicts a simplified view of the scope as well as connectivity of tenant and provider networks in an OpenStack environment.

Bridges and Ports. Figure 2 [3] and Figure 3 [4] depict the networking components and connectivity in an OpenStack compute node (similar in other compute nodes if multiple compute nodes exist) with two VLAN provider networks using Open vSwitch (OVS) and Linux bridge mechanism drivers respectively. The Neutron port connecting to a Nova instance or a virtual network device is an abstraction of the TAP interface that is written by a VM or a networking service process. Neutron also employs software switches such as Linux bridge and OVS to organize networks and it provides packet filtering on these switches through iptables or OpenFlow rules.

When driven by the Linux bridge mechanism driver, Neutron connects computing resources to a provider network by plugging the provider network interface to the Linux bridge that manages TAP ports. The connection establishment is a bit

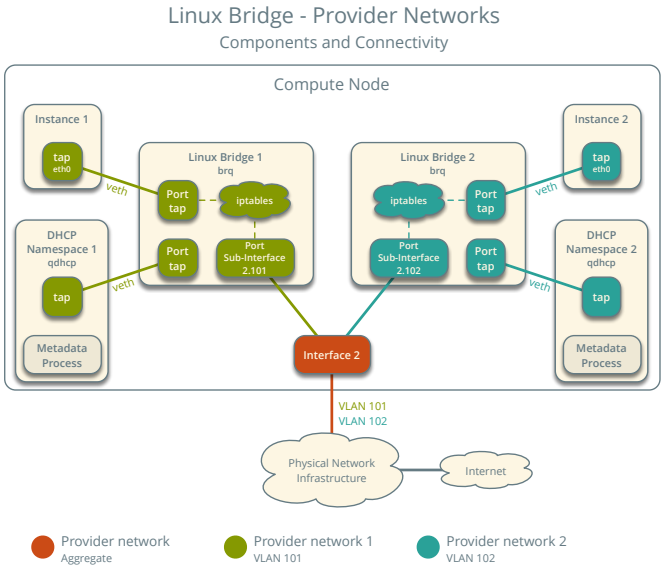

Fig. 2: OpenStack provider networking with Linux bridge

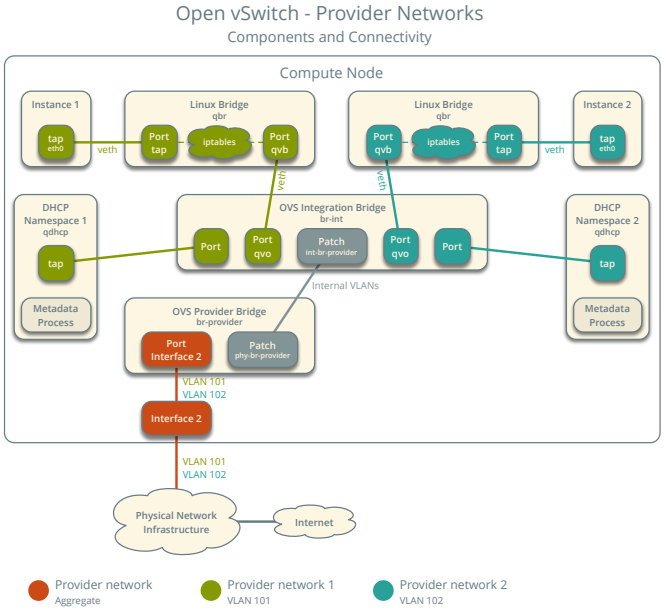

Fig. 3: OpenStack provider networking with Open vSwitch

more complicated with Open vSwitch. In this case, Neutron introduces an integration bridge and several provider bridges. An integration bridge is used for internal VLAN to tag (or untag) the network traffic from (or to) the instances. A provider bridge is used to swap internal VLAN tags with actual VLAN tags (if VLANs are used) and pass packets to a physical interface. To connect computing or networking devices to a provider network, Neutron creates a pair of patch or Veth (Virtual Ethernet) ports and attach them to their corresponding integration bridge and provider bridge. This pair of ports acts as the two ends of a cable to enable network traffic to flow from one end to the other.

Security Group. Security group is a network security feature provided by most cloud providers. One security group may contain one or multiple security group rules, which define the source, destination, protocol type and direction of the

allowed network traffic from/to a virtual instance [5]. By associating computing resources with security groups rules, users can set up virtual firewalls in hypervisors.

OpenStack's security groups are implemented through firewall drivers that utilize Netfilter or OpenFlow for packet filtering. When a security group is updated, Neutron refreshes the corresponding firewall rules deployed on the hypervisors through Neutron ML2 (Modular Layer 2) agents. The firewall drivers loaded by the agents are responsible for translating security group rules to Netfilter or OpenFlow rules and applying them to corresponding network interfaces. These firewall rules are port based and they are applied to inbound traffic to and outbound traffic from a VM instance. The default action for a security group is to deny packets that do not match any security group rules.

III. System Design

Isoflat aims at 1) enabling multiple flat provider networks on a single network interface and 2) providing layer 3 and layer 4 isolation for those provider networks. In this section, we discuss Isoflat software architecture, network model, and firewall design in detail.

A. Isoflat Architecture and Workflow

The architecture of Isoflat is shown in Figure 4. Isoflat extends Neutron's functionality. It consists of a client extension, a service plugin, and an agent extension in Neutron's subsystems. The Isoflat client extension supplies a management CLI (Command-Line Interface) to the Isoflat firewall. Commands can be issued by administrators from a client remotely through REST (REpresentational State Transfer) API. The Isoflat service plugin, which resides in the Neutron server, listens on the Isoflat REST API endpoints for Isoflat commands. It also stores changes made to the firewall in the Neutron database and dispatches SDN operations to the Isoflat agents through RPC (Remote Procedure Call). Isoflat plugin agents, which run on hypervisors, are responsible for multiplexing a network interface and deploying packet filtering rules through Isoflat firewall drivers.

The workflow of Isoflat is illustrated in Figure 5. Prior to the deployment of Isoflat, the administrator needs to specify bridge interface to provider network mapping in a configuration file in a way similar to configuring a normal flat provider network in OpenStack. The only difference is that a user can assign multiple provider networks to the same bridge in Isoflat. During deployment, Isoflat agents will read the mapping, create virtual network interfaces, and connect them to the Neutron managed bridges, which is depicted as ① in Figure 5. Later, if the owner of a created provider network wants to create or delete a firewall rule, she will need to send a request from the Isoflat client extension. The request will be directed to the Isoflat service plugin and then delegated by the Isoflat plugin agents. The agents will find the corresponding network interfaces through the mapping and then update the associated firewall rules. This user interaction process is denoted as ② in the figure.

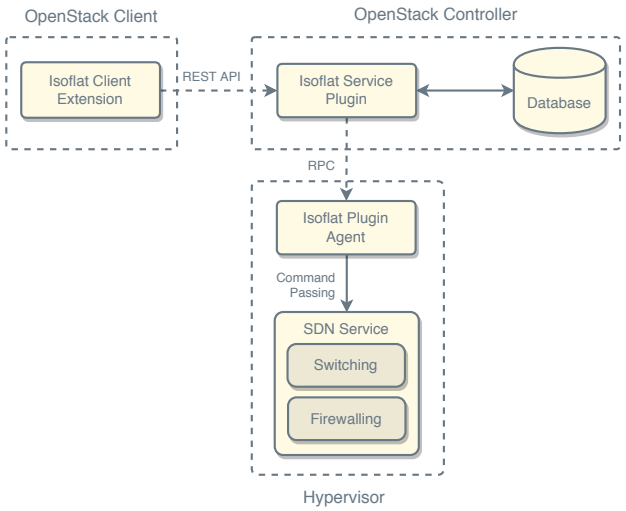

Fig. 4: Isoflat architecture

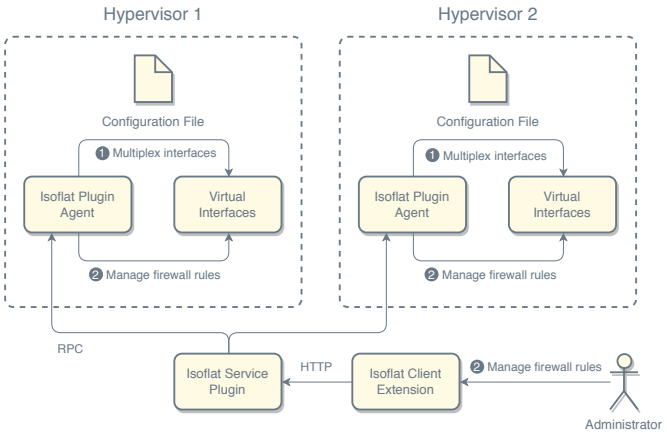

Fig. 5: Isoflat's workflow during initialization and user interaction

B. Network Model

To tackle the limitation of setting up multiple flat provider networks on a single network interface NIC_S, Isoflat employs a technique called network interface multiplexing, through which Isoflat creates multiple virtual network interfaces NIC_V that Neutron can bind to and build provider networks upon. Figure 6 and Figure 7 depict our proposed network models (in one compute node) with interface multiplexing for OVS-based and Linux bridge-based Neutron deployment cases respectively. In our design, we assume that the bridge BRG_S, which would be used as the sole flat provider network bridge in a common OpenStack deployment using flat provider network, has already been created and that the physical interface NIC_S connecting to the physical infrastructure has been attached to the bridge. By searching the bridge to network mapping in the configuration file, Isoflat agents can figure out the number of provider networks to be supported by BRG_S and create the same number of virtual interface (e.g., veth) pairs (NIC_V). Next, the agents connect the Neutron managed bridges (e.g., brq in Figure 7) to BRG_S using those NIC_V pairs and update the bridge mapping in the Neutron ML2 plugin configuration. If the OVS driver is used, Isoflat will create the same number of provider bridges BRG_P (in Figure 6) and connect NIC_V

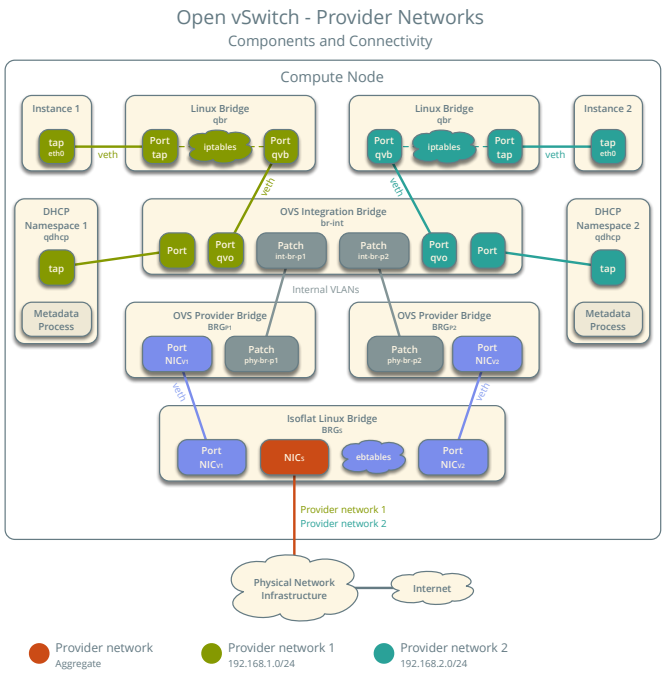

Fig. 6: OpenStack provider networking with Isoflat using Open vSwitch driver

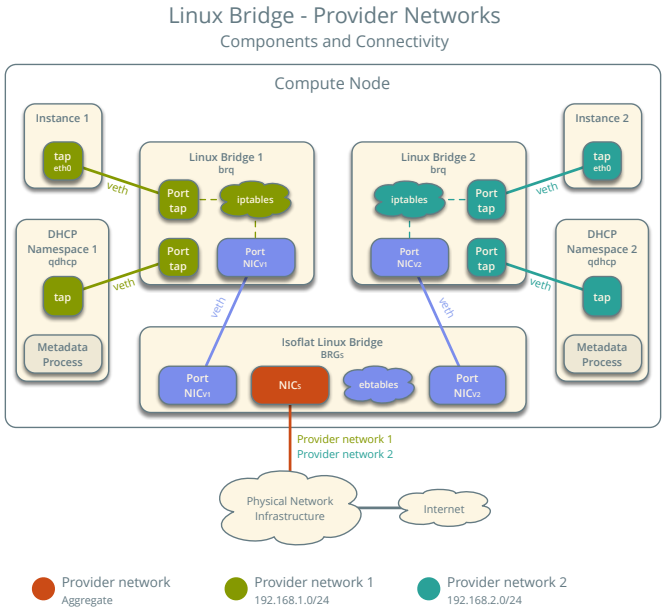

Fig. 7: OpenStack provider networking with Isoflat using Linux bridge driver

to them. After that, data links between VM instances and the physical interface NIC_S are established.

With network interface multiplexing, multiple flat provider networks can be created while Neutron's logic of associating one provider network to one network bridge remains unchanged, since each provider network still binds to one network bridge. Moreover, all provider network traffic still goes through the original interface NIC_S thanks to the NIC_V pairs and bridge BRG_S.

C. Provider Network Firewall

The network interface multiplexing also provides an opportunity for building firewalls at the provider network level. As each pair of NIC_V builds a tunnel between a BRG_P (or brq) and BRG_S, the interface NIC_V on BRG_S for a given provider network can see all the network traffic from/to that provider network in the hypervisor where it resides. Hence, we can apply firewall rules to network traffic of a provider network passing through the virtual interface. The filter implementation depends on the underlying network driver (e.g., Linux bridge or Open vSwitch). For Linux bridge-based deployment, the Netfilter is a good choice to filter packets through the virtual interfaces. For Open vSwitch-based deployment, OpenFlow is an appropriate tool for constructing complex packet filters.

To build an implementation-agnostic firewall, Isoflat uses firewall rules that contain key layer 3 and layer 4 properties such as protocol type and destination IP address space to match and filter network traffic. Those properties identify the source and/or destination, protocol type and port numbers of the packets to be filtered. Isoflat firewall enforces no restriction on traffic by default with the rationale that interruption to existing tenants should be minimized and tenant owners should be given full control as to what traffic to block. Upon receiving new firewall rules, Isoflat agents will translate them into Netfilter rules or OpenFlow table entries and deploy them on the target node(s) using the corresponding firewall driver. As an Isoflat firewall rule only applies to one flat provider network, the provider network's NIC_V will also become a constraint in the translation results so that the firewall driver will filter packets for the correct provider network.

The instance placement does not affect the effectiveness of Isoflat firewalls on packet filtering between provider networks. Packets that originate from one provider network to another have to pass the Isoflat Linux bridge BRG_S even if the two communicating instances reside in the same node. If the Linux bridge driver is used, each provider network has a separate Neutron managed provider bridge brq; Any packet sending from one provider network to another has to travel through the intermediate bridge BRG_S. When the OVS driver is used, a packet P will be tagged with an internal VLAN tag once it enters the OVS integration bridge *br-int*; The internal VLAN tag will be used to forward the packet to the correct OVS provider bridge BRG_P; And the packet eventually passes the BRG_S. Therefore, in both cases, the Isoflat firewall rules will be applied to packets flowing from one provider network to another.

IV. Implementation

We have implemented an Isoflat prototype for OpenStack Pike with Neutron version 12.0. The code base of Isoflat consists of about 1,300 lines of Python code[1]. Our current Isoflat agent implementation only supports OVS-based Neutron deployment. Implementation for other network models can be easily derived from the current implementation.

Isoflat consumes a separate configuration file from Neutron to look up for BRG_S to provider network mapping. The

[1]Isoflat's source code is available at https://github.com/ppoffice/isoflat

configuration file should be set accessible to only OpenStack administrators. Based on the mapping, Isoflat will first generate veth pairs and bridge interfaces and then map them to the flat provider networks as described in Section III. The updated bridge-to-network mapping is saved to the same configuration file that is later loaded by the ML2 agents for creating provider networks.

We chose Netfilter as the firewall driver for Isoflat in our implementation. Although *iptables* is a widely used firewall application based on Netfilter in Linux, our evaluation found that it has limitations in distinguishing network traffic direction on a bridge port. Isoflat uses *ebtables* to implement firewall function as *ebtables* has better support for packet filtering on bridge ports. The tool is mainly used for link layer filtering and its network layer filter is fairly simple. Nevertheless, its network layer filtering support is sufficient for our firewall implementation.

TABLE I: Composition of an Isoflat firewall rule

Property	Type	Description
id	string	Unique ID of the rule
network_id	string	ID of the flat provider network which the firewall applies to
direction	enum	Direction of the traffic to be rejected
remote_network_id	string	ID of the remote flat provider network where traffic comes from/goes to
remote_ip	string	IP address space in CIDR where traffic comes from/goes to
port_range_min	integer	Starting port number or ICMP type
port_range_max	integer	Ending port number or ICMP code
ethertype	string	IPv4 or IPv6
protocol	string	Protocol of the packet to be dropped
description	string	Description of the firewall rule

Table I lists the attributes for Isoflat firewall rules. The `network_id` field refers to the provider network where the Isoflat firewall rule applies. Isoflat rules also have a `remote_network_id` or `remote_ip` to indicate the source/destination of the network traffic to be blocked. The rest of the fields define fine-grained filtering criteria such as protocol type and port numbers. The firewall rules will be translated into *ebtables* chains and rules and deployed on the correct BRG_S.

V. Evaluation

We have evaluated our Isoflat implementation on a three-node OpenStack testbed. The testbed setup and connectivity of the nodes are shown in Figure 8. Two OpenStack Nova compute nodes (i.e., hypervisors) and the controller node have the same hardware configuration. Each node is equipped with a quad-core 3.60GHz CPU, 16GB DDR4 memory, and two 10Gbps network interfaces. These nodes, along with a router, are connected to one 1Gbps switch and one 10Gbps switch, which are used for the management network and provider network respectively. Isoflat has been installed on all the nodes.

We measure aggregate network throughputs, collected at the receiver node's physical network interface, for TCP and

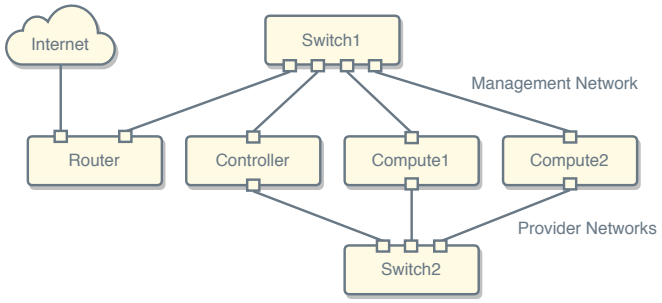

Fig. 8: Isoflat testbed composition and connectivity

UDP, the two most frequently used transport layer protocols [6], as well as Stream Control Transmission Protocol (SCTP) [7], which was proposed to replace TCP. For each experiment on a given transport protocol, we create N pairs of sender and receiver VM instances and connect them to the following three different types of provider network in turn: (i) a single flat provider network, (ii) N VLAN provider networks built upon a single physical network interface, and (iii) N flat provider networks multiplexed through the same physical network interface using Isoflat. Each experiment was repeated 10 times, and the mean and standard deviation values were calculated from 10 runs as the results. All the experiments were conducted using Ubuntu 16.04 and *iperf3* on the 10Gbps links with *iperf3* default parameters.

We first measure network throughputs without traffic filtering. Figure 9 shows the relative wire throughputs of TCP/UDP/SCTP on the physical provider network interface using different types of provider network when $N = 1, 2, 4, 8$. The mean and standard deviation values are represented in percentage of the flat provider network's throughput with each specific protocol. The results indicate that Isoflat is very close to plain flat and VLAN provider networks in terms of TCP throughput performance and that its UDP and SCTP throughputs have slight drops (up to 10% less than the flat provider network). Interestingly, SCTP throughputs exhibit relatively large variations in all three types of provider network compared to TCP and UDP throughputs. Note that *iperf3* is only recommended for TCP performance measurement and large variations of SCTP throughputs may be due in part to *iperf3* implementation and/or its default setting. It is clear from Figure 9 that Isoflat's relative performance compared to the flat network does not degrade with increase of the number of VM pairs (i.e., N).

We then measure the impact of the security group and Isoflat firewall on throughput. Note that *iptables* and *arptables* filtering on the bridge are disabled during Isoflat firewall experiments as the Isoflat firewall uses *ebtables*. We apply the same filtering criterion–"allow only SSH and iperf traffic"–for each transport protocol. Based on the criterion, we create the security group rules or Isoflat firewall rules. The results are shown in Figure 10. For TCP traffic, the Isoflat firewall achieves the same throughput performance as the flat provider network. There is a minor throughput degradation for Isoflat firewall when it deals with UDP and SCTP traffic, but way better than security group. Significant drops in UDP and SCTP throughputs manifest when the security group is employed

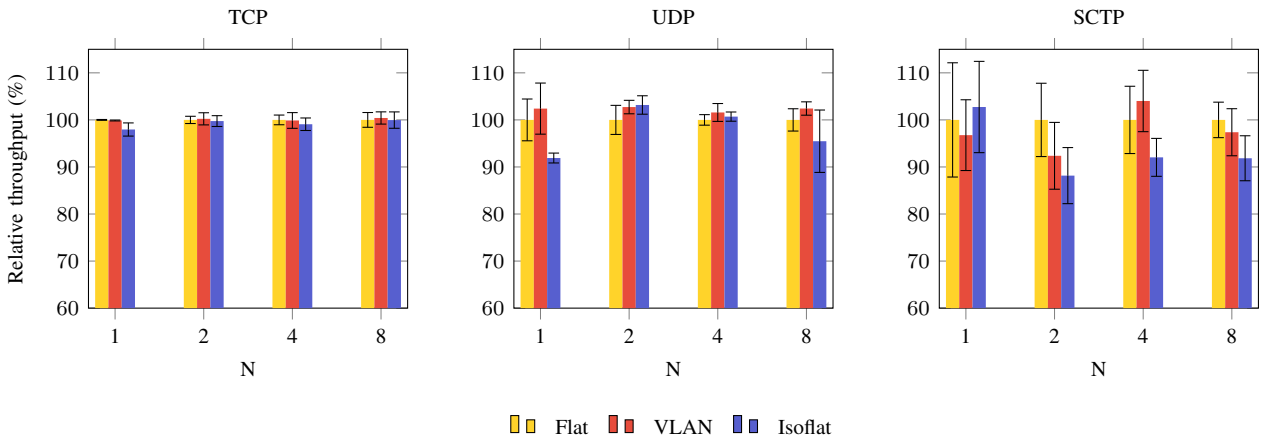

Fig. 9: Throughput performance of flat network, VLAN, and Isoflat

for filtering, with or without Isoflat interface multiplexing. The drastic performance degradation is primarily caused by *iptables* applied to the Linux bridges. Clearly, Isoflat firewall has throughput advantages over security group when dealing with inter-tenant packet filtering.

To find out the influence of VM placement on network performance, we also measure the (relative) TCP throughput between VMs that are allocated on the same hypervisor. The evaluation results are shown in Figure 11. Similar to previous results, Isoflat and its firewall show little overhead compared to flat or VLAN provider networks while the application of security group introduces substantial impacts on throughput performance. This can be explained by the fact that the traffic, which traverses different provider networks, still has to go through the OVS integration bridge, OVS provider bridge, and Isoflat Linux bridge if Isoflat is used, and the traffic is filtered by the security group rules or Isoflat firewall in the same way as it will be in a scenario of different-host VM placement. Since the throughput largely depends on the performance of virtual interfaces and bridges and the packet filtering mechanism, similar data paths will result in similar throughput results or even worse results on the same node due to high CPU loads.

VI. Related work

Network isolation is essential for multi-tenant data center networks. Based on the subject of isolation, network isolation can be further divided into performance isolation and data isolation. Many approaches have been proposed for network performance isolation in the past decade. A typical approach is rate limiting, which is enforced at hypervisors or switches to ensure fair share of bandwidth among participating tenants. Such an approach is adopted by Seawall [8], Gatekeeper [9], Netshare [10], and Pulsaar [11]. Another approach relies on VM placement or network link allocation to achieve bandwidth guarantee, which is used in [12] and [13]. Some network performance isolation frameworks (e.g., [14]) also combine the above two approaches.

In terms of data isolation, Hao *et al.* [15] proposed a network model that combines VLAN and programmable switches for constructing intra-tenant network across different geographical zones while isolating inter-tenant traffic based on customer defined policies. This work can be seen as an early attempt at applying SDN to cloud network isolation. Mudigonda *et al.* [16] achieved multi-tenant network isolation by encapsulating and decapsulating packets and overwriting source (destination) MAC addresses at the egress (ingress) port of the hypervisor, which is managed by dedicated agents. Nunes *et al.* [17] took a similar approach in which packets' MAC addresses are overwritten. They also employed an application in the SDN controller to keep tracking the tenants of all the VMs and issue OpenFlow rules to drop packets traveling between two tenants. There are other approaches such as [18], [19] that rely on the labels attached to packets or specific data and enforce filtering by labels on network edges to realize isolation.

Isoflat focuses on data isolation in flat provider networks where tenants utilize the same physical infrastructure and possibly communicate with each other. Firewalls are more flexible to control data flow in such environments. Security group is such a type of firewall that is widely used in commercial clouds. However, Jin *et al.* [5] pointed out that misconfiguration is common among user-defined security groups and that it poses a great security risk using security group as the only resort for tenant network protection. Isoflat's firewall is very similar to security group in terms of the composition of firewall rules. However, their internal implementation and scope of protection are quite different. Moreover, Isoflat firewalls are only configurable to cloud administrators. There also exist discussions of applying SDN technology for distributed cloud firewalls (e.g., [20] and [21]), which are potentially applicable to Isoflat firewalls as the underlying drivers are interchangeable.

VII. Conclusion

In this paper, we have presented a new mechanism named Isoflat that improves utilization and security of OpenStack flat provider networks. By introducing an additional layer of switching, Isoflat enables OpenStack clouds to create multiple flat provider networks on a single physical interface without significant change to existing networking infrastructure. Furthermore, Isoflat firewalls give users a shortcut to restricting

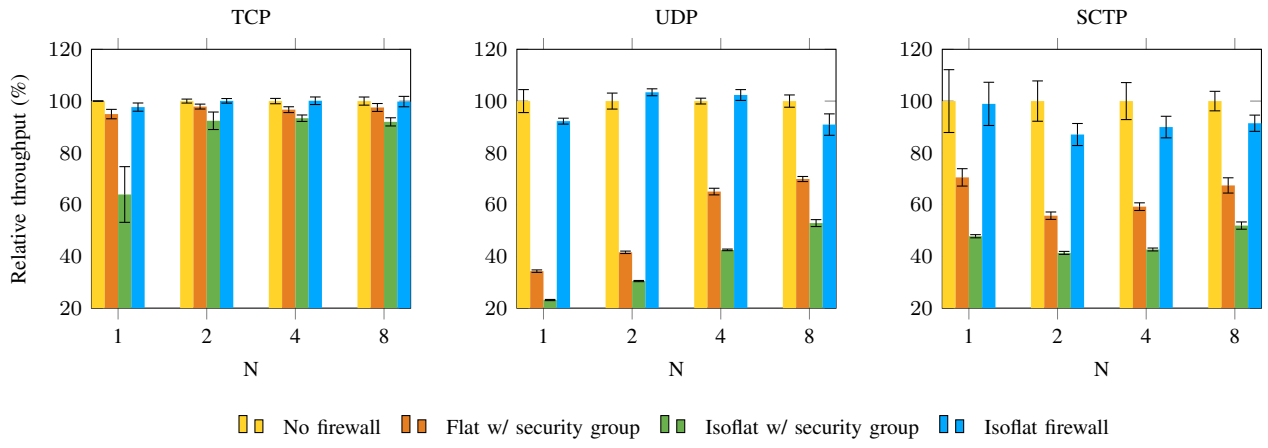

Fig. 10: Throughput performance of Isoflat firewall and security groups

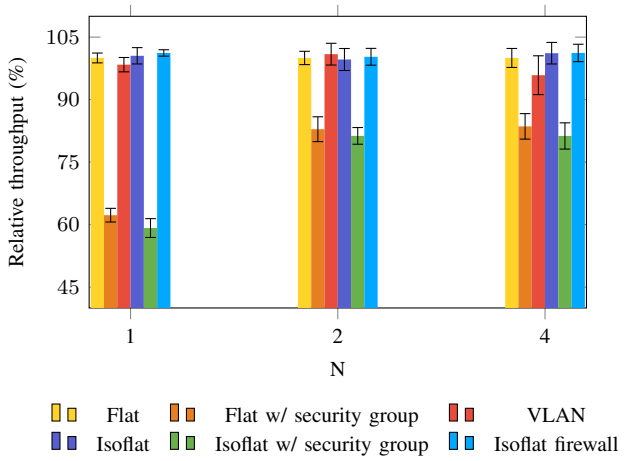

Fig. 11: TCP throughput performance comparison with the same-host VM placement

provider network data flows. Our evaluation shows that Isoflat makes little impact on TCP throughput but has noticeable performance improvements on the provider network throughput compared to the security group. We plan to adapt Isoflat to other OpenStack ML2 mechanism drivers besides Linux bridge and Open vSwitch and to enhance provider network protection with data link layer firewalling.

References

[1] "April 2017 OpenStack User Survey," 2017. [Online]. Available: https://www.openstack.org/assets/survey/April2017SurveyReport.pdf

[2] V. Del Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, "A Survey of Network Isolation Solutions for Multi-Tenant Data Centers," *IEEE Communications Surveys and Tutorials*, vol. 18, no. 4, pp. 2787–2821, 2016.

[3] "Openstack Docs: Linux bridge: Provider networks," 2017. [Online]. Available: https://docs.openstack.org/neutron/latest/admin/deploy-lb-provider.html

[4] "Openstack Docs: Open vSwitch: Provider networks," 2017. [Online]. Available: https://docs.openstack.org/neutron/latest/admin/deploy-ovs-provider.html

[5] C. Jin, A. Srivastava, and Z. L. Zhang, "Understanding security group usage in a public iaas cloud," in *Proc. 35th INFOCOM*, San Francisco, CA, 2016, pp. 1–9.

[6] "Caida data monitors – active and passive data monitors," 2019. [Online]. Available: http://www.caida.org/data/monitors/

[7] R. Stewart, "Stream Control Transmission Protocol," Internet Requests for Comments, RFC Editor, RFC 4960, September 2007. [Online]. Available: http://www.rfc-editor.org/rfc/rfc4960.txt

[8] A. Shieh, S. Kandula, A. Greenberg, and C. Kim, "Seawall: Performance isolation for cloud datacenter networks," in *Proc. 2nd USENIX HotCloud*, 2010, pp. 1–7.

[9] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting Bandwidth Guarantees for Multi-tenant Datacenter Networks," in *Proc. 3rd USENIX Workshop on I/O Virtualization*, 2011, pp. 784–789.

[10] V. T. Lam, S. Radhakrishnan, A. Vahdat, G. Varghese, and R. Pan, "NetShare and stochastic netshare: Predictable bandwidth allocation for data centers," *ACM SIGCOMM CCR*, vol. 42, no. 3, pp. 6–11, 2012.

[11] S. Angel, H. Ballani, T. Karagiannis, G. O'Shea, and E. Thereska, "End-to-end Performance Isolation Through Virtual Datacenters," in *Proc. 11th USENIX OSDI*, Broomfield, CO, USA, 2014, pp. 233–248.

[12] C. Guo, G. Lu, H. J. H. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: a data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th Intl. Conf. on emerging Networking EXperiments and Technologies (CoNEXT)*, 2010, pp. 15:1–15:12.

[13] E. Zahavi, A. Shpiner, O. Rottenstreich, A. Kolodny, and I. Keslassy, "Links as a Service (LaaS): Guaranteed Tenant Isolation in the Shared Cloud," in *2016 ACM/IEEE ANCS*, 2016, pp. 87–98.

[14] H. Ballani, K. Jang, and T. Karagiannis, "Chatty tenants and the cloud network sharing problem," in *Proc. 10th USENIX NSDI*, 2013, pp. 171–184.

[15] F. Hao, T. V. Lakshman, S. Mukherjee, and H. Song, "Secure Cloud Computing with a Virtualized Network Infrastructure," in *Proc. 2nd USENIX HotCloud*, 2010, pp. 1–7.

[16] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary, "NetLord: A Scalable Multi-tenant Network Architecture for Virtualized Datacenters," *ACM SIGCOMM CCR*, vol. 41, no. 4, p. 62, October 2011.

[17] R. V. Nunes, R. L. Pontes, and D. Guedes, "Virtualized network isolation using software defined networks," in *Proc. 38th Annual IEEE Conference on Local Computer Networks*, 2013, pp. 683–686.

[18] Y. Mundada, A. Ramachandran, and N. Feamster, "Silverline: data and network isolation for cloud services," in *Proc. 3rd USENIX HotCloud*, 2011, pp. 1–6.

[19] K. Onoue, N. Matsuoka, and J. Tanaka, "Host-based multi-tenant technology for scalable data center networks," in *Proc. 8th ACM/IEEE ANCS*, 2012, pp. 87–98.

[20] A. Mahesh, A. Chandrasekaran, R. ArunKumar, K. SivaKumar, and N. Vigneshwaran, "Cloud based firewall on OpenFlow SDN network," in *Proc. 2017 Intl. Conf. on Algorithms, Methodology, Models and Applications in Emerging Technologies*, 2017, pp. 1–6.

[21] Y. Chang and T. Lin, "Cloud-clustered firewall with distributed SDN devices," in *Proc. 2018 IEEE WCNC*, vol. 12, no. 4, 2018, pp. 1–5.