Design and Implementation of Virtual TAP for SDN-based OpenStack Networking

Seyeon Jeong*, Jae-Hyoung You† and James Won-Ki Hong*

*Department of Computer Science and Engineering, POSTECH, Korea.
{jsy0906, jwkhong}@postech.ac.kr

†Graduate School of Information Technology, POSTECH, Korea.
styoo@postech.ac.kr

***Abstract*—Currently, virtualization technology that enables a data center to efficiently use server resources is promising as cloud services are being prevalent with increasing traffic volumes and requirements for service quality. vTAP (Virtual Test Access Port), we propose in this work, can overcome the problem that existing hardware TAP devices cannot be used in duplicating inter-VM (Virtual Machine) packets being transmitted over virtual links. vTAP can be implemented using a virtual switch that provides network connectivity to VMs by switching packets over the virtual links. Port mirroring, or SPAN (Switched Port Analyzer), that is available in some software switches can be a naive solution but using it in a system that needs to treat a large volume of network traffic implies performance degradation in packet switching and error-prone manual configuration. In this work, we describe the design and detailed considerations on our OpenFlow-based vTAP implementation. It is based on Open vSwitch and ONOS (Open Network Operating System) SDN (Software-Defined Networking) controller to enable packet-level traffic monitoring in OpenStack environment. We also present the performance evaluation on our vTAP used in some network monitoring cases, with packet processing acceleration by DPDK (Data Plane Development Kit).**

***Index Terms*—SDN, OpenStack, Network Monitoring, Packet Mirroring**

I. Introduction

As software implementation of traditional hardware TAP devices, the major purpose of vTAP (Virtual Test Access Port) is to provide visibility on traffic between VMs (Virtual Machines) especially in server virtualization environment. Usually, a hardware TAP device is deployed on a physical link between a server and a switch or between a switch and a router to duplicate packets and send them to a separate entity such as NPB (Network Packet Broker), where the copied packets are aggregated and then distributed to several network tools [1]. Those tools encompass IDS (Intrusion Detection System), traffic analyzer, threat management system and so on. This type of dedicated TAP devices on each link requires additional CAPEX but it provides fast packet duplication for network monitoring. However, as the trend moves toward cloud data center with server and network virtualization, hardware TAPs are losing their position because it cannot be used to monitor traffic between VMs in a host. Those packets usually pass through virtual links via a virtual switch, which is largely different from traditional physical networks. The absence of TAP functionality may result in difficulty in traffic monitoring and analysis to satisfy service quality and security requirements. So, the necessity of vTAP which can replace traditional TAP devices in virtualization environment comes to the fore.

In discussing the virtualization technologies, OpenStack is one of favorites where large pools of compute, storage, and networking resources throughout a data center can be managed [2]. Hundreds of the world's largest companies rely on OpenStack to run their business everyday. There would be millions of service requests involving massive internal VM-to-VM traffic, so monitoring OpenStack network is closely related with their revenue. In addition, a new type of EDoS (Economic Denial of Sustainability) attacks which induces purveyors to pay their cloud-based service bills beyond their economic means by abusing the elastic scaling features of the target cloud become a large threat besides DDoS attacks [3]. So, traffic (packet) monitoring in OpenStack which work as underlying platform of cloud services is attracting interest.

As a networking service component in OpenStack, Neutron maps the underlying physical networks to instances of provider networks and tenant networks [4] to realize east-west and north-south communication. However, with the evolution of SDN (Software-Defined Networing), there have been some voices that the traditional Neutron-based OpenStack networking is insufficient to virtualize networking resources, compared to other resource types. So, using SDN controllers such as ONOS (Open Network Operating System) and ODL (OpenDaylight) with Neutron together to complement each other in OpenStack networking have become a promising solution. For example, an SDN controller can be actively used to realize network slicing in OpenStack through various SBI (Southbound Interface) for different types of data plane entities.

In the view of telcos who want to reduce CAPEX and OPEX by replacing existing network functions that are vertically integrated in dedicated middleboxes with VNFs (Virtualized Network Functions) dynamically operating in VMs of their data centers [5], OpenStack can be a desirable underlying platform for their NFV (Network Function Virtualization) system. vIMS (Virtual IP Multimedia Subsystem) and vEPC (Virtual Evolved Packet Core) [6] are major use cases of

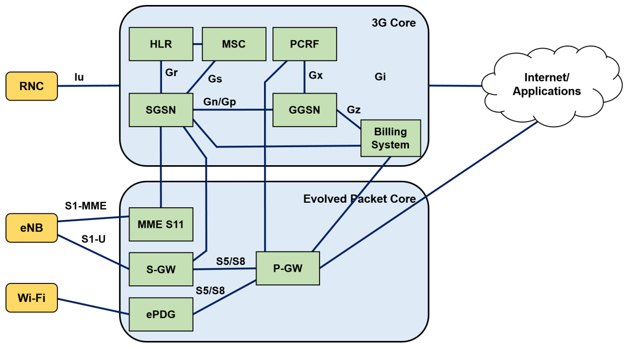


Fig. 1: A traditional EPC compositon in a core network

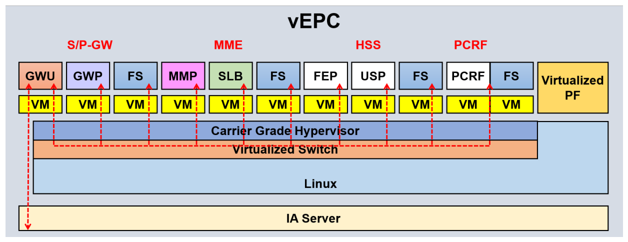


Fig. 2: A virtual EPC (vEPC) composition in a data center network

OpenStack (networking) with SDN/NFV. Coming back to the vTAP concept, we can note that most of traditional EPC systems monitor their networks by deploying TAP devices in each physical link between EPC components (Fig. 1). So, some vEPC implementation based on OpenStack (Fig. 2) can use our vTAP for the inter-VM packet monitoring, because our approach targets OpenStack working with Open vSwitch (OVS) virtual switch and ONOS SDN controller.

In this work, we cover an OpenStack system where the vTAP functionality can be performed to monitor (capture) packets between OpenStack VM instances, regardless of their physical (host) locations, using ONOS which provides some SDN features to Neutron-driven OpenStack networking through the OpenFlow protocol. Our contributions are as follows.

- Propose a novel vTAP method using OpenFlow Group Table. This approach follows the SDN concept where control plane and data plane is separated.
- Provide design details about the system where ONOS and Open vSwitch with DPDK (Data Plane Development Kit) [7] is compatible with existing OpenStack services (e.g. Neutron and Nova) for the vTAP functionality. To do this, we use ONOS SONA (Simplified Overlay Network Architecture) [8] as a base platform.
- Based on the design, we implemented the monitoring system where the proposed vTAP collects inter-VM traffic through the centralized TAP policy management. We also evaluate its performance in some network monitoring use cases.

The remainder of this paper is organized as follows. Section II presents related work and Section III describes design considerations of our proposal. We provide implementation details in Section IV and evaluation of some test cases in Section V. Finally, we conclude our work in Section VI including future work.

II. Related Work

IXIA and Gigamon are two representative vendors providing each commercial vTAP product that is a part of their data center network monitoring solutions. The purpose of these vTAP solutions is to provide visibility on traffic between VMs especially in a server virtualization environment. IXIA vTAP is known to realize the vTAP functionality in somewhere between the OS kernel and the hypervisor. An individual monitor located in the host server aggregates copied packets and offers analysis reports on the in-flight traffic between VMs [9]. On the other hand, Gigamon's approach is known to install per-VM vTAP agents which copy inbound/outbound packets of the VMs and then sends them to a central monitoring system through tunnel networks [10]. Based on the high-performance proprietary implementation (so not easy to find the details), they have proven their usefulness in the industry field. However, in terms of customers, there are difficulties of vendor dependency, maintenance and expandability due to the exclusiveness of the commercial products. So, our vTAP approach tries to focus on providing compatibility and programmability by utilizing solid open (source) projects such as Open vSwitch, DPDK, ONOS and OpenStack.

Overall, research on SDN network monitoring in data center networks has been much more active compared to research on vTAP itself. Planck [11] accelerates traffic collection speed using the port mirroring function of a hardware OpenFlow switch. This work intentionally oversubscribes switch ports in order to gain "sampling" information of certain traffic flows, and analyzes it for network monitoring. EverFlow [12] leverages commodity switch's "match and action" capability to trace specific packets and help administrators troubleshoot network faults. Inspired from those above, NetAlytics [13] reduces overload of real-time monitoring on data center servers by dynamically placing instances of Monitor, Aggregator and Processor in those servers so that it can balance network bandwidth and load. NetAlytics uses OpenFlow flow rules with DPDK for fast packet processing in order to copy traffic flows in SDN-based data center networks. These approaches mainly use OpenFlow or vendor-specific port/flow mirroring as the common foundation method to grasp network states from a set of packets, but they focus on monitoring traffic between host servers through hardware (OpenFlow) switches. On the other hand, our vTAP is more focusing on monitoring traffic between VMs through accelerated virtual switches and flexible management on TAP polices in SDN-integrated OpenStack environment.

NFVPerf [14] can detect bottleneck points of an NFV system through online performance monitoring especially in OpenStack testbeds. Collecting traffic between VMs in the same host or different hosts is realized using the port mirroring feature of Open vSwitch, and collected data is sent to a

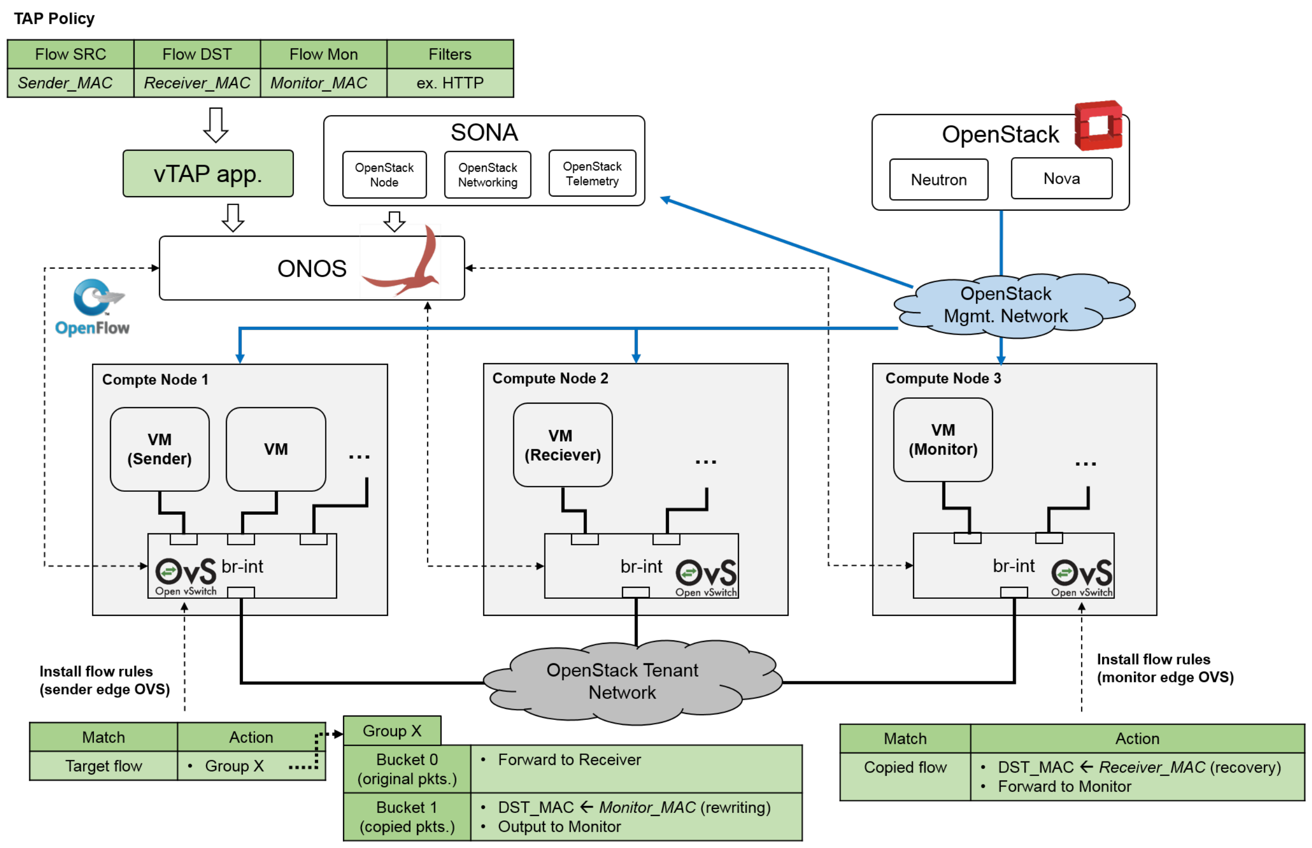


Fig. 3: Overall design architecture of the proposed system

performance analyzer that detects bottleneck indications from throughput and delay measurement. Despite its novel detection algorithm, the use of port mirroring limits the performance of collecting packet data. T-NOVA [15], an open source MANO stack for NFV infrastructures, includes OpenFlow-based network (switch) monitoring as a part of its OpenStack monitoring framework because it is critical to measure network statistics for NFV orchestration. It has limitation on analyzing packet payload for further analysis because it only utilizes OpenFlow port and flow statistics.

III. System Design

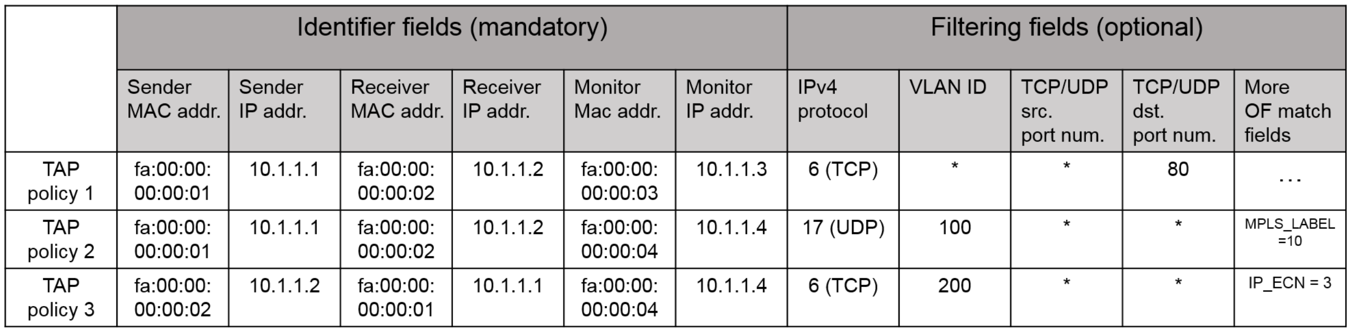
Fig. 3 illustrates the overall architecture of the proposed system. In this section, we provide functional requirements of major components in the system and their integration.

A. *vTAP Application*

As the previous work of this proposal, the vTAP application running on ONOS has two main functions; central management on TAP policies for network administrators (control plane) and flow rule setup for Open vSwitch (OVS) instances to reflect requested TAP polices (data plane) [16]. The vTAP application provides an user interface for TAP policy management (addition, removal and modification). Fig. 4 depicts probable TAP policies specified by network administrators.

The Identifier fields should not be omitted because they define target flows to be duplicated from the specifications of the Sender, Receiver and Monitor MAC or IP address fields. Consequently, Monitor (VM) receives the duplication of the target flow that is being transmitted from Sender (VM) to Receiver (VM). The Filtering fields whose specification is optional restrict the range of target flows depending on their packet header values in each protocol layer. For example, TAP policy 1 in Fig. 4 enables its Monitor to only receive the duplication of the HTTP traffic from 10.1.1.1 to 10.1.1.2, by specifying the IPv4 protocol value as 6 (TCP) and the (TCP) destination port number as 80. Because a TAP policy is applied to the data plane in the form of OpenFlow flow rules, we can extend the Filtering fields depending on the OpenFlow version the target network use in order to offer more granularity on TAP policy specification. OpenFlow 1.5.1 allows 44 match fields from OXM (OpenFlow eXtended Match) support [17]. A TAP policy also should support wildcard matching for target flows by specifying asterisk (*) in the Filtering fields. We assume that network administrators use a global network view offered by the central SDN controller for this management.

Then, the vTAP application converts the high-level policy into specific OpenFlow flow rules to be installed into related switches. To reflect the TAP policy in the actual data plane, we define two types of flow rules which are the rewriting rule and the recovery rule. The rewriting rule is installed into the OVS bridge where Sender (VM) is directly connected to (sender edge OVS). This type of flow rule consists of one



	Identifier fields (mandatory)						Filtering fields (optional)				
	Sender MAC addr.	Sender IP addr.	Receiver MAC addr.	Receiver IP addr.	Monitor Mac addr.	Monitor IP addr.	IPv4 protocol	VLAN ID	TCP/UDP src. port num.	TCP/UDP dst. port num.	More OF match fields
TAP policy 1	fa:00:00:00:00:01	10.1.1.1	fa:00:00:00:00:02	10.1.1.2	fa:00:00:00:00:03	10.1.1.3	6 (TCP)	*	*	80	. . .
TAP policy 2	fa:00:00:00:00:01	10.1.1.1	fa:00:00:00:00:02	10.1.1.2	fa:00:00:00:00:04	10.1.1.4	17 (UDP)	100	*	*	MPLS_LABEL =10
TAP policy 3	fa:00:00:00:00:02	10.1.1.2	fa:00:00:00:00:01	10.1.1.1	fa:00:00:00:00:04	10.1.1.4	6 (TCP)	200	*	*	IP_ECN = 3

Fig. 4: An examples of TAP policy specifications in the vTAP application

match-action entry and one group table with two buckets. The match-action entry in the flow table of the sender edge OVS can match the target flows of the corresponding TAP policy and delegate their treatment to the following group table. The first bucket of the group table is used to forward the original packets to their destination (Receiver), but the second bucket is used to duplicate the original packets and forward the duplications to Monitor. This packet copy process based on OpenFlow group table features is useful for not only its simple implementation without any modification to OVS but also flexibility in handling the duplicated packets. Actually, packet duplication itself can be simply realized by OpenFlow "multicast" action within a single match-action entry, but structurally it cannot apply different actions on the duplicated packets, which means they cannot be reached to Monitor. On the other hand, the group table (bucket) approach is able to change the destination MAC or IP address of the duplications to that of Monitor's (rewriting) and apply additional treatments (e.g. VLAN tagging). This approach needs only one flow rule in a sender edge switch to reflect a TAP policy.

The final job to revert the MAC or IP address modified by the rewriting rule is performed by the recovery rule for consistency of the packet contents. The recovery rule is installed into the flow table of the OVS bridge where Monitor (VM) is directly connected to. The vTAP application can easily provide this stateful information of the original MAC or IP address to be recovered in the monitor edge switch. The overall process of the vTAP application is illustrated in Fig. 5.

This vTAP approach improves central management on TAP polices in control plane using a global network view, and provides granularity in handling packet flows (both original and duplication) in data plane using OpenFlow. However, compared to hardware TAP, one critical weakness of our vTAP based on virtual (software) switch is performance degradation on production traffic because OVS should handle both general packet forwarding and packet duplication at the same time using the restricted computing power of a server. The performance problem become even worse if the host server start to increase the number of VM instances or to process a CPU-intensive jobs, because OVS should compete host resources then. So, we decided to run OVS instances with DPDK to accelerate packet processing performance by

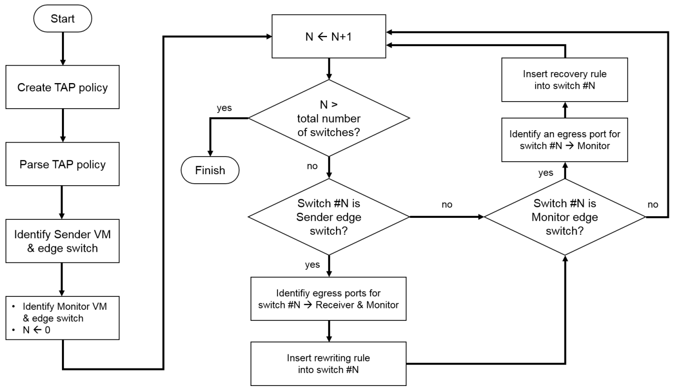

Start
Create TAP policy
Parse TAP policy
Identify Sender VM & edge switch
• Identify Monitor VM & edge switch
• N ← 0
N ← N+1
N > total number of switches?
yes
Finish
no
Switch #N is Sender edge switch?
yes
Identifiy egress ports for switch #N → Receiver & Monitor
Insert rewriting rule into switch #N
no
Switch #N is Monitor edge switch?
no
yes
Identify an egress port for switch #N → Monitor
Insert recovery rule into switch #N

Fig. 5: Operation flow chart in the vTAP application

allocating dedicated CPU cores and memory on them. Related implementation details are described in Section IV.

B. OpenStack and SONA

To integrate OpenStack with ONOS so that the roles of Neutron for OpenStack networking are distributed to the SDN controller, there have been several initiatives [8] [18] [19]. Although Neutron supports somewhat SDN-based L2/L3 networking for OpenStack by configuring Open vSwitch or Linux bridge agents on each OpenStack node [20], we need a dedicated SDN controller to fully utilize SDN features of the target network especially in traffic engineering and QoS management. The need becomes more important in developing an NFV system with SFC (Service Function Chaining) and auto-scaling functions. Another application of the OpenStack-ONOS integration can be our vTAP application running on ONOS for east-west traffic (from tenant networking) and north-south traffic (from external networking) monitoring in OpenStack.

We choose to use SONA as the fundamental integration bridge because SONA has been mature as an initial ONOS killer application and is being actively developed by ONOS contributors. SONA is composed of a set of ONOS applications, OpenstackNode and OpenstackSwitching, and now it extends to OpenstackTelemetry.

In Fig. 3, OpenStackNode abstracts underlying OpenStack nodes (Controller, Compute, Gateway) in terms of their internal networking attributes such as network interface types, OVS bridges and IP addresses along with a few of metadata such as hostname and SONA-specific host states. OpenStackNode stores those modeling instances in the ONOS shared store system for management on the networking side of each node and information access by OpenStackNetworking.

OpenStackNetworking has two major responsibilities of OpenStack switching and OpenStack routing. OpenStack switching mainly configures L2 forwarding rules in each OVS bridge (br-int) for OpenStack VM instances which belong to the same tenant to communicate with each other through overlay networks. Currently, SONA supports network types of FLAT (no tenant isolation), VLAN (L2 isolation) and VXLAN (L2 over L3) to realize tenant networking across different OpenStack setups. Whereas OpenStack routing configures flow rules, in each OVS bridge of Compute and Gateway nodes, for any OpenStack VM instances retaining floating IPs to communicate with OpenStack external. Besides, ARP and DHCP proxies can work based on related flow rules defined by OpenStackNetworking.

While SONA is handling OpenFlow-based OpenStack networking, existing Neutron components in Controller nodes still need to be active to respond requests or events from other OpenStack components (e.g. Nova). After SONA is deployed, Neutron just calls REST APIs of SONA to handle networking requests, instead of using OVS, DHCP, L3 and other agents/plugins/mechanisms by itself. Then, SONA performs required operations in a way that using OpenFlow flow rules in each "br-int" OVS bridge of OpenStack nodes and their abstractions stored in the ONOS. vTAP policies mentioned in Section III-A are also managed in the form of OpenFlow flow rules inside OVS and several metadata in ONOS. At this end, network administrators can focus on using the master ONOS controller where both the vTAP and the SONA application are running on.

IV. Implementation

To implement the proposed system in Section III, we constructed the OpenStack testbed with several basic OpenStack service components (e.g. Nova, Neutron, Keystone) in our lab server. We installed OpenStack Pike release through the corresponding version of devstack [21] to simplify the installation process and to have a deployment profile of OpenStack for further use of our PoC in different environments.

A. Integration of OpenStack, SONA and OVS-DPDK

Integrating OpenStack with ONOS can be done by networking-onos [22] which is an OpenStack sub-project to help the integration process between Neutron and ONOS. To make it work, the Neutron configuration should be changed to use networking-onos as a plugin for the OpenStack L2/L3 netwkroing, instead of the default L2 mechanism drivers (e.g. Open vSwitch or Linux bridge) and L3 agents. In summary, when Neutron receives events or service requests for L2/L3 networking services, it relays them to SONA (NBI) through networking-onos. SONA can be activated as a set of related ONOS applications in an ONOS node where the ONOS instance is deployed and reachable to all Compute and Gateway nodes through an OpenStack management network.

After the deployment of networking-onos, which means ONOS become a centralized control plane for the target OpenStack, SONA takes a network configuration file (network-cfg.json) that specifies details of the target OpenStack nodes with their management/data network IPs, network interface names, and datapath IDs of OVS bridges (br-int) where flow rules are being installed. To synchronize SONA (ONOS) with current OpenStack configuration which may be requested by the outside of SONA (e.g. OpenStack dashboard and CLI), developers can use several SONA CLIs so that SONA fetches the current settings from OpenStack services (DBs) and then stores them in the ONOS store.

As we mentioned in Section III-A, we chose to use Open vSwitch with DPDK, or OVS running in DPDK mode (OVS-DPDK), as data plane switches. So, we built Open vSwitch 2.9.2 with DPDK 17.11.2 in every Compute node to run the OVS "br-int" bridges in DPDK mode (user space). An OVS-DPDK bridge requires the datapath type of netdev and one of two types of DPDK-backed vHost User Ports (dpdkvhostuser and dpdkvhostuserclient) [23] to establish network connection with a VM through an user-space virtual network interface. However, in general OpenStack-SONA environment, a newly created OpenStack VM instance is connected to the host's "br-int" OVS bridge using the TAP [24] type kernel-space virtual network interface by default. As of the commit version of SONA we used, OpenStack fails to create a VM instance with a DPDK-backed virtual interface because SONA is not yet data-plane-agnostic, which means SONA assumes only the use of TAP type connection regardless of the actual use of DPDK vHost User Ports. So, in this work, we tried to construct an OpenStack testbed where its networking is managed by ONOS SONA in control plane and realized by OVS-DPDK in data plane to verify the feasibility of TAP policy management through our vTAP application.

To integrate all of them above, first we need to make OpenStack-SONA allow the use of data plane variant in spawning a new VM instance. Because our primary goal was constructing a testbed for verification, not developing a general solution such as SONA transparent to data plane, we made patches for the default source codes of networking-onos, OpenStack Nova and SONA. After the application of these patches, OpenStack-SONA assumes the use of OVS-DPDK as a default data plane switch. The major changes are as follows.

1) networking-onos: Originally, it forces SONA to use the TAP type network interface for OpenStack VM instances when the ONOS driver is used for L2 networking. Because the Neutron library in the Pike release already have defined the general vHost-user type interface supported by QEMU, we changed related codes of networking-onos to use the vHost-user type interface by default.

2) OpenStack Nova: Creating a new OpenStack VM instance is followed by creation of the corresponding Neutron Port for the VM. A Neutron Port request contains details of the required virtual interface including the interface type and the path name of the socket file which is created by OVS for the new connection. This request enables for Nova to configure the VM's profile for QEMU to deploy the required VM instance on the target host server. So, in this process, we made code changes for the use of the vHost-user type interface instead of the default OVS type (TAP) and specific socket path names to enable server-client model connection between OVS-DPDK (client) and VM (server).

3) SONA: SONA is a collection of ONOS applications such as OpenstackNetworking, OpenstackNode and OpenstackTelemetry. Creating a new OpenStack VM with a DPDK-backed network port requires modification on some L2 switching parts of OpenstackNetworking (files with prefix of OpenstackSwitching). In OpenStack-SONA, each OpenStack node (Controller, Compute, Gateway) has own states specific for SONA (INIT, DEVICE_CREATED, COMPLETE, INCOMPLETE). For any Compute nodes in the COMPLETE state, OpenstackSwitchingHostProvider scans hosts (VMs) which are connected to the "br-int" OVS bridge using the TAP type interface. Because the default code can detect a host from the existence of the corresponding OVS port with the specific name prefix ("tap") allocated only to the TAP type interface, we modified it to detect the (DPDK) vhost-user type interface with the different name prefix ("vhost"). Then, we manually created OVS ports named with the new prefix in order for them to be detected by OpenstackSwitchingHostProvider. According to the ONOS architectural model [25], hosts found events in OpesStackSwitchingHostProvider are handled by OpenstackNetworkManager where overall operations for OpenStack networking are implemented with ONOS core APIs. OpenstackNetworkManager also treats TAP type interfaces only, we needed to change it to embrace the (DPDK) vhost-user type in order to successful creation of Neutron Ports without related OpenStack errors.

The overall process aforementioned is described in Fig. 6.

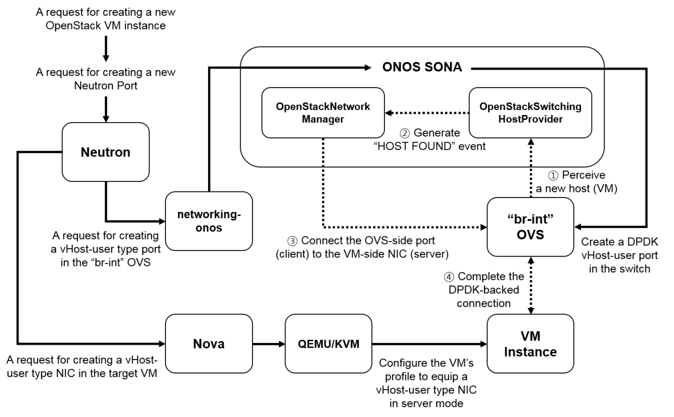

A request for creating a new OpenStack VM instance
A request for creating a new Neutron Port
Neutron
ONOS SONA
OpenStackNetwork Manager
OpenStackSwitching HostProvider
② Generate "HOST FOUND" event
① Perceive a new host (VM)
networking-onos
A request for creating a vHost-user type port in the "br-int" OVS
③ Connect the OVS-side port (client) to the VM-side NIC (server)
"br-int" OVS
Create a DPDK vHost-user port in the switch
④ Complete the DPDK-backed connection
Nova
QEMU/KVM
VM Instance
A request for creating a vHost-user type NIC in the target VM
Configure the VM's profile to equip a vHost-user type NIC in server mode

Fig. 6: Internal operations in the system during the creation of a new OpenStack VM

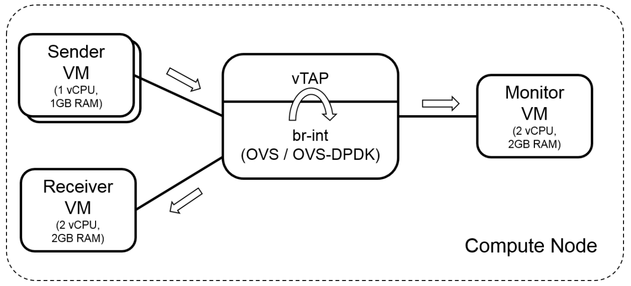

Sender VM (1 vCPU, 1GB RAM)
Receiver VM (2 vCPU, 2GB RAM)
vTAP
br-int (OVS / OVS-DPDK)
Monitor VM (2 vCPU, 2GB RAM)
Compute Node

Fig. 7: Base OpenStack Testbed; OVS mode and VM resources change according to test scenario

B. vTAP Application

In the proposed system of this work, TAP policy management and its provisioning on data plane through OpenFlow are implemented in the ONOS vTAP application. After getting the information of Identifier fields and Optional fields specified in vTAP GUI from network administrators (Fig. 4), it creates a rewriting rule and a recovery rule for the TAP policy. As we mentioned in Section III-A, a rewriting rule which is installed in a Sender edge switch (one of "br-int"s) has a Group Table instance of type "ALL", which performs each flow treatment in all buckets belong to the group, with two buckets. We relate each Group Table instance with own key of the order that the TAP policy is added for the purpose of management (e.g. policy modification or removal).

The second bucket which handles copied packets forwards them to Monitor by changing their destination MAC or IP addresses to that of Monitor and using the same egress port for the next hop delivery. We can easily extract an egress port toward Monitor from a routing path abstraction defined by the ONOS Topology service. Next, a recovery rule is installed in a Monitor edge switch (one of "br-int"s). This rule allows the Monitor edge switch to receive the copied packets originated from the corresponding Sender edge switch, and to recover each of them back to the original one. Thanks to the centralized control plane, ONOS SDN controller, this type of stateful flow handling is easy to realize.

V. Evaluation

In this section, we evaluate the performance of the proposed vTAP, according to the use of OVS-DPDK or not in the simple OpenStack testbed (Fig. 7). The first test scenario applies a TAP policy on flows from Sender to Receiver in order for Monitor to receive copied packets. The second test scenario also applies a TAP policy on a bunch of attack packets from Sender to Receiver, which are generated by pytbull [26] IDS/IPS testing tool and replayed on each trial through Tcpreplay [27], in order for Monitor running suricata [28] IDS to receive copied packets and analyze them.

We used one bare metal server with the hardware specification of 2.67 GHz Intel Xeon CPU (12 cores) and 24 GB RAM to construct the OpenStack testbed composing of a each single Controller/Compute/Gateway node (VM), which is the minimal requirement for SONA-based multi-node setup. It is note that because each OpenStack node is VM, we applied nested virtualization [29] to allow Compute node to use KVM acceleration in spawning OpenStack VM instances. Summary of platforms and tools we used in the following evaluations is in Table I.

TABLE I: Summary of platforms and tools used

Name	Version	Purpose
OpenStack	Pike	Base Virtual Infrastructure Manager (VIM)
ONOS	1.14.0	SDN controller for OpenStack networking in combination with SONA embedded
OpenFlow	1.4	Data and control plane comm. protocol
Open vSwitch	2.9.2	Base virtual switch
DPDK	17.11.2	Packet processing acceleration for OVS
pktgen	4.4.0 (Kernel)	IPv4 packet generation
Iperf	2.0.12	Background traffic generation
pytbull	2.1	Network attack traffic generation
Tcpreplay	4.2.5	Replay of packet traces on the testbed
Suricata	4.0.5	Network intrusion detection system

A. RX Throughput Measurement on Original and Duplicated Packets

In this test scenario, we applied the TAP policy that requires the “br-int” OVS switch to copy all Ethernet frames from Sender to Receiver. Then, we generated IPv4 packets in Sender so that the switch sent the original packet flow to Receiver and the duplicated packet flow to Monitor. In our previous work [16], we performed these trials with different packet sizes on both the OVS switch in normal (kernel) mode and the OVS switch in DPDK (user) mode. In the evaluation, OVS-DPDK outperformed normal OVS, by showing 8∼25 times improvement on RX throughput measurement at both Receiver (original pakcet flows) and Monitor (duplicated packet flows) according to the packet size.

Additionally, we performed RX throughput measurement when OVS-DPDK uses a single PMD or two PMDs (Fig. 8). PMD threads are the threads that do the heavy lifting for the DPDK datapath and perform tasks such as continuous polling of input ports for packets, classifying packets once received, and executing actions on the packets once they are classified [30]. So, if there are multiple DPDK-backed ports producing traffic, performance can be improved by creating multiple PMD threads running on separate cores. Allocating one additional PMD thread to OVS-DPDK needs one more CPU core which will be dedicated for those functions.

In this test, we made two Sender VMs transmit each of their packet flows to the single Receiver VM, while the OVS-DPDK switch duplicated them using the corresponding TAP policy in order to send them to the single Montior VM. Then we affected the performance of the OVS-DPDK switch by generating loopback traffic inside the Compute node. This background traffic enabled us to more clearly distinguish the performance gap between a single PMD and double PMDs, since this placed major performance bottleneck on the switch, not the destination VMs. We used pktgen [31] at Sender to

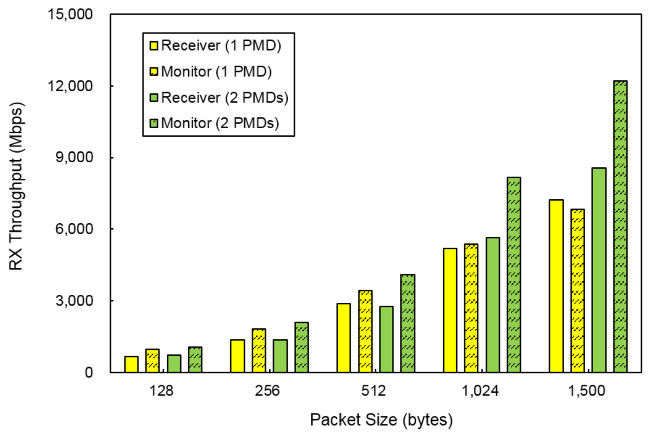


Fig. 8: Measurement on RX throughput with different packet sizes in case of single PMD and two PMDs in OVS-DPDK

generate and send different sizes of IPv4 packets to Receiver and Monitor.

In Fig. 8, we observed that using one additional PMD threads took effect on the performance in the case of relatively large size of packets. In the two PMD case, there was around 16% improvement on RX throughput measurement of 256-byte packets but around 52% and 79% improvement on 1024-byte packets and 1500-byte (Ethernet v2 MTU) packets respectively. Under the same RX rate during each trial, processing smaller packets involves more CPU interrupts which result in more packet drops with drastic CPU saturation. We also observed that Monitor showed better results than Receiver. This is because original packet flows are more aggressively transmitted, resulting in larger packet drops at its destination (Receiver), while copied packet flows have some processing delays in the switch due to the additional duplication. This phenomenon can be resolved not only by allocating more resources to the destinations but also imposing the performance bottleneck on the switch with a proper amount of background (loopback) traffic, which is our future work.

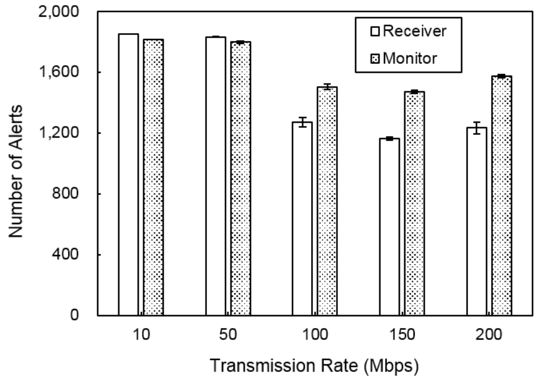
B. Suricata NIDS Reports

In this scenario with the same testbed (Fig. 7), we considered Sender as a malicious attacker, Receiver as a server, and Monitor as an NIDS (Network Intrusion Detection System) which receives copied packets from a switch and analyzes them to report anomalies. We used pytbull to generate a sequence of various attack packets (Brute force, DoS, Shellcode, etc.), and captured them using tcpdump to create a packet trace at Sender. We enlarged the trace with sending loop and bandwidth adjustment features of Tcpreplay in order to overload the OVS-DPDK switch (single PMD thread), Receiver or Monitor VMs. The switch where the TAP policy, “copy all Ethernet frames from Sender to Receiver”, was applied by ONOS sent the replayed attack packets to Receiver, and also sent duplications of them to Monitor at the same time. Then Suricata instances, running on both the Receiver and Monitor VMs, generated reports including attack details and RX statistics such as packet drop rate. Those reports at Receiver can be used as the ground truth at the end of each trial, to compare with Monitor's reports generated by copied packets based on our vTAP implementation.

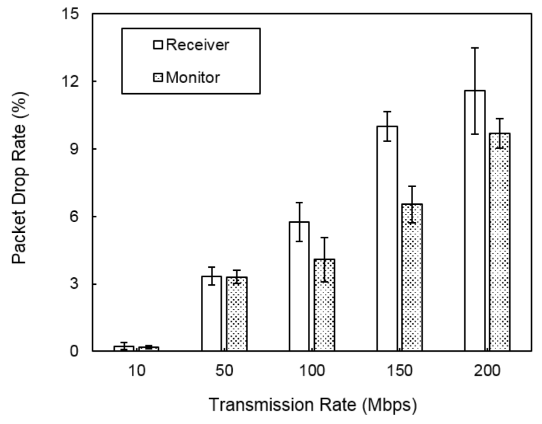
Figure 9a and Figure 9b show the mean number of alerts and the mean packet drop rate respectively. Each trial with a different transmission rate (by Tcpreplay) was repeated 10 times. Note that Receiver and Monitor VMs were with two vCPUs for performance improvement from multi-threaded Suricata, but Sender was with only one vCPU because Tcpreplay is single-treaded so cannot benefit from multi-processors. All the vCPUs were pinned to dedicated physical cores on the host machine.

Figure 9 shows Suricata reports on the mean number of alerts and the mean packet drop rate at different transmission rates. Tcpreplay can replay packet transmission at a highest speed the target testbed can afford which was around 210 Mbps in our case. We can observe few things that lower transmission rates involve lower packet drop rates with more numbers of alerts obviously, and that significant reduces in the number of alerts start at the 100 Mbps rate with around 5% of packet drop rate. We also noted that Monitor Suricata generated more alerts with lower drop rates in the 100, 150 and 200 Mbps cases. This is because original packet flows are more aggressively transmitted, resulting in larger packet drops at its destination (Receiver), while copied packet flows have some processing delays in the switch due to the additional duplication.

However, we cannot assert that larger drop rates linearly reduce the number of alerts, because drops of certain packets that account for small amount and determine attack alerts could not be reproducible on each trial due to time-dynamics of the testbed (cache, packet sequence, etc.). Note that packet drops would be slightly reduced from the use of more RAMs on VM sides, but not from the use of multiple PMD threads on the OVS-DPDK switch side because now the performance bottleneck is single thread implementation of Tcpreplay.



(a) Measurements on the mean number of alerts with different transmission rates



(b) Measurements on the mean packet drop rate with different transmission rates

Fig. 9: Suricata NIDS Reports on Attack Packet Traces

VI. Conclusion

In this work, we have presented the design and implementation details of vTAP applicable to SDN-based OpenStack networking. Particularly, several open (source) projects such as ONOS, SONA, Open vSwitch and DPDK, and OpenFlow protocol are integrated with OpenStack to enable the proposed vTAP system to provide users with the centralized TAP policy management and improve performance in monitoring inter-VM traffic at a packet level.

The major component of the vTAP control plane is the SDN controller (ONOS) application that enables users to define a TAP policy consisting of identifier (source, destination and monitor VMs) and optional specifications of various header match fields to filter out interested traffic flows. Then the TAP definition is transformed into OpenFlow flow rules being installed in edge switches (mainly virtual switches) of an OpenStack environment, in order to reflect required data plane functions such as forwarding and reproducing of packets.

In Section V, we performed RX throughput measurement on original and duplicated packet flows when we use OVS-DPDK as a virtual switch. Using two PMD threads with the cost of one additional CPU core results in around 80% improvement in RX throughput of the duplicated packet flow. We also tested applicability of our vTAP in providing packets to be analyzed by an IDS such as Suricata.

As future work, we plan to evaluate the scalability of our vTAP in more realistic OpenStack environment with a large amount and various types of traffic. It includes mounting abundant physical resources, and finding use cases of our vTAP such as fast packet monitoring for NFV orchestration or intelligent packet collection (sampling) according to a state of a network.

Acknowledgment

This work was supported by the ICT R&D program of MSIT/IITP. [2018-0-00749, Development of virtual network management technology based on artificial intelligence]

References

[1] Spectrami, "Network Packet Brokers." [Online]. Available: https://www.network-visibility.com/cm/products/network-packet-broker

[2] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Openstack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[3] M. N. Kumar, P. Sujatha, V. Kalva, R. Nagori, A. K. Katukojwala, and M. Kumar, "Mitigating economic denial of sustainability (edos) in cloud computing using in-cloud scrubber service," in *Computational Intelligence and Communication Networks (CICN), 2012 Fourth International Conference on*. IEEE, 2012, pp. 535–539.

[4] A. Swanson, "Tenant networks vs. provider networks in the private cloud context," 2016. [Online]. Available: http://superuser.openstack.org/articles/tenant-networks-vs-provider-networks-in-the-private-cloud-context/

[5] O. Hohlfeld, T. Zinner, T. Benson, and D. Hausheer, "Special issue on software-defined networking and network functions virtualization for flexible network management," *International Journal of Network Management*, vol. 26, no. 1, pp. 4–5, 2016.

[6] O. Narmanlioglu and E. Zeydan, "New era in shared cellular networks: Moving into open and virtualized platform," *International Journal of Network Management*, vol. 27, no. 6, 2017.

[7] "Data plane development kit." [Online]. Available: http://dpdk.org/

[8] J. Li, "SONA: DC Network Virtualization," 2018. [Online]. Available: https://wiki.onosproject.org/display/ONOS/SONA%3A+DC+Network+Virtualization

[9] IXIA, "Ixia phantom vtap with tapflow filtering," Tech. Rep., 2016. [Online]. Available: https://www.ixiacom.com/sites/default/files/2016-07/V-DS-Phantom-vTap.pdf

[10] Gigamon, "Gigavue-vm data sheet," Tech. Rep., 2016. [Online]. Available: https://www.gigamon.com/content/dam/resource-library/korean/data-sheet/ds-gigavue-vm-virtual-machine-kr.pdf

[11] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," in *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4. ACM, 2014, pp. 407–418.

[12] Y. Zhu, N. Kang, J. Cao, A. Greenberg, G. Lu, R. Mahajan, D. Maltz, L. Yuan, M. Zhang, B. Y. Zhao *et al.*, "Packet-level telemetry in large datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 479–491.

[13] G. Liu, M. Trotter, Y. Ren, and T. Wood, "Netalytics: Cloud-scale application performance monitoring with sdn and nfv," in *Proceedings of the 17th International Middleware Conference*. ACM, 2016, p. 8.

[14] P. Naik, D. K. Shaw, and M. Vutukuru, "Nfvperf: Online performance monitoring and bottleneck detection for nfv," in *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*. IEEE, 2016, pp. 154–160.

[15] M.-A. Kourtis, M. J. McGrath, G. Gardikis, G. Xilouris, V. Riccobene, P. Papadimitriou, E. Trouva, F. Liberati, M. Trubian, J. Batallé *et al.*, "T-nova: An open-source mano stack for nfv infrastructures," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 586–602, 2017.

[16] S. Jeong, D. Lee, J. Li, and J. W.-K. Hong, "OpenFlow-based virtual TAP using open vSwitch and DPDK," in *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2018.

[17] Open Networking Foundtation, "OpenFlow Switch Specification Version 1.5.1," 2015. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf

[18] B. Sullivan, "ONOS Framework Home," 2016. [Online]. Available: https://wiki.opnfv.org/display/onosfw/ONOS+Framework+Home

[19] H. Moon, "CORD VTN," 2016. [Online]. Available: https://wiki.onosproject.org/display/ONOS/CORD+VTN

[20] SDxCentral, "What is OpenStack Neutron?" [Online]. Available: https://www.sdxcentral.com/cloud/open-source/definitions/what-is-openstack-quantum-neutron/

[21] OpenStack, "DevStack." [Online]. Available: https://docs.openstack.org/devstack/latest/

[22] ——, "networking-onos documentation." [Online]. Available: https://docs.openstack.org/devstack/latest/

[23] Open vSwitch, "DPDK vHost User Ports," 2018. [Online]. Available: http://docs.openvswitch.org/en/latest/topics/dpdk/vhost-user/

[24] Wikipedia, "TUN/TAP." [Online]. Available: https://en.wikipedia.org/wiki/TUN/TAP

[25] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "ONOS: towards an open, distributed SDN OS," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 1–6.

[26] pytbull, "pytbull - IDS/IPS Testing Framework." [Online]. Available: http://pytbull.sourceforge.net/

[27] Tcpreplay, "Tcpreplay - Pcap editing and replaying utilities." [Online]. Available: https://tcpreplay.appneta.com/

[28] Suricata, "Suricata - Open Source IDS / IPS / NSM engine." [Online]. Available: https://suricata-ids.org/

[29] OpenStack, "Configure DevStack with KVM-based Nested Virtualization," 2018. [Online]. Available: https://docs.openstack.org/devstack/latest/guides/devstack-with-nested-kvm.html

[30] Open vSwitch, "PMD Threads." [Online]. Available: http://docs.openvswitch.org/en/latest/topics/dpdk/pmd/

[31] R. Olsson, "Pktgen the linux packet generator," in *Proceedings of the Linux Symposium, Ottawa, Canada*, vol. 2, 2005, pp. 11–24.