Master's Thesis

Design and Implementation of Virtual TAP for Software-Defined Networks

Seyeon Jeong (정 세 연)

Department of Computer Science and Engineering

Pohang University of Science and Technology

2018

소프트웨어 정의 네트워크를 위한 Virtual TAP 설계 및 구현

Design and Implementation of Virtual TAP for Software-Defined Networks

Design and Implementation of Virtual TAP for Software-Defined Networks

by

Seyeon Jeong
Department of Computer Science and Engineering
Pohang University of Science and Technology

A thesis submitted to the faculty of the Pohang University of Science and Technology in partial fulfillment of the requirements for the degree of Master of Science in the Computer Science and Engineering

Pohang, Korea
12. 14. 2017
Approved by
James Won-Ki Hong (Signature)
Academic advisor

Design and Implementation of Virtual TAP for Software-Defined Networks

Seyeon Jeong

The undersigned have examined this thesis and hereby certify that it is worthy of acceptance for a master's degree from POSTECH

12. 14. 2017

Committee Chair	James Won-Ki Hong	(Seal)
Member	Young-Joo Suh	(Seal)
Member	Hanjun Kim	(Seal)

MCSE 정 세 연. Seyeon Jeong
20152304 Design and Implementation of Virtual TAP for Software-Defined Networks,
소프트웨어 정의 네트워크를 위한 Virtual TAP 설계 및 구현
Department of Computer Science and Engineering , 2018, 40p, Advisor : James Won-Ki Hong. Text in English.

ABSTRACT

Currently, server (host) virtualization technology that brings effective use of server resources to a datacenter is promising as cloud services are being prevalent with increasing traffic volumes and requirements for higher service quality. The proposed network TAP, named vTAP (Virtual Test Access Point), overcomes the problem that existing hardware TAP devices cannot be utilized for virtual network links in a virtualized server, and enables to monitor traffic among virtual machines (VMs) at a packet level. vTAP can be implemented by a virtual switch that gives network connectivity to VMs by switching packets over virtual network links. The port mirroring feature of a virtual switch can be a naive solution to provide packet level monitoring among VMs. However, using the feature in an environment that needs to treat large volume of network traffic with low delay such as NFV (Network Function Virtualization) incurs performance degradation of packet switching capability of the switch and error-prone manual configuration. We provide design and implementation approaches to vTAP using Open vSwitch with DPDK (Data Plane Development Kit) and an OpenFlow SDN (Software-Defined Networking) controller to overcome the problems. DPDK can accelerate overall packet processing operations needed in vTAP, and OpenFlow controller can provide a centralized and flexible way to apply and manage TAP policies in an SDN network. We also provide several performance comparisons of vTAP and the naive method.

Contents

List of Tables

List of Figures

I. Introduction

Over the past decade, cloud computing has been widely used to support a variety of applications over the Internet. The key to the increased use of cloud computing has been the server virtualization technology that allows efficient use of computing resources in data centers as well as flexibility and agility to application deployment and management [1]. Major components of server virtualization are a hypervisor, virtual switch and virtual machines (VMs). Generally, current datacenters deploy them to each (commodity) server, and provide various business services that each of them is composed of several related applications that run across the VMs. In this environment, a lot of network traffic is generated simultaneously among VMs in the same host or VMs in different hosts to handle a service request quickly and efficiently. So, a proportion of this VM-to-VM (or East-West) traffic in datacenter traffic volumes is increasing rapidly.

As a software version of existing TAP devices, vTAP (Virtual Test Access Point) is to provide traffic visibility among VMs in a server virtualization environment. An existing TAP device is installed on a physical link between a server and a switch or between a switch and a router to duplicate packet signals and send them to a separate monitoring entity that aggregates packet copies from those TAP devices and monitors network states. A dedicated TAP device on each link requires additional cost but provides fast packet duplication capability. However, as moving toward to datacenter networks with server virtualization, hardware

TAPs cannot be used to monitor traffic among VMs in the same host because the traffic passes through virtual links via a virtual switch. The absence of TAP functionality means being difficult to monitor traffic at a packet level including packet analysis to satisfy QoS (Quality of Service) and security requirements. So, we have designed vTAP and implemented it based on an accelerated virtual switch to solve the problems.

With the evolution of SDN (Software-Defined Networking) [2] and NFV (Network Function Virtualization) [3] technology, telcos and cloud service providers are trying to reduce CAPEX and OPEX by replacing existing network functions that are vertically implemented in dedicated middleboxes with VNFs (Virtualized Network Functions) dynamically operating in VMs of their datacenter [4]. vIMS (Virtual IP Multimedia Subsystem) and vEPC (Virtual Evolved Packet Core) [5] are major use cases of SDN/NFV (Fig. 1.1).

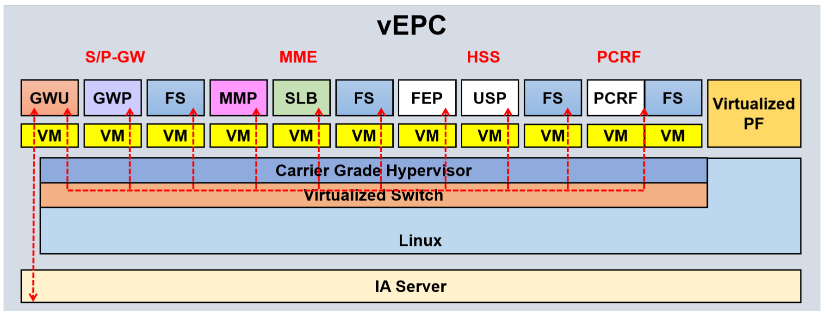


Figure 1.1: vEPC composition

Most of current EPC systems monitor their network by installing TAP devices in each physical link between EPC components (Fig. 1.2). To satisfy monitoring requirements in vEPCs, placing vTAP functionality on a virtual switch

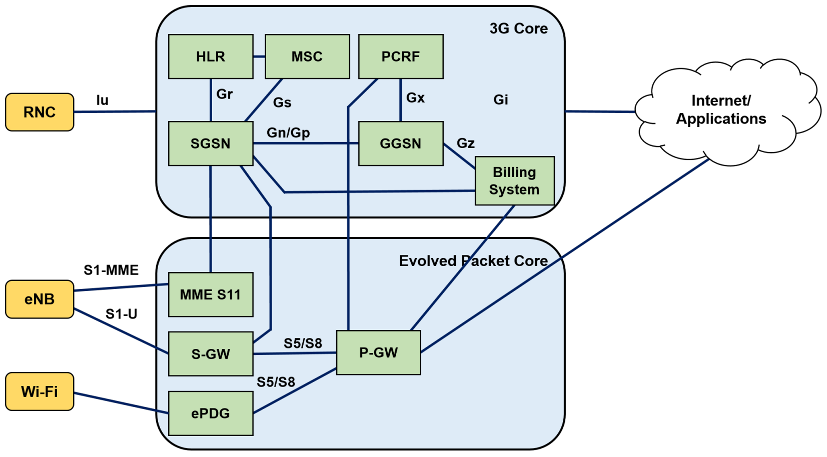


Figure 1.2: EPC composition

that connects VNFs such as vSGW (Virtual Serving Gateway) and vPGW (Virtual Packet Data Network Gateway) is a key component.

This thesis covers design and implementation of vTAP using Open vSwitch [6] with DPDK (Data Plane Development Kit) [7] and an OpenFlow SDN controller. Our vTAP spans both data plane and control plane of a network. The data plane function of vTAP performs packet processing including packet copy and forwarding according to TAP policies that are defined by the control plane of vTAP. The function is the same as hardware TAPs. To realize the function, port mirroring, or SPAN (Switched Port ANalyzer) [8], that is normally provided by current (virtual) switches can be a most simple and naive method. However, using the method in a virtual switch of a host server that implements NFV (e.g, vEPC and vIMS) results in significant performance degradation of VMs (VNFs) and packet switching in the virtual switch because the CPU-intensive port mirroring job uses the server's CPU resources that are shared with the running VMs. So, instead of port mirroring, we used Open vSwitch with DPDK (OVS-DPDK) [9] to implement a novel vTAP function that accelerates the data plane jobs and isolates the resource usage between vTAP and VMs. This thesis also provides performance comparisons of our vTAP and port mirroring, showing about 10 to 25 times throughput improvement in our implementation.

The control plane function of our vTAP has been designed to use the OpenFlow protocol and an SDN controller. We have implemented the control plane part as an SDN application that runs on top of an open source SDN controller - Open Network Operating System (ONOS) [10]. ONOS provides useful information of data plane composition such as (virtual) switches, hosts and flow rules to our vTAP application where actual TAP policies are defined by users. To install flow rules that reflect the required TAP policies in OpenFlow switches, OpenFlow control messages are generated and sent from the application to the switches. This SDN-based implementation offers flexible and centralized management of TAP policies.

The remainder of this thesis is organized as follows. Section II presents background and related work of this thesis, and Section III describes design considerations of our vTAP. We provide implementation details in Section IV and performance evaluation results in Section V. Finally, we conclude our work in Section VI including future work.

II. Background and Related Work

2.1 Background

IXIA and *Gigamon* are two representative companies to provide commercial vTAP solutions that are usually part of their datacenter network monitoring solutions. The purpose of these vTAP solutions is to provide visibility of traffic among VMs especially in a server virtualization environment. *IXIA* vTAP modifies an OS kernel of a host server to realize its vTAP function. A monitoring process operating in the host server aggregates copied packets, and provides analysis reports on in-flight traffic among VMs [11]. On the other hand, *Gigamon* installs a vTAP agent in each VM where traffic going in/out of the VM is copied, and then the copies are sent to an external monitoring system through an overlay network [12]. These two solutions have different implementation approaches, but they feature high performance with their proprietary implementation. However, in terms of customers, there are difficulties of vendor dependency, maintenance and expandability due to the product's exclusiveness. Whereas, our vTAP relies on open projects such as Open vSwitch (OVS), DPDK and ONOS that provide room for programmability and customization.

One of the major components of our vTAP is Open vSwitch. Besides its well-known features for various L2 switching functionalities, OVS is generally used as a virtual switch to provide internal and external network connectivity to

VMs in a server virtualization environment. With an upward trend of server and network virtualization in datacenter networks, interest about functionalities and performance of OVS is increasing. We used its port mirroring function for baseline measurement in evaluation and OpenFlow-based packet processing/switching functions.

DPDK is a data plane development framework including libraries for fast packet processing and drivers for DPDK-compatible network interface cards (NICs). It also supports EAL (Environment Abstraction Layer) to enable network applications to effectively use server's resources, and developers to easily implement their DPDK-based applications (e.g, traffic generator). The most important thing of packet processing mechanism in DPDK is PMD (Poll Mode Driver) to allow network applications to directly fetch and push packets from/to server's NICs, without intervention of the traditional Linux networking stack. This rapidly accelerates packet processing speed because it minimizes the number of context switchings and in-memory kernel-to-user packet copy that are frequently occurred by the traditional networking stack. DPDK also supports CPU affinity, multi-queue, hugepages and etc. in DPDK-compatible hardware NICs and virtual NICs.

In the overall design and implementation of our vTAP, it follows SDN (Software-Defined Networking) architecture. The major purpose of the SDN architecture is to separate data plane and control plane in a network. Over underlying data plane entities (switch, router), an SDN controller can enable the data plane entities to activate the actual packet processing operations that a target application needs

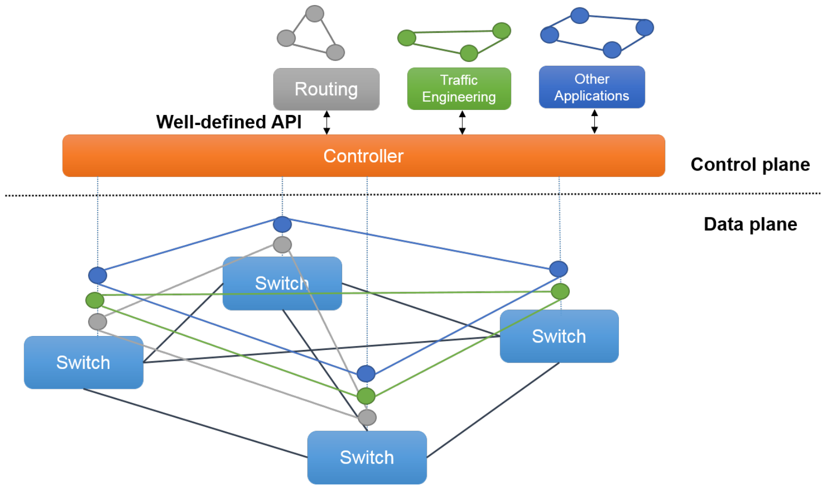


Figure 2.1: SDN architecture

by exposing APIs for users to specify related policies in a flexible and centralized manner (Fig. 2.1). This concept or architecture may reduce CAPEX/OPEX especially in a large datacenter network. Our vTAP application deploys packet copy and forwarding operations to some parts of data plane according to TAP policies that users define.

For translation of a high-level policy to specific data plane operations, OpenFlow protocol is generally used as a defacto standard for implementation of SDN. An OpenFlow-enabled switch maintains a set of rules that each includes various match fields and action directives for incoming packets. These flow rules are made by OpenFlow Flow-mod messages from the connected OpenFlow (SDN) controller. Other types of OpenFlow messages that can be made by a switch such as flow statistics are sent to the connected controller or vice versa. The latest version of OpenFlow is 1.5.1 and is evolving with increasing number of match fields to support various routing protocols, advanced features such as Group Table that we will cover in detail later, and etc.

Lastly, we use ONOS as an SDN controller where our vTAP application (control plane) is based on. ONOS is one of the most promising SDN controllers like ODL (OpenDaylight), targeting a carrier-grade SDN controller with rich functionality, stability and the distributed architecture for scalability. ONOS is made up of mainly the three layers: Application, Manager and Provider. Provider means SBIs (South Bound Interfaces) for communication with data plane entities. OpenFlow, NETCONF, LISP, and other protocol agents fall into this layer. Manager abstracts underlying network components into Java objects such as FlowRule, Device, Path and etc. It also manages distributed stores for synchronization of information between remote ONOS controllers. Application falls various SDN applications that can be implemented by APIs and Java objects that Manager provides to ONOS users.

2.2 Related Work

Overall, research on network monitoring using SDN and OpenFlow in data center networks has been much more active compared to research on vTAP itself. Planck [13] focuses on acceleration of traffic collection speed by using the port mirroring function of hardware OpenFlow switches. This work intentionally oversubscribes switch ports to gain sampling information of certain traffic flow, and analyzes it for network monitoring. NetAlytics [14] reduces overhead of real-time monitoring to data center servers by dynamically placing instances of Monitor, Aggregator and Processor in servers in a way that balances network bandwidth and load. NetAlytics uses DPDK to increase packet processing performance, and OpenFlow flow rules to copy traffic flows, assuming SDN-based data center networks. These two systems utilize similar technologies such as DPDK and port/flow mirroring that our vTAP implementation is based on, but they focus on server-to-server traffic with assumption of using hardware OpenFlow switches. On the other hand, our vTAP is more focusing on acceleration of VM-to-VM traffic duplication in virtual switches, and flexible policy management of virtual switches.

NFVPerf [15] detects bottleneck points of NFV through online performance monitoring especially in OpenStack testbeds. Collection of traffic among VMs in the same host or different hosts is realized by port mirroring of Open vSwitch bridges, and collected data is sent to performance analyzer that detects bottleneck indications from throughput and delay measurement. Despite of its novel detection algorithm, the use of port mirroring limits performance of collecting packet data.

There is work focusing on performance of virtual switches and their virtual interfaces. Because performance of virtual data plane forwarding is a key issue for migrating existing services running in bare metal servers or middle boxes into VMs, comparing virtual forwarding technologies such as Open vSwitch, IP forwarding, Linux Bridge and DPDK vSwitch is important [16]. Studying various composition of virtual interfaces supported by virtual switches is also crucial to understand virtual NIC implementation trends. Current implementation of DPDK-based virtual NIC (DPDK vhost-user) connected to a QEMU VM is mainly realized in user space on the host [17].

III. Design

Fig. 3.1 depicts the difference between a hardware TAP and a vTAP in a server virtualization environment. Hardware TAP still can be used for copying traffic going in and out of host machines, but we need vTAP to copy traffic among VMs especially in the same host over virtual links. A capture server (monitor) can be of a local entity like a VM in the same host, or external entity like a remote VM or probe box. So, vTAP also have to send copied packets to a remote entity according to configurations.

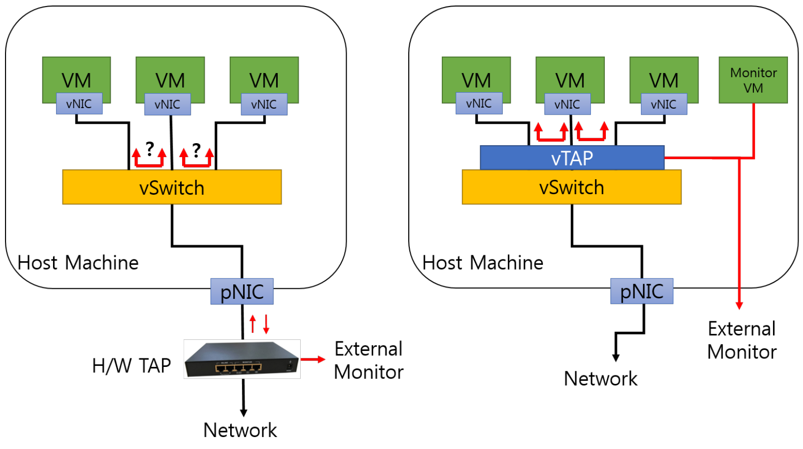


Figure 3.1: Physical (Hardware) TAP vs. Virtual TAP

The overall design of our vTAP integrating its data plane and control plane functionalities is shown in Fig. 3.2.

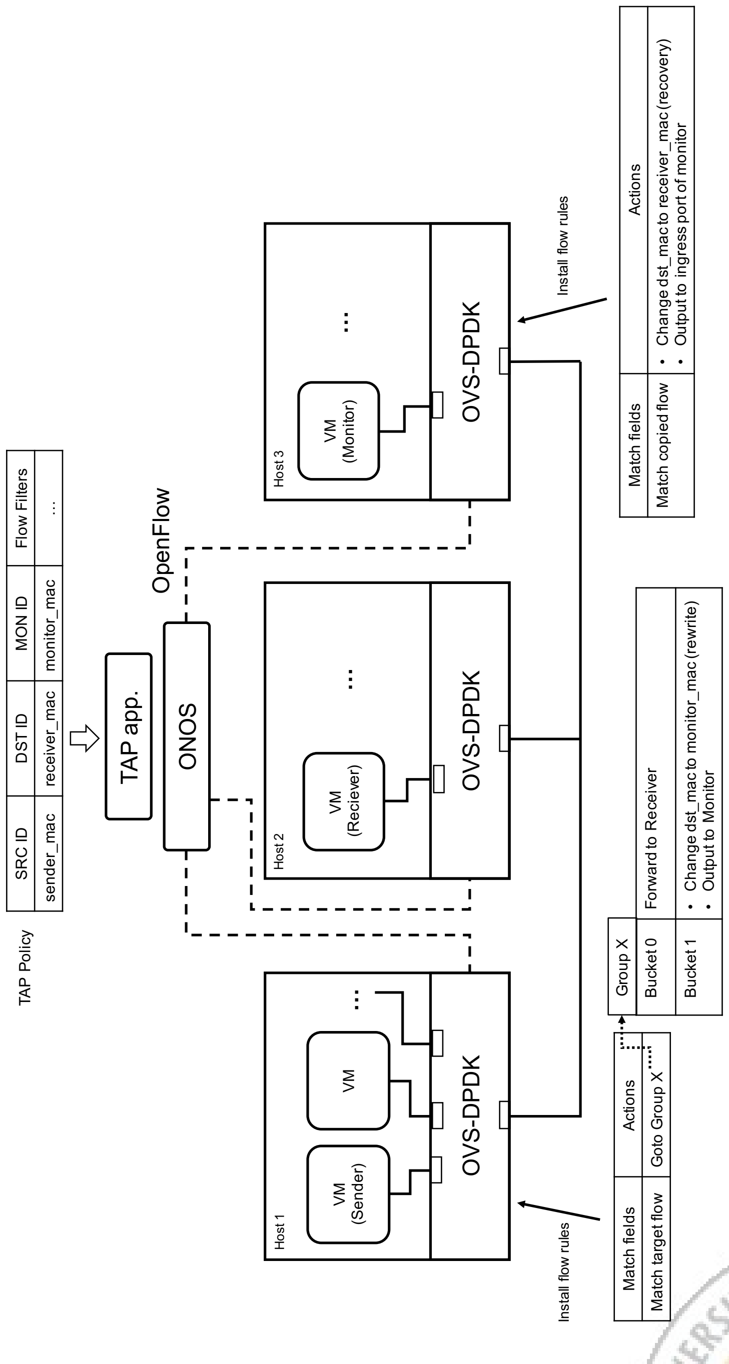


Figure 3.2: Overall design architecture of the proposed vTAP

3.1 vTAP Data Plane

We have chosen OVS-DPDK as a virtual switch that implements the vTAP functionality. OVS-DPDK is re-implementation of the packet processing mechanism of Open vSwitch (OVS) in user space so that the revision can accelerate packet processing speed by using features that the DPDK framework provides (Fig. 3.3). OVS-DPDK can replace OVS that is generally used for a virtual switch in a virtualization environment such as OpenStack, because OVS-DPDK does not only maintain most of existing features and management interfaces such as OpenFlow and Open vSwitch Database Management Protocol (OVSDB), but also provides configurations of PMD, CPU affinity, hugepages in both DPDK-compatible hardware NICs and virtual NICs. In the use of OVS-DPDK for vTAP implementation, we have mainly used the following four features.

Memory pool

Using a dedicated memory pool for OVS-DPDK to create and manage queues and buffers.

Hugepage

Spawning a VM on hugepages previously allocated in and shared with host machines.

PMD thread

Assigning multiple PMD threads to dedicated CPU cores according to the size of the network.

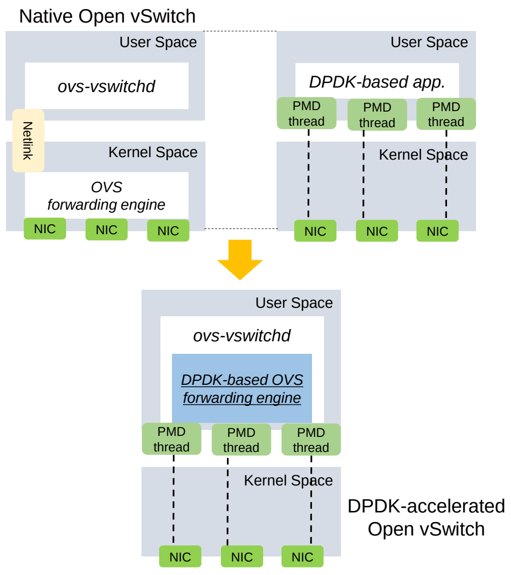


Figure 3.3: Open vSwitch with DPDK

CPU affinity

Pinning DPDK-related threads and vCPUs of VMs to the same or adjacent NUMA (Non-Uniform Memory Access) node to make them utilize the locality of references in memory accesses.

OVS officially supports the DPDK-based virtual interface (DPDK vhost-user) to connect VMs created by a hypervisor and its instance running as DPDK mode (OVS-DPDK) since its 2.4 version release.

In the initialization stage of OVS-DPDK, it is possible to increase packet processing performance by pinning PMD and DPDK core threads of OVS-DPDK to dedicated CPU cores in a certain NUMA node to realize CPU affinity and use of memory locality. Some amount of 2MB or 1GB size unit of hugepages also can be allocated for OVS-DPDK to realize zero-packet copy between kernel and user space, and reduce the number of TLB (Translation Lookaside Buffer) misses in the packet processing. In our work, if possible, we pinned PMD threads to the same NUMA node where vCPUs of VMs were pinned to maximize locality of memory references for packet processing between PMD threads and vCPUs over DPDK vhost-user interfaces. If a vCPU is operated in a different NUMA node with OVS-DPDK PMD threads, additional *Intel* QuickPath Interconnect (QPI) and performance issues are introduced [18].

Now, general packet transmissions among VMs are realized by OVS-DPDK. Ordering packet copy jobs to a virtual switch can be simply implemented by inputing port mirroring commands or requests through the virtual switch's CLI or proprietary protocols (if it supports). However, TAP policy management in this distributed manner is very exhausted and error-prone. So, we mentioned that our vTAP utilizes centralized and flexible management of SDN over OpenFlow protocol. To mediate the TAP management of the control plane and its realization in the data plane, we used OpenFlow Group Table. Group Table provides additional packet processing pipeline to the existing OpenFlow's Flow Table mechanism. Group Table supports four types of packet processing: ALL, SELECT, INDIRECT and FAST-FAILOVER. We used ALL type Group Table that takes any packets received as input and duplicate it to be operated on independently by each group bucket with different and distinct actions [19]. The input packet is copied up to the number of buckets.

The internal implementation of Group Table depends on a switch (including

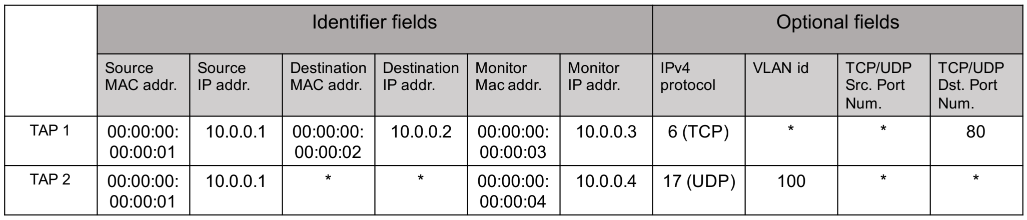
OVS-DPDK) that supports OpenFlow 1.1 and above. Our vTAP needs a Group Table instance with two buckets per TAP policy. Packets of a traffic flow that are matched with desirable flow rules inserted by our vTAP application (control plane) for routing are sent to the Group Table instance as the corresponding action at first. Then, each packet is copied to both the bucket 1 and bucket 2. Bucket 1 treats the copy as an original production packet, so it just outputs the packet to an egress port to the destination. Whereas, bucket 2 sends the packet to an egress port to one of capture (monitor) servers. To route the copy of the input packet holding the MAC and IP address of the original destination, bucket 2 also rewrites the addresses to those of the target capture server. Then, the edge switch of the capture server has to recover the original MAC and IP addresses from the rewritten MAC and IP addresses when it receives the copied packets, and forward them to the capture server. These stateful actions per TAP policy are realized by two flow rules that are installed by vTAP application with global information about the SDN network; the first flow rule is deployed in the edge switch of the sender server for packet copy, routing and address rewriting, and the second rule is deployed in the edge switch of the capture server for address recovery.

3.2 vTAP Control Plane

For flexible and centralized management of TAP policies, we have implemented an SDN controller application that provides a graphical user interface for users to easily define TAP policies and translates the definitions into OpenFlow messages for provision of related flow rules and Group Table instance in the data plane. We used ONOS as the SDN controller, since ONOS is one of the most promising carrier-grade SDN controllers similar to ODL (OpenDaylight), and ONOS offers useful controller APIs through various core services such as Device, Flow Rule, Group and other abstractions from the SDN concept and SBI (South Bound Interface). By using the references to the core services, developers can easily gain information of the underlying data plane, and effectively translate user requests from NBI (North Bound Interface) into desirable data plane provisions to realize their application purposes.

A TAP policy that can be specified in the application GUI is composed of a 6-tuple of identifier fields and 4-tuple of optional fields in the current vTAP (Fig. 3.4). Source MAC/IP addresses and Destination MAC/IP addresses identify a target sender VM and target receiver VM respectively. Monitor MAC/IP addresses identify a target capture machine (bare metal server or VM). By specifying only these identifier fields, all traffic flows among the sender and destination VM is transmitted, while the copy of packets of each flow are sent to the capture machine. Specifying the other optional fields means a fine-grained TAP policy definition that filters out interested certain traffic flows. Currently, VLAN ID, IP protocol type (TCP/UDP) and source/destination port numbers can be used for filtering, but wanted match fields are easily expandable using the OpenFlow field extension capability [20].

The pseudo code (Algorithm 1) describes the algorithm that fetches information from a given TAP policy and installs related flow rules for packet copy,



	Identifier fields						Optional fields			
	Source MAC addr.	Source IP addr.	Destination MAC addr.	Destination IP addr.	Monitor Mac addr.	Monitor IP addr.	IPv4 protocol	VLAN id	TCP/UDP Src. Port Num.	TCP/UDP Dst. Port Num.
TAP 1	00:00:00:00:00:01	10.0.0.1	00:00:00:00:00:02	10.0.0.2	00:00:00:00:00:03	10.0.0.3	6 (TCP)	*	*	80
TAP 2	00:00:00:00:00:01	10.0.0.1	*	*	00:00:00:00:00:04	10.0.0.4	17 (UDP)	100	*	*

Figure 3.4: An example of TAP policy specifications

address rewriting (sender edge) and address recovery (monitor edge).

Algorithm 1 : TAP Policy Apply

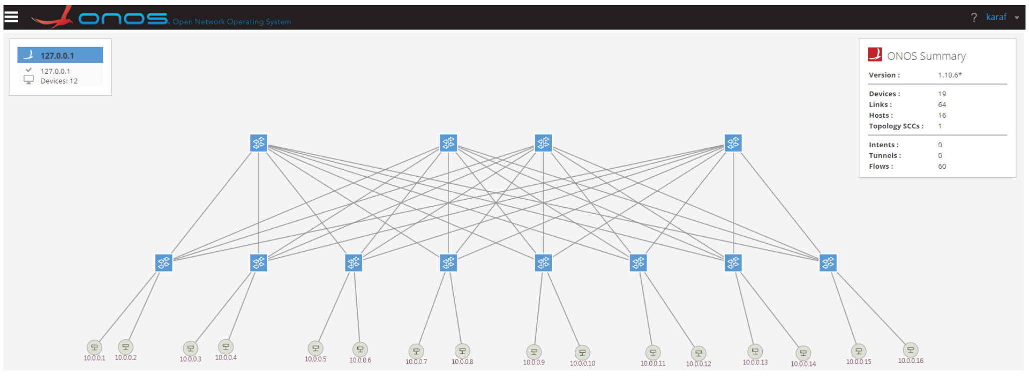
```
Require: Input ($policy$, $deviceSrv$, $hostSrv$, $topologySrv$)
 1: $devices \leftarrow deviceSrv.getAll()$
 2: for $device \in devices$ do
 3:     $dstHost \leftarrow hostSrv.getHost(policy.getDstMac)$
 4:     $dstDevice \leftarrow dstHost.getEdgeDevice()$
 5:     $monHost \leftarrow hostSrv.getHost(policy.getMonMac)$
 6:     $monDevice \leftarrow monHost.getEdgeDevice()$
 7:     if $device \equiv dstDevice$ then
 8:         $dstPath \leftarrow topologySrv.getPath(device, dstDevice)$
 9:         $outPortToDstDev \leftarrow dstPath.getSrcPort()$
10:         $monPath \leftarrow topologySrv.getPath(device, monDevice)$
11:         $outPortToMonDev \leftarrow monPath.getSrcPort()$
12:         $installRewriteRule($
13: $device, outPortToDstDev, outPortToMonDev, policy)$
14:     else if $device \equiv monDevice$ then
15:         $outPortToMonHost \leftarrow monHost.getInPort()$
16:         $installRecoveryRule($
17: $device, outPortToMonHost, policy)$
18:     end if
19: end for
```

IV. Implementation

Our vTAP implementation can be divided into control plane and data plane. As we mentioned in Section I, II, vTAP is needed especially in (datacenter) networks where server virtualization technique is used to deploy multiple VMs in a single server. A cloud service or an NFV-based system (Fig. 1.1) is realized by communication among VMs that operate subcomponents (e.g, application, VNF) across a single server or multiple servers.

So, we emulated a small datacenter network to show feasibility of our vTAP application that takes a target vTAP policy (Fig. 3.4) as an input from a network administrator, and inserts desired packet processing rules (Fig. 3.2) into the emulated switches as an output. We used Mininet 2.3.0 [21] to emulate a leaf-spine network with 4 spine switches (OVS), and 8 leaf switches connected with 2 hosts on each (Fig. 4.1). To simplify the network, we assumed the each of leaf switches as a Top-of-Rack (ToR) switch, and a MAC address of each host is directly mapped to an unique IP address of each host so that a vTAP policy identifies target hosts only by their MAC addresses.

We additionally implemented visualization UI for flow statistics in our vTAP so as to allow network administrators to easily identify in-flight traffic flows and provide their flow 5-tuple and throughput at each switch on their routing path (Fig. 4.3, 4.4). Administrators can find interested target flows such as suspicious or congestion traffic to apply vTAP policy on them, then can analyze the flows in



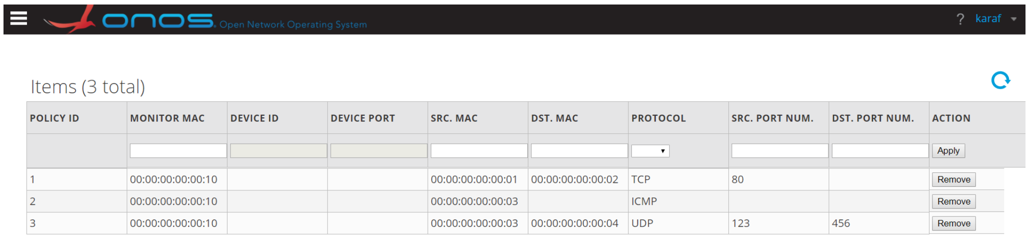
ONOS Summary

Version :	1.10.6*
Devices :	19
Links :	64
Hosts :	16
Topology SCCs :	1
Intents :	0
Tunnels :	0
Flows :	60

Figure 4.1: Topology view of the emulated network

packet-level using analysis tool or machine learning algorithms in a monitoring machine.

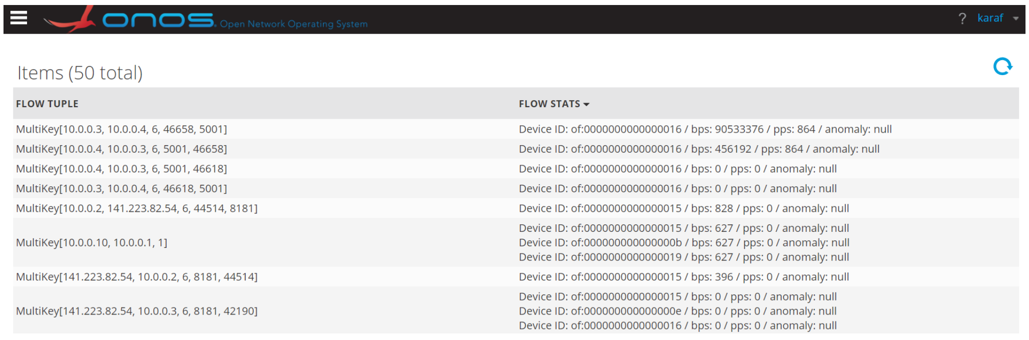
In the given emulated network, we tested overall operation of the system by taking several vTAP policies as inputs of the vTAP application. ONOS 1.10 was used as the base SDN controller where the vTAP application depends on and runs, and the Mininet emulator used Open vSwitch 2.7.0 as the base OpenFlow (software) switch rather than the default reference OpenFlow switch. Those OVS instances communicate with the ONOS controller via OpenFlow 1.3. We here describe two representative vTAP policies specified in Fig 4.2. The vTAP policy with ID 1 applies all TCP packets with source MAC address of 00:00:00:00:00:01, destination MAC address of 00:00:00:00:00:02 and TCP source port number of 80 (HTTP). The vTAP policy definition is transformed into two OpenFlow Flow-mod messages; one allows the edge switch of the sender host to forward the target packets normally and copy them with change of the destination MAC address as 00:00:00:00:00:10 to forward them to the monitor host (rewriting); the other allows the edge switch of the monitor host to forward the mirrored packets to the host with change of the destination MAC address as 00:00:00:00:00:02 (recovery). Likewise, The vTAP policy with ID 2 applies all ICMP packets with source MAC address of 00:00:00:00:00:03, by inserting a flow rule in the sender edge switch (rewriting) and a flow rule in the monitor edge switch (recovery).



Items (3 total)

POLICY ID	MONITOR MAC	DEVICE ID	DEVICE PORT	SRC. MAC	DST. MAC	PROTOCOL	SRC. PORT NUM.	DST. PORT NUM.	ACTION
									Apply
1	00:00:00:00:00:10			00:00:00:00:00:01	00:00:00:00:00:02	TCP	80		Remove
2	00:00:00:00:00:10			00:00:00:00:00:03		ICMP			Remove
3	00:00:00:00:00:10			00:00:00:00:00:03	00:00:00:00:00:04	UDP	123	456	Remove

Figure 4.2: UI for management of vTAP policies



Items (50 total)

FLOW TUPLE	FLOW STATS
MultiKey[10.0.0.3, 10.0.0.4, 6, 46658, 5001]	Device ID: of:0000000000000016 / bps: 90533376 / pps: 864 / anomaly: null
MultiKey[10.0.0.4, 10.0.0.3, 6, 5001, 46658]	Device ID: of:0000000000000016 / bps: 456192 / pps: 864 / anomaly: null
MultiKey[10.0.0.4, 10.0.0.3, 6, 5001, 46618]	Device ID: of:0000000000000016 / bps: 0 / pps: 0 / anomaly: null
MultiKey[10.0.0.3, 10.0.0.4, 6, 46618, 5001]	Device ID: of:0000000000000016 / bps: 0 / pps: 0 / anomaly: null
MultiKey[10.0.0.2, 141.223.82.54, 6, 44514, 8181]	Device ID: of:0000000000000015 / bps: 828 / pps: 0 / anomaly: null
MultiKey[10.0.0.10, 10.0.0.1, 1]	Device ID: of:0000000000000015 / bps: 627 / pps: 0 / anomaly: null Device ID: of:000000000000000b / bps: 627 / pps: 0 / anomaly: null Device ID: of:0000000000000019 / bps: 627 / pps: 0 / anomaly: null
MultiKey[141.223.82.54, 10.0.0.2, 6, 8181, 44514]	Device ID: of:0000000000000015 / bps: 396 / pps: 0 / anomaly: null
MultiKey[141.223.82.54, 10.0.0.3, 6, 8181, 42190]	Device ID: of:0000000000000015 / bps: 0 / pps: 0 / anomaly: null Device ID: of:000000000000000e / bps: 0 / pps: 0 / anomaly: null Device ID: of:0000000000000016 / bps: 0 / pps: 0 / anomaly: null

Figure 4.3: Default UI for collected flow statistics

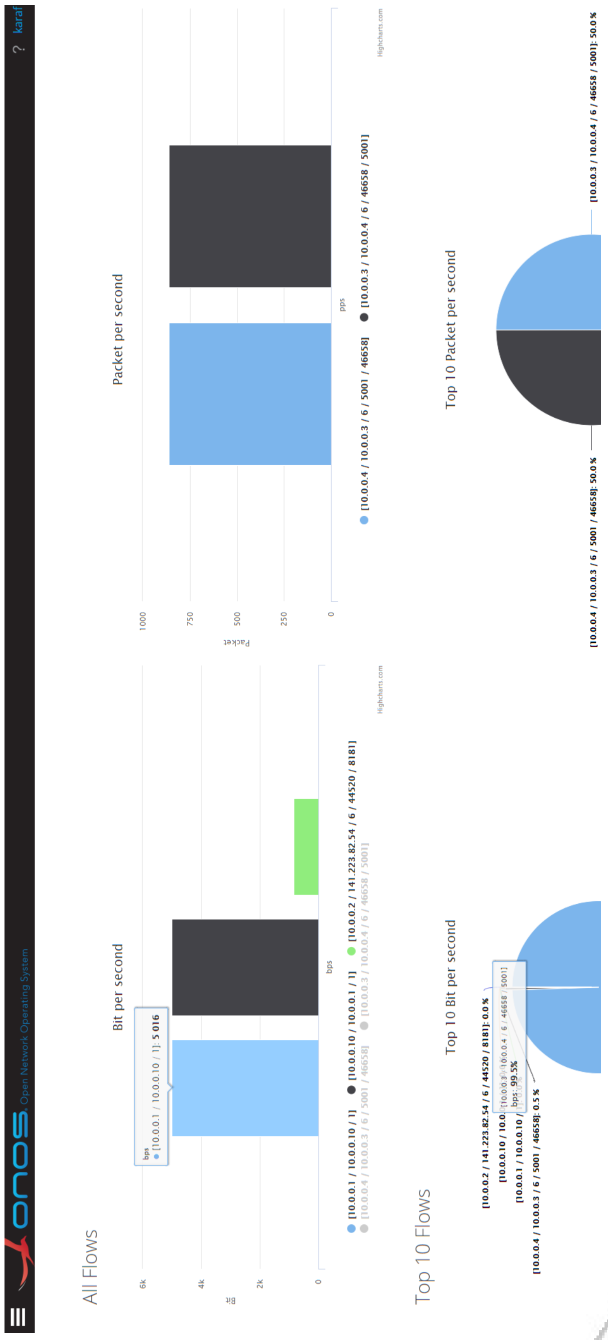


Figure 4.4: Visualization UI for collected flow statistics

V. Evaluation

Data plane implementation for performance evaluation of our vTAP was deployed in the following two testbeds: one pair of (Sender, Receiver and Monitor VM) (Fig. 5.1) and two pairs of (Sender and Receiver VM) and a single Monitor VM (Fig. 5.2). We used two bare metal servers with the same hardware specification (2.67 GHz Intel Xeon CPU with 6 cores x two NUMA nodes and 24GB RAM), and installed Open vSwitch 2.7.2 and DPDK 16.11 to operate a single DPDK-enabled OVS (OVS-DPDK) bridge as a virtual switch on each Linux CentOS 7.3 server. Each OVS-DPDK instance connects each of VMs through a DPDK-backed vhost-user type virtual interface (vNIC), and communicates with an ONOS 1.10 SDN controller running in a separated server through OpenFlow 1.3. Each VM has one vCPU thread pinned to a dedicated core and 1024MB memory backed by host server's 2MB unit hugepages. In each testbed, we used the kernel-based packet generation tool, pktgen [22], to generate various size of IPv4 packets in a Sender VM to send them to the paired Receiver VM, while the virtual switch copies the packets and forwards the duplicates to the Monitor VM.

Now, we present network performance evaluation results on each of the following test scenarios to show feasibility of our vTAP based on OVS-DPDK and OpenFlow Group Table, compared to the naive way to realize vTAP using only OVS and its port mirroring feature.

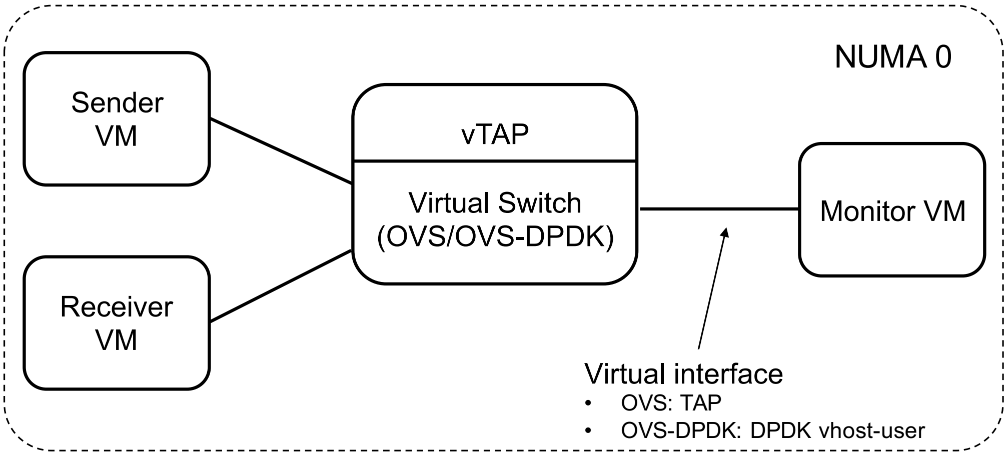


Figure 5.1: Testbed 1: one pair of (Sender, Receiver, Monitor)

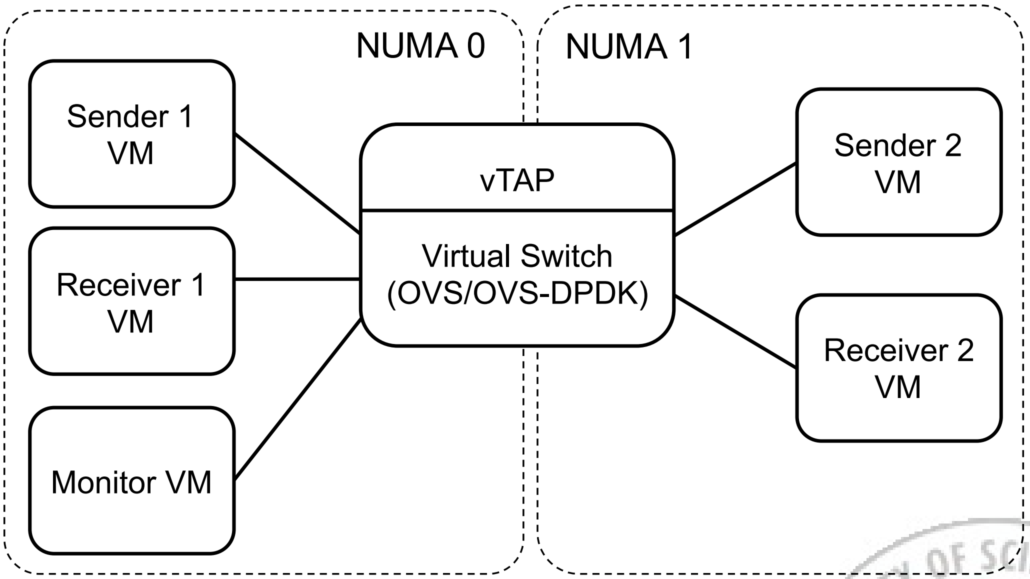


Figure 5.2: Testbed 2: two pairs of (Sender, Receiver) and a single Monitor

5.1 Port Mirroring vs. OpenFlow Group Table in normal OVS

OVS provides a port mirroring function to users through Open vSwitch Database (OVSDB) protocol [23]. To apply a port mirroring rule, a user has to input related commands over OVS's CLI, and the mirroring rule can be more detailed by specifying source port, destination port and output port for the copies. This type of configuration requires careful creation of batch files to apply a number of mirroring rules over OVS instances with avoiding configuration conflicts. It also lacks of granularity to realize flow-selective TAP policies (e.g, only interested in HTTP flows). However, the OpenFlow Group Table approach used in our vTAP can solve those problems as described in Section III. Apart from those considerations, we measured network performance of two approaches to realize TAP functionality in terms of throughput and pps (packet per second) in the testbed shown in Fig. 5.1. The figures in the table and graphs are average values with round-up of the first decimal place from 10 trials of each.

The measurements given in Table 5.1 and 5.2 show that about 6% decrease in both Receiver pps and throughput and Monitor pps and throughput in the case of Group Table, compared to port mirroring shown in Fig. 5.3.

5.2 Port Mirroring vs. OpenFlow Group Table in OVS-DPDK

To measure how much performance gain in the use of OVS-DPDK, and check whether it affects the performance of both port mirroring and Group Table

Table 5.1: vTAP performance measurement (port mirroring, normal OVS)

Packet size (bytes)	Receiver PPS	Receiver Throughput (Mbps)	Monitor PPS	Monitor Throughput (Mbps)
64	125,413	64	125,692	58
128	125,302	128	126,002	113
256	128,196	262	128,483	247
512	129,237	529	129,512	513
1,024	129,799	1,063	130,029	1,049
1,280	131,518	1,346	131,677	1,334
1,518	128,250	1,557	128,509	1,546

Table 5.2: vTAP performance measurement (Group Table, normal OVS)

Packet size (bytes)	Receiver PPS	Receiver Throughput (Mbps)	Monitor PPS	Monitor Throughput (Mbps)
64	119,962	61	120,468	56
128	119,778	122	120,086	108
256	120,927	247	121,353	233
512	121,864	499	122,423	485
1,024	122,393	1,002	122,700	991
1,280	122,708	1,256	122,830	1,243
1,518	120,632	1,464	121,240	1,455

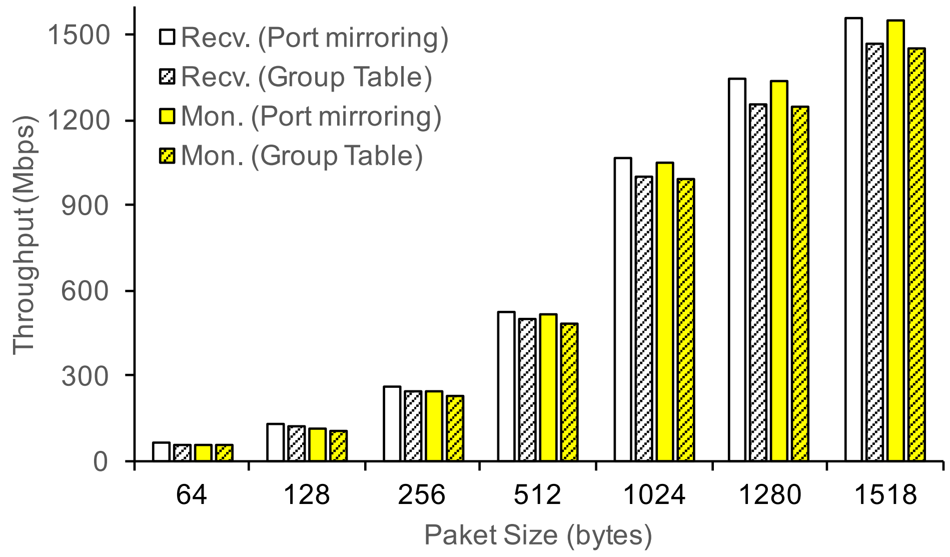


Figure 5.3: vTAP throughput comparison of Port mirroring and Group Table in Receiver and Monitor

approaches, we repeated the same tests above except replacing OVS with OVS-DPDK. Additionally, we pinned all the vCPU threads and the only one PMD thread to cores in the same NUMA node to maximize locality of memory references, and allocated 1024MB of hugepages for the OVS-DPDK configurations.

The measurements given in Table 5.3 and 5.4 show that about 2% increase in Receiver pps and throughput and about 0.5% increase in Monitor pps and throughput in the case of Group Table, compared to port mirroring. We concluded that no meaningful performance differences are observed between port mirroring approach and Group Table approach in the case of OVS-DPDK. However, the performance of OVS-DPDK for the vTAP functionality was observed, showing a significant improvement (Fig. 5.4). The gap spans from about 8 times

Table 5.3: vTAP performance measurement (port mirroring, OVS-DPDK)

Packet size (bytes)	Receiver PPS	Receiver Throughput (Mbps)	Monitor PPS	Monitor Throughput (Mbps)
64	2,787,450	1,427	2,697,358	1,253
128	2,620,189	2,683	2,661,700	2,387
256	2,731,848	5,594	1,717,897	3,295
512	2,346,705	9,612	1,754,977	6,975
1,024	1,250,409	10,243	1,267,004	10,215
1,280	1,078,537	11,044	1,087,539	10,997
1,518	968,994	11,767	975,531	11,723

Table 5.4: vTAP performance measurement (Group Table, OVS-DPDK)

Packet size (bytes)	Receiver PPS	Receiver Throughput (Mbps)	Monitor PPS	Monitor Throughput (Mbps)
64	2,904,785	1,487	2,686,912	1,247
128	2,685,778	2,750	2,685,256	2,409
256	2,779,845	5,693	1,717,954	3,304
512	2,370,807	9,710	1,757,060	6,984
1,024	1,255,015	10,281	1,272,086	10,257
1,280	1,087,108	11,131	1,096,598	11,092
1,518	977,926	11,875	984,650	11,832

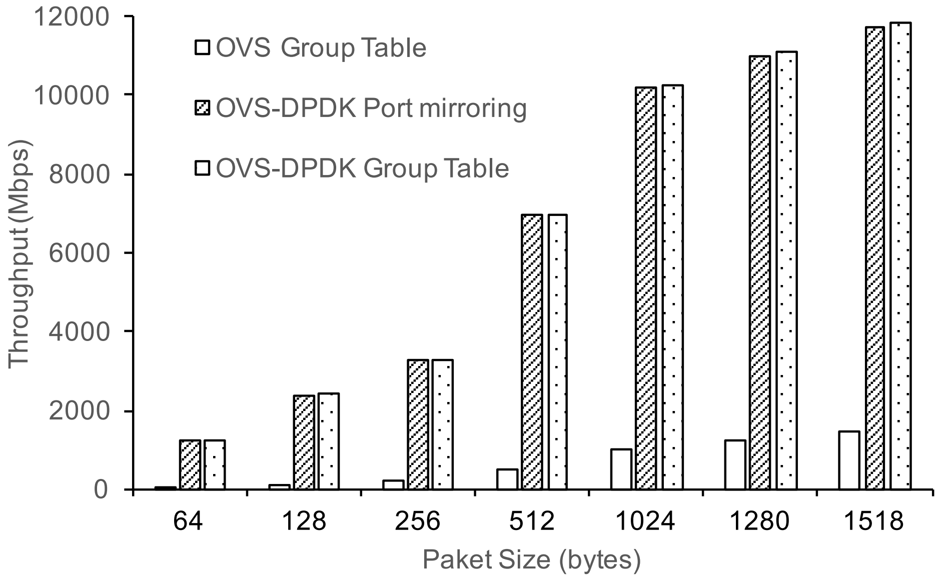


Figure 5.4: vTAP throughput comparison of OVS and OVS-DPDK in Monitor

for 1518-bytes packets to about 25 times for 64-bytes packets, compared to those of OVS. vTAP on small size packets in the packet processing pipeline of OVS involves much more CPU interventions resulting in packet drops and performance degradation than large size packets. However, vTAP based on OVS-DPDK with the PMD-based packet processing pipeline that allows packets to bypass the Linux kernel networking stack to avoid CPU interrupts, and the use of dedicated memory pool to realize zero-copy of packets largely improves its performance.

5.3 Effect of PMD threads in OVS-DPDK

In this measurement, we tried to discover the effect of the number of PMD threads on the vTAP functionality. We spawned an additional pair of Sender and

Receiver VMs, and measured vTAP performance gap between the case of using only one PMD thread and using two PMD threads. In the case of single PMD, we pinned each of 5 vCPUs and the single PMD thread to a core located in the same NUMA node as shown in Fig. 5.2. Due to the limited number of 6 cores in one NUMA node, we placed one PMD thread, one pair of Server-Receiver VMs, and Monitor VM in the NUMA 0, and the remaining PMD thread and one pair of Server-Receiver VMs are deployed in the NUMA 1 for the two PMD threads scenario.

The measurements given in Table 5.5 and 5.6 show that about 84% increase in Receiver 1 throughput, 87% increase in Receiver 2 throughput, and 35% increase in Monitor throughput as illustrated in Fig. 5.5. The result also shows that Monitor could not guarantee its throughput as the sum of the two Receivers' throughput in both cases. This is because the single vCPU of Monitor is not enough to handle the whole packets mirrored from the two concurrent traffic flows, resulting in many packet drops at the vNIC of Monitor. The number of packet drops is larger especially in small size packets because they necessarily involve more interruptions to the vCPU of Monitor. This could be mitigated by allocating more vCPUs to the Monitor VM. Increasing the number of PMD thread is not a solution to guarantee the effective throughput at Monitor. We consider it as future work to find relations between the number of vCPUs of Monitor and acceptable maximum traffic volume without any packet drops at Monitor.

In this evaluation, we can conclude that as the number of Sender-Receiver pairs is added or overall traffic volumes are increased, vTAP performance can be improved at the cost of using idle CPU cores for additional PMD threads and vCPUs.

Table 5.5: vTAP performance measurement (Group Table, OVS-DPDK, one PMD thread)

	Receiver 1		Receiver 2		Monitor	
Packet size (bytes)	PPS	Throughput (Mbps)	PPS	Throughput (Mbps)	PPS	Throughput (Mbps)
64	1,428,289	731	1,303,902	667	2549178	1,183
128	1,240,971	1,270	1,219,225	1,248	2,436,770	2,185
256	1,355,872	2,776	1,273,592	2,608	1,726,259	3,317
512	1,179,136	4,829	1,075,859	4,406	1,751,487	6,950
1,024	564,215	4,622	551,638	4,519	1,072,958	8,686
1,280	489,371	5,011	481,275	4,928	952,156	9,620
1,518	457,422	5,554	444,207	5,394	870,952	10,468

Table 5.6: vTAP performance measurement (Group Table, OVS-DPDK, two PMD threads)

	Receiver 1		Receiver 2		Monitor	
Packet size (bytes)	PPS	Throughput (Mbps)	PPS	Throughput (Mbps)	PPS	Throughput (Mbps)
64	2,824,075	1,445	2,718,243	1,391	2,343,191	1,088
128	2,654,693	2,718	2,552,526	2,613	2,356,312	2,119
256	2,717,280	5,564	2,637,447	5,401	1,535,653	2,952
512	2,348,520	9,619	2,294,205	9,397	1,542,191	6,133
1,024	1,060,279	8,685	1,030,101	8,438	1,555,016	12,555
1,280	857,901	8,784	841,325	8,615	1,527,316	15,469
1,518	722,424	8,773	714,475	8,676	1,420,458	17,078

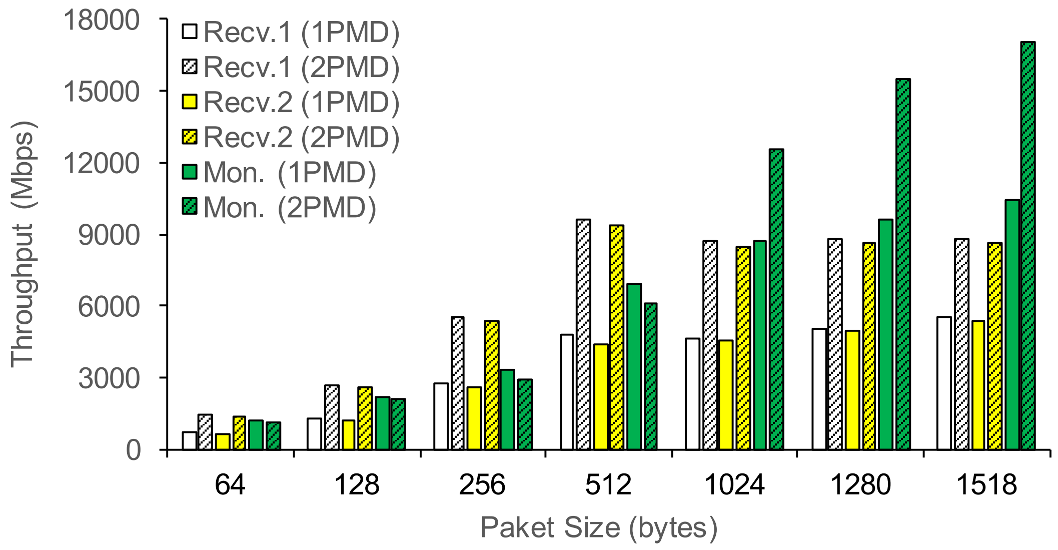


Figure 5.5: vTAP throughput comparison of the use of a single PMD and two PMDs in OVS-DPDK

VI. Conclusion

In this thesis, we have presented the design and implementation of vTAP mainly based on OVS-DPDK and OpenFlow. The data plane implementation of our vTAP includes the use of OVS-DPDK as virtual switches, and the provision of flow rules with Group Table based packet copy and forwarding actions to realize desired TAP policies. The control plane of vTAP implements an SDN (ONOS) controller application that provides a GUI to allow users to define TAP policies with the identifier specifications of source, destination and monitor entities, and optional specifications of various match fields to filter out interested traffic flows. A TAP definition is transformed into related OpenFlow flow rules in some of edge switches that are mainly virtual switches in a server virtualization environment such as datacenter.

The comparison of OVS port mirroring and OpenFlow Group Table as packet copy methods provides no meaningful performance differences. So, we chose to use the Group Table approach to our vTAP implementation due to its flexibility and expandability for TAP policy management.

Another comparison of OVS and OVS-DPDK to perform the actual packet processing (copy and forwarding) needed in vTAP provides significant performance gap between the two types of virtual switches. We observed that the use of OVS-DPDK offers 8 to 25 times throughput increase according to size of packet generated. So, we can conclude that, by allocating additional CPU cores to DPDK PMD threads, our vTAP can be used for high performance packet monitoring in a datacenter where East-West traffic among VMs is prevalent. However, limited CPU resources of a host server may have to be used for a VM itself to support running applications/VNFs or avoid excessive packet drops. We plan to find a function of the required number of vCPUs according to increase of traffic volume along with the number of DPDK PMD threads (Section 5.3).

For another future work, we consider the deployment of our vTAP to popular and practical cloud environments such as OpenStack. This may require to change the main component of OpenStack networking from Neutron to ONOS, and replace default OVS instances with OVS-DPDK in each compute node. Our vTAP supports flow level TAP polices, while other solutions support switch port level TAP policies. We may also support the additional type of TAP policy so that users can satisfy various traffic monitoring needs such as DPI, IDS and IPS using the accelerated data plane and flexible policy management of our vTAP.

요 약 문

최근, 늘어나는 네트워크 트래픽과 보다 높은 서비스 품질에 대한 요구사항과 함께 클라우드 서비스가 활발히 사용됨에 따라, 데이터센터에서의 효과적인 자원 사용을 가능케 하는 서버(호스트) 가상화 기술이 큰 주목을 받고 있다. vTAP (Virtual Test Access Port)은 이러한 서버상에서 가상 스위치를 통해 구축되는 가상 네트워크 링크에 기존의 하드웨어 TAP 장비를 설치할 수 없는 문제점을 극복하고, 가상머신들 간 트래픽을 패킷 수준에서 모니터링할 수 있게 한다. vTAP 기능은 가상 스위치를 통해 구현할 수 있는데, 가장 기본적인 포트 미러링(Port Mirroring) 방식의 패킷 복제는 가상 스위치의 성능을 상당히 저하시키며 개별적인 수동 설정이 필요하기 때문에 대규모 데이터센터 네트워크에는 적절하지 않다.

본 논문에서는 대표적인 가상 스위치인 Open vSwitch와 고속 패킷 프로세싱 기술인 DPDK (Data Plane Development Kit)를 기반으로 vTAP 기능을 구현하고, SDN (Software-Defined Networking) 컨트롤러에서 네트워크 관리자가 설정한 TAP 정책이 vTAP 기능과 연동되어 실현되도록 OpenFlow 프로토콜을 사용한다. 이를 통해 관리자는 중앙 집중화된 컨트롤러에서 전체적인 네트워크 뷰(View)를 가지고 TAP 정책을 유연하게 설정 및 관리할 수 있으며, DPDK에 유휴 서버 자원을 할당하여 기존 구현 기술 대비 초당 10배 이상의 속도로 복제 패킷을 수집할 수 있다. 실험을 통해 대용량 트래픽이 발생하는 환경에서 고속 패킷 모니터링의 가능성을 보였으며, 후속 연구를 통해 수집된 패킷 정보를 바탕으로 네트워크 분석, 보안, 관리 등의 응용 기술을 위한 기반 기술로 활용될 수 있음을 보이고자 한다.

References

[1] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3), 2012.

[2] Nick McKeown. Software-defined networking. *INFOCOM keynote talk*, 17(2):30–32, 2009.

[3] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[4] Oliver Hohlfeld, Thomas Zinner, Theophilus Benson, and David Hausheer. Special issue on software-defined networking and network functions virtualization for flexible network management. *International Journal of Network Management*, 26(1):4–5, 2016.

[5] Omer Narmanlioglu and Engin Zeydan. New era in shared cellular networks: Moving into open and virtualized platform. *International Journal of Network Management*, pages e1986–n/a. e1986 nem.1986.

[6] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan J Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al.

The design and implementation of open vswitch. In *NSDI*, pages 117–130, 2015.

[7] DPDK. Data plane development kit. Available at `http://dpdk.org/`.

[8] Jian Zhang and Andrew Moore. Traffic trace artifacts due to monitoring via port mirroring. In *End-to-End Monitoring Techniques and Services, 2007. E2EMON'07. Workshop on*, pages 1–8. IEEE, 2007.

[9] Open vSwitch. Open vswitch with dpdk. Available at `http://docs.openvswitch.org/en/latest/intro/install/dpdk/`.

[10] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[11] IXIA. Ixia phantom vtap with tapflow filtering. Technical report, 2016. Available at `https://www.ixiacom.com/sites/default/files/2016-07/V-DS-Phantom-vTap.pdf`.

[12] Gigamon. Gigavue-vm data sheet. Technical report, 2016. Available at `https://www.gigamon.com/content/dam/resource-library/korean/data-sheet/ds-gigavue-vm-virtual-machine-kr.pdf`.

[13] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. Planck: Millisecond-scale mon-

itoring and control for commodity networks. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 407–418. ACM, 2014.

[14] Guyue Liu, Michael Trotter, Yuxin Ren, and Timothy Wood. Netalytics: Cloud-scale application performance monitoring with sdn and nfv. In *Proceedings of the 17th International Middleware Conference*, page 8. ACM, 2016.

[15] Priyanka Naik, Dilip Kumar Shaw, and Mythili Vutukuru. Nfvperf: Online performance monitoring and bottleneck detection for nfv. In *Network Function Virtualization and Software Defined Networks (NFV-SDN), IEEE Conference on*, pages 154–160. IEEE, 2016.

[16] Paul Emmerich, Daniel Raumer, Florian Wohlfart, and Georg Carle. Performance characteristics of virtual switching. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 120–125. IEEE, 2014.

[17] Yoshihiro Nakajima, Hitoshi Masutani, and Hirokazu Takahashi. High-performance vnic framework for hypervisor-based nfv with userspace vswitch. In *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*, pages 43–48. IEEE, 2015.

[18] Ciara Loftus. Use the data plane development kit with open vswitch* to create vhost user non-uniform memory access awareness, 2016. Available at `https://software.intel.com/en-us/articles/vhost-user-numa-awareness-in-open-vswitch-with-dpdk`.

[19] Ryan Izard. How to work with fast-failover openflow groups, 2017. Available at https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/7995427/How+to+Work+with+Fast-Failover+OpenFlow+Groups.

[20] Ryan Izard. How to add an openflow experimenter extension, 2016. Available at https://floodlight.atlassian.net/wiki/spaces/floodlightcontroller/pages/22380593/How+to+Add+an+OpenFlow+Experimenter+Extension.

[21] Mininet. An instant virtual network on your laptop (or other pc). Available at http://mininet.org/.

[22] Robert Olsson. Pktgen the linux packet generator. In *Proceedings of the Linux Symposium, Ottawa, Canada*, volume 2, pages 11–24, 2005.

[23] SDxCentral. What is open vswitch database or ovsdb? Available at https://www.sdxcentral.com/cloud/open-source/definitions/what-is-ovsdb/.

Acknowledgements

먼저 석사학위를 성공적으로 이수할 수 있게 물심양면으로 도와주시고 항상 저를 지지해주신 부모님께 감사의 말씀을 전합니다. 그리고 어린 시절부터 저를 보살펴 주시면서 제가 엇나가지 않게 인도해주신 할머니와 고모님께도 감사의 말씀을 드리고 싶습니다. 또한 좋은 연구 환경 속에서 다양한 연구를 지도해주시고, 학생들의 건강, 인격 형성 등에도 많은 신경을 써주신 홍원기 교수님께 깊은 감사의 말씀을 드립니다.

포항공대와 대학원생이라는 새로운 환경과 역할에서 예상치 못하게 어려웠던 시기도 있었고 지금도 아직 많이 부족하다고 느끼지만, 지금와서 되돌아 보니 3년이란 시간 동안 학업뿐만 아니라 다양한 방면에서 많은 성장을 이룬것 같아 뿌듯한 마음이 들고, 쉽게 경험할 수 없는 굉장히 값진 경험이었다고 생각됩니다.

언제나 긍정적인 연구실의 분위기 메이커 정태열 박사님, 부지런하고 꾸준한 점을 꼭 배우고 싶은 종환이 형, 그리고 동기로 같이 입학해서 지금은 연구실의 실세가 된 도영(부담이 많겠지만 잘 해낼거라 생각), 똑똑하고 예상치 못한 풋살 실력자 준묵, 재밌고 배려심 깊은 경찬, 활기차고 시원시원한 성격의 동호, 조용하지만 묵묵히 자기 할 일 하는 지범, 그리고 먼 타지에서 외롭겠지만 열심히 생활하는 Vanesco와 Tu 모두 좋은 시간 보낼 수 있어 좋았고, 각자가 생각하고 있는 앞날에 좋은 기운이 있기를 바랍니다. 또, 신입생 때 여러모로 도움주신 정재윤 박사님, 이건 박사님, 한윤선 박사님께도 감사의 마음을 전합니다. 저도 목표하는 바를 이룰 수 있도록 앞으로도 열심히 하겠습니다.

Curriculum Vitae

Name : Seyeon Jeong

Research Interest

Software-Defined Networking (SDN); Network Function Virtualization (NFV); Network Traffic Monitoring and Analysis

Education

2009 – 2015	School of Computer Science and Engineering, Kyungpook National University (B.S.)
2015 – 2018	Department of Computer Science and Engineering, Pohang University of Science and Technology (M.S.)

Research/Project Experience

2015. 3. – 2018. 2.	ICBMS 플랫폼 간 정보모델 연동 및 서비스 매쉬업을 위한 스마트 중재 기술 개발 (Funded by IITP)
2015. 6. – 2017. 5	한 · 미 SDN/NFV WAN 네트워크 안전성 기술연구 및 TEST BED 구축 (Funded by IITP)

Publications: International Conference

1. **Seyeon Jeong**, Doyoung Lee, James Won-Ki Hong, ”OpenFlow-based Virtual TAP using Open vSwitch and DPDK”, 16th IEEE/IFIP Network Operations and Management Symposium (NOMS 2018), Taipei, Taiwan, April 23-27, 2018. (Accepted to appear)

2. Doyoung Lee, **Seyeon Jeong**, James Won-Ki Hong, ”OpenAPI-based Message Router for Mashup Service Development”, 19th Asia-Pacific Network Operations and Management Symposium (APNOMS 2017), Seoul, Korea, Sep. 27-29, 2017, pp. 94-99. (Best Paper Award)

3. **Seyeon Jeong**, Doyoung Lee, Jonghwan Hyun, Jian Li, James Won-Ki Hong, ”Application-aware Traffic Engineering in Software-Defined Network”, 19th Asia-Pacific Network Operations and Management Symposium (APNOMS 2017), Seoul, Korea, Sep. 27-29, 2017, pp. 315-318.

4. Doyoung Lee, **Seyeon Jeong**, Taeyeol Jeong, Jae-Hyoung Yoo, James Won-Ki Hong, ”ICBMS SM: A Smart Mediator for Mashup Service Development”,

18th Asia-Pacific Network Operations and Management Symposium (APNOMS 2016), Kanazawa, Japan, Oct. 5-7, 2016, pp. 1-6.

5. **Seyeon Jeong**, Doyoung Lee, Junemuk Choi, Jian Li, James Won-Ki Hong, "Application-aware Traffic Management for OpenFlow Networks", 18th Asia-Pacific Network Operations and Management Symposium (APNOMS 2016), Kanazawa, Japan, Oct. 5-7, 2016, pp. 1-5.

Publications: Domestic Journals

1. **정세연**, 홍지범, 홍원기, "서버 가상화 환경에서 트래픽 모니터링을 위한 Virtual TAP 설계 ", KNOM Review, Vol. 20, No. 1, August 2017, pp. 1-8.
2. 이도영, **정세연**, 현종환, 이건, 홍원기, "소프트웨어 정의 네트워크에서의 응용 프로그램 인지 트래픽 엔지니어링 기법 연구", KNOM Review, Vol. 19, No. 2, December 2016, pp. 1-12.
3. **정세연**, 이도영, 리건, 유재형, 홍원기, "다중 컨트롤러 환경에서의 제어 평면 모니터링 및 스위치 배치 방법에 대한 연구", KNOM Review, Vol. 18, No. 1, August 2015, pp. 11-24. (MSIP/IITP support)

Publications: Domestic Conference

1. 이도영, **정세연**, 홍원기, "매쉬업 서비스 개발을 위한 OpenAPI 기반 메시지 라우터 연구", KNOM Conference 2017, Gwangju, Korea, June 2-3, 2017, pp. 19-20.

2. **정세연**, 이도영, 최준묵, 홍원기, "DPI를 이용한 SDN 트래픽 매니지먼트 시스템", KNOM Conference 2016, Chuncheon, Korea, May 12-13, 2016, pp. 6-10. (Best Paper Award)

3. 이도영, **정세연**, 정태열, 홍원기, "ICBMS 플랫폼 연동 및 매쉬업 서비스 개발을 위한 Smart Mediator 연구", KNOM Conference 2016, Chuncheon, Korea, May 12-13, 2016, pp. 71-75.