Design and Implementation of Virtual TAP for Software-Defined Networks

- Master Thesis Defense -

Seyeon Jeong

Supervisor: Prof. James Won-Ki Hong
Dept. of CSE, DPNM Lab., POSTECH, Korea
jsy0906@postech.ac.kr

2017.12.14

Contents

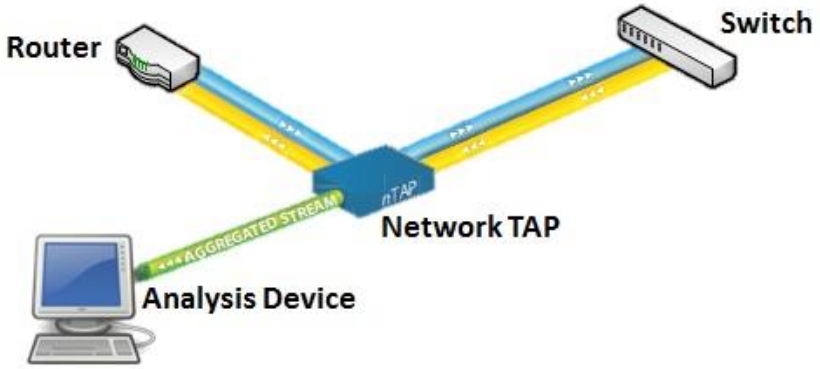
- Introduction
- Background and Related Work
- Design and Implementation
- Evaluation and Conclusion

Introduction

Introduction

❖ TAP (Test Access Point) Device

- A dedicated hardware device installed on a physical link b/w server and switch or b/w switch and router
  - Split packet signals and forward the duplications to external
- Support for high speed duplication without affecting original traffic
  - Price (1Gbps link) : $50 ~ $200



Source: http://www.usr.com/products/networking-taps/usr4503/

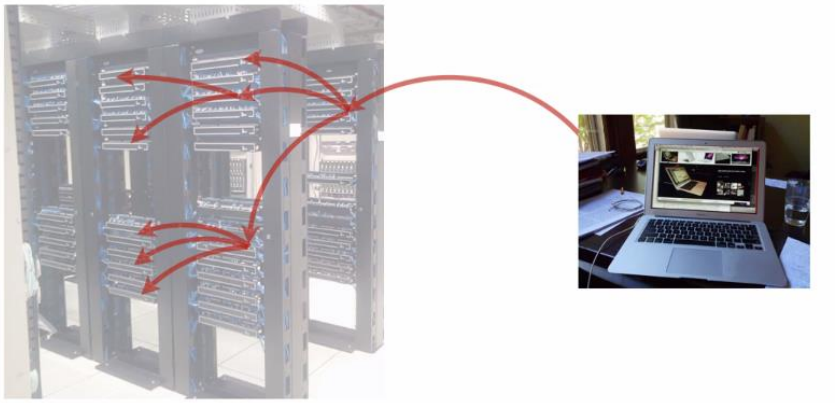
Introduction

❖ Server (host) Virtualization

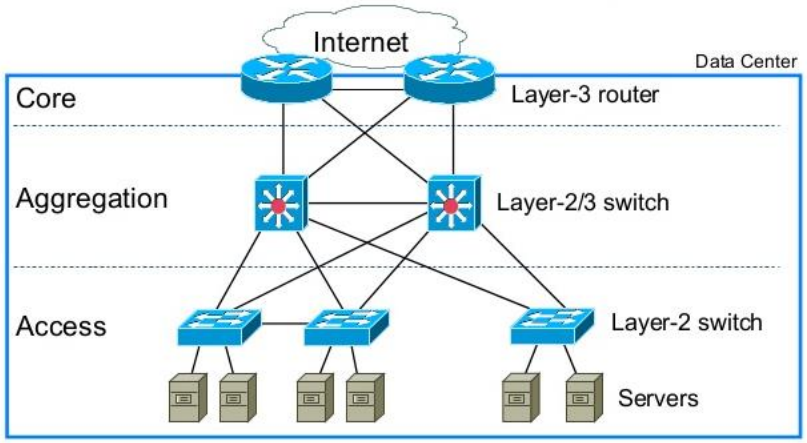
- Enables VMs (Virtual Machines) to effectively use the host server's shared resources
- A virtual switch is needed for connectivity among VMs

❖ Cloud Service

- Is realized over Data Center Networks (DCNs)
  - Many commodity servers in server virtualization
- A service request (ex. web search) involves a lot of network flows among VMs (East-West traffic)



Source: https://www.coursera.org/learn/cloud-networking/lecture/qSi4A/1-1-1-application-and-traffic-patterns

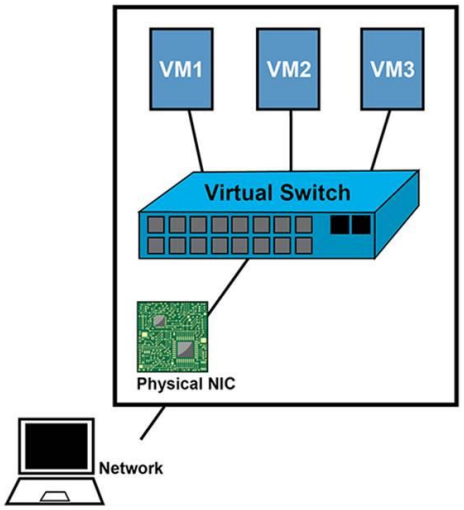


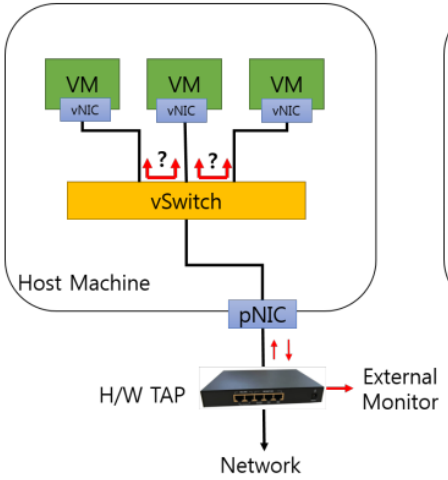
Source: https://www.slideshare.net/gunawanjusuf/a-scalable-commodity-data-center-network-architecture-40807428

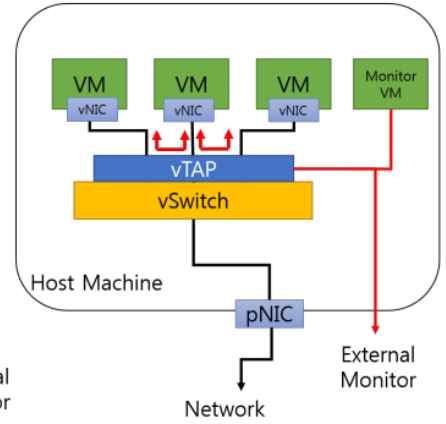
Introduction

❖ Virtual TAP (vTAP)

- Needs for traffic monitoring and analysis in DCNs are increasing
- TAP devices cannot be deployed in virtual links of server virtualization
- vTAP should provide inter-VM traffic monitoring with high performance packet processing







Introduction

❖ Motivation

- Simplest way for vTAP: port mirroring
  - Large computing overhead (CPU intensive)
  - Manual and error-prone management (vendor-specific)
- Needs for efficient packet monitoring in DCN
  - Increasing needs for cloud service (quality)
  - Machine Learning for network behavior, anomaly detection, etc.

❖ Research Goals

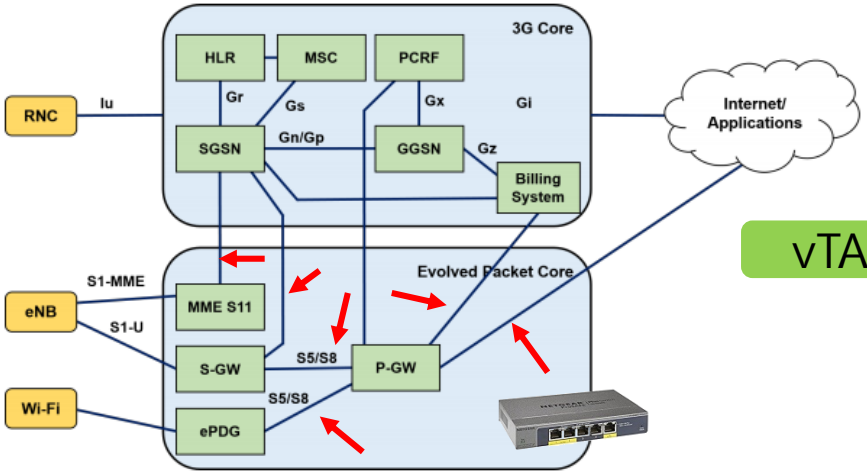
- Design and implementation of vTAP
  - Find an alternative approach to port mirroring
    - High performance packet processing
    - More centralized and efficient policy management
  - Apply promising network paradigm and combine related open source projects
    - Compatibility to today's DCN
    - Expandability

Background & Related Work

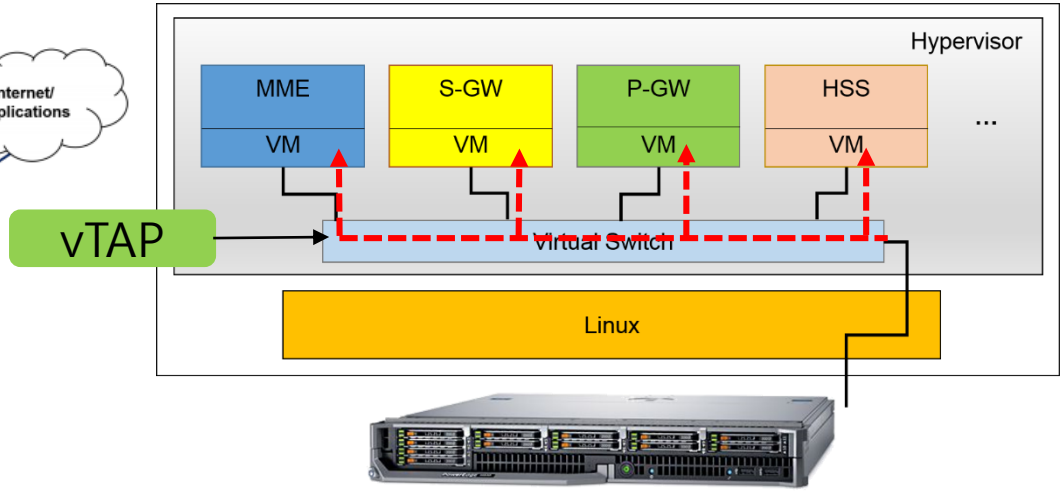
Background

❖ SDN/NFV use cases

- Telcos' try to softwarization of existing hardware-oriented networks
  - Preparation for 5G service requirements (efficiency, scalability, etc.)
- vEPC (virtualized Evolved Packet Core), vIMS (virtualized IP Multimedia Subsystem), etc.
  - Existing EPCs require per-link TAP device
  - VM-to-VM traffic passes through virtual switches in a virtualized system



- EPC -



- vEPC -

Related Work

❖ Comercial vTAP solutions

- IXIA, Gigamon, etc.
  - A subcomponent of their datacenter monitoring solution products
    - Vendor lock-in, maintanace and expandability problems (mainly cost)
- Different implementation approaches
  - Kernel modification, VM agent, tunneling, etc.

❖ Research

- Planck (ACM SIGCOMM 2014)
  - Accelerated packet sampling by saturating a collection port
  - Port mirroring of HW OpenFlow switch
- NFVperf (IEEE NFV-SDN 2016)
  - Detects bottleneck points of an NFV system
  - Packet collection by port mirroring of virtual switch
- DPDK Open vSwitch performance validation with mirroring feature (IEEE ICT 2016)
  - Shows port mirroring performance of Open vSwitch with DPDK

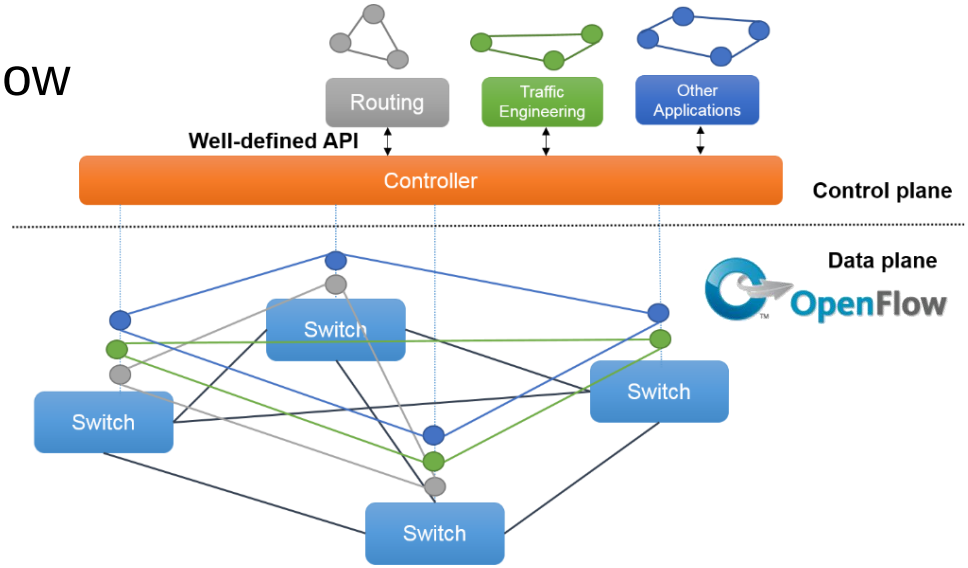
Design and Implementation

Proposed Design

❖ Main Design Concept

- SDN seperates data and control plane of a network
  - Data plane
    - Switch, router (distributed)
    - Delivers packets according to forwarding rules
  - Control plane
    - Controller, application (logically centralized)
    - Making a routing decision, collecting a global network view
- Representative protocol: OpenFlow
- vTAP design
  - Data plane
    - Packet duplication and forwarding
  - Control plane:
    - TAP policy management

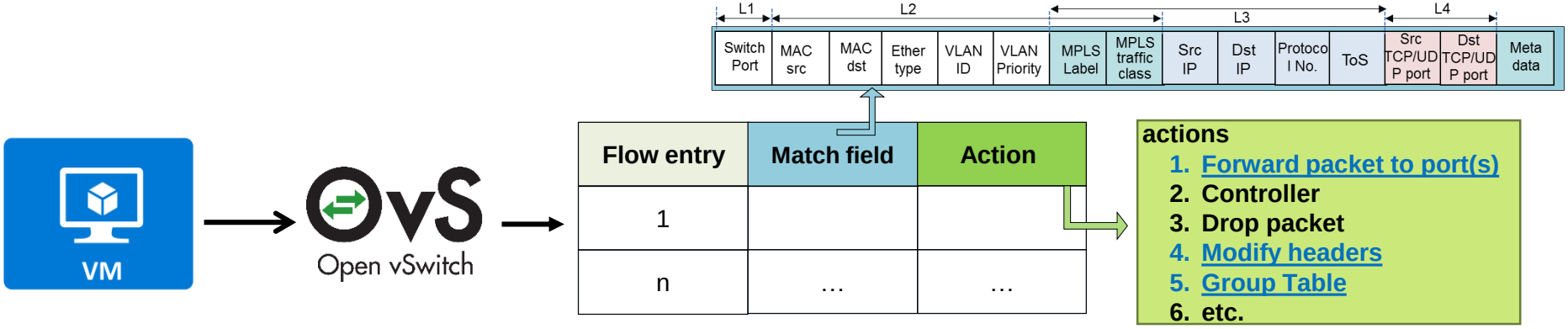
    based on a global view of a network



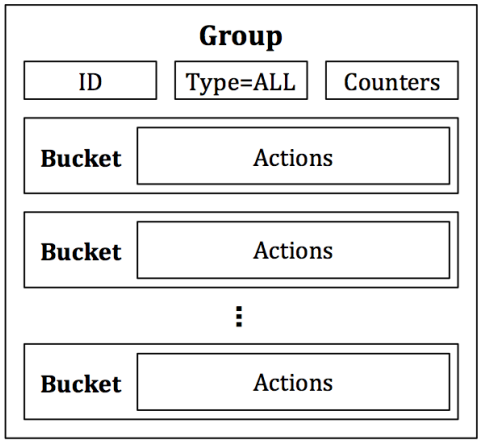
Proposed Design

❖ vTAP Data Plane Design

- Open vSwitch flow handling (OpenFlow-based)



- OpenFlow Group Table
  - Groups network flows that should be treated by the same set of actions
  - Buckets
    - Each bucket can have different actions
    - A packet is duplicated as the number of buckets



- We need packet copy and modification to the duplications for the vTAP function

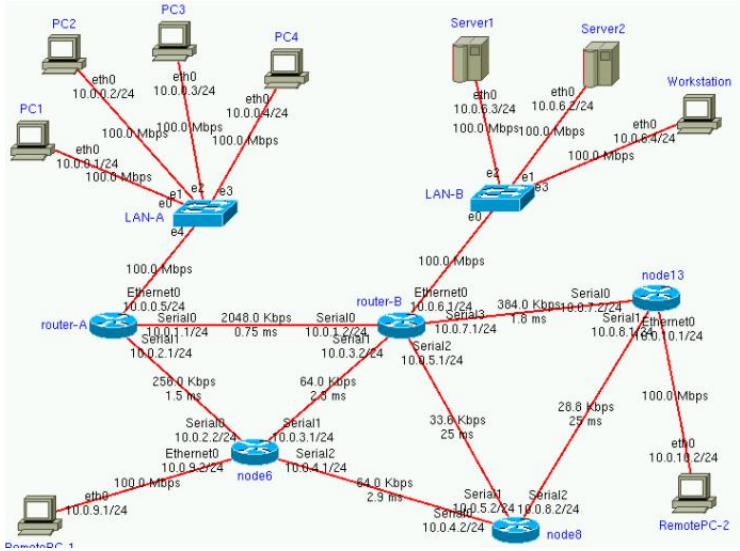
Proposed Design

❖ vTAP Control Plane Design

- Network admins can create TAP policies using a global view
  - Target flow (MAC src/dst address OR IP src/dst address)
  - Destination
  - Filtering options (TCP/UDP src/dst port number, VLAN ID, etc.)

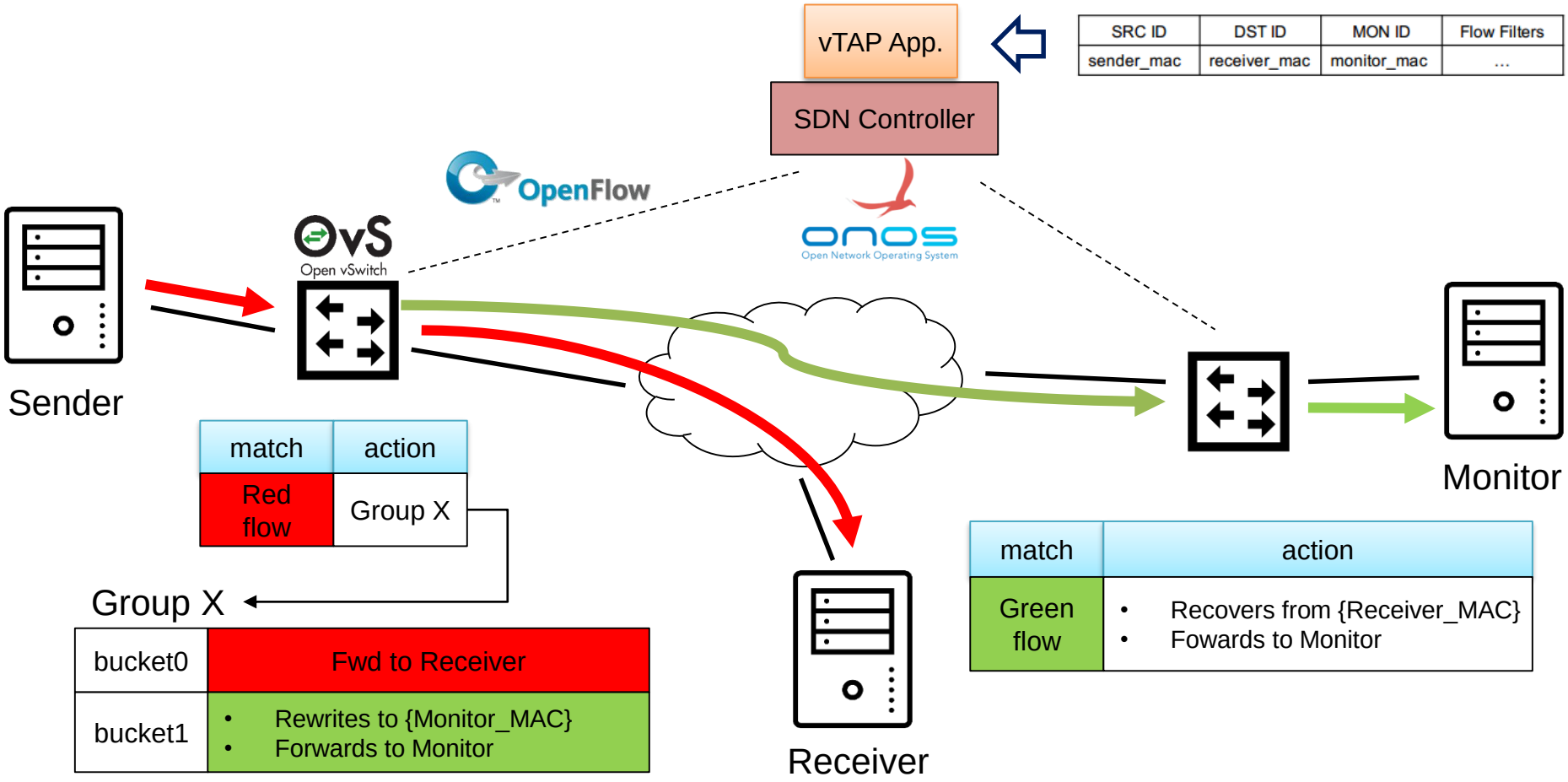
	Identifier fields						Optional fields			
	Source MAC addr.	Source IP addr.	Destination MAC addr.	Destination IP addr.	Monitor Mac addr.	Monitor IP addr.	IPv4 protocol	VLAN id	TCP/UDP Src. Port Num.	TCP/UDP Dst. Port Num.
TAP 1	00:00:00: 00:00:01	10.0.0.1	00:00:00: 00:00:02	10.0.0.2	00:00:00: 00:00:03	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	*	*

- vTAP application (backed by controller)
  - Provides TAP policy management interface
  - Translates a TAP policy into related OpenFlow messages
    - Apply the TAP policy to the data plane

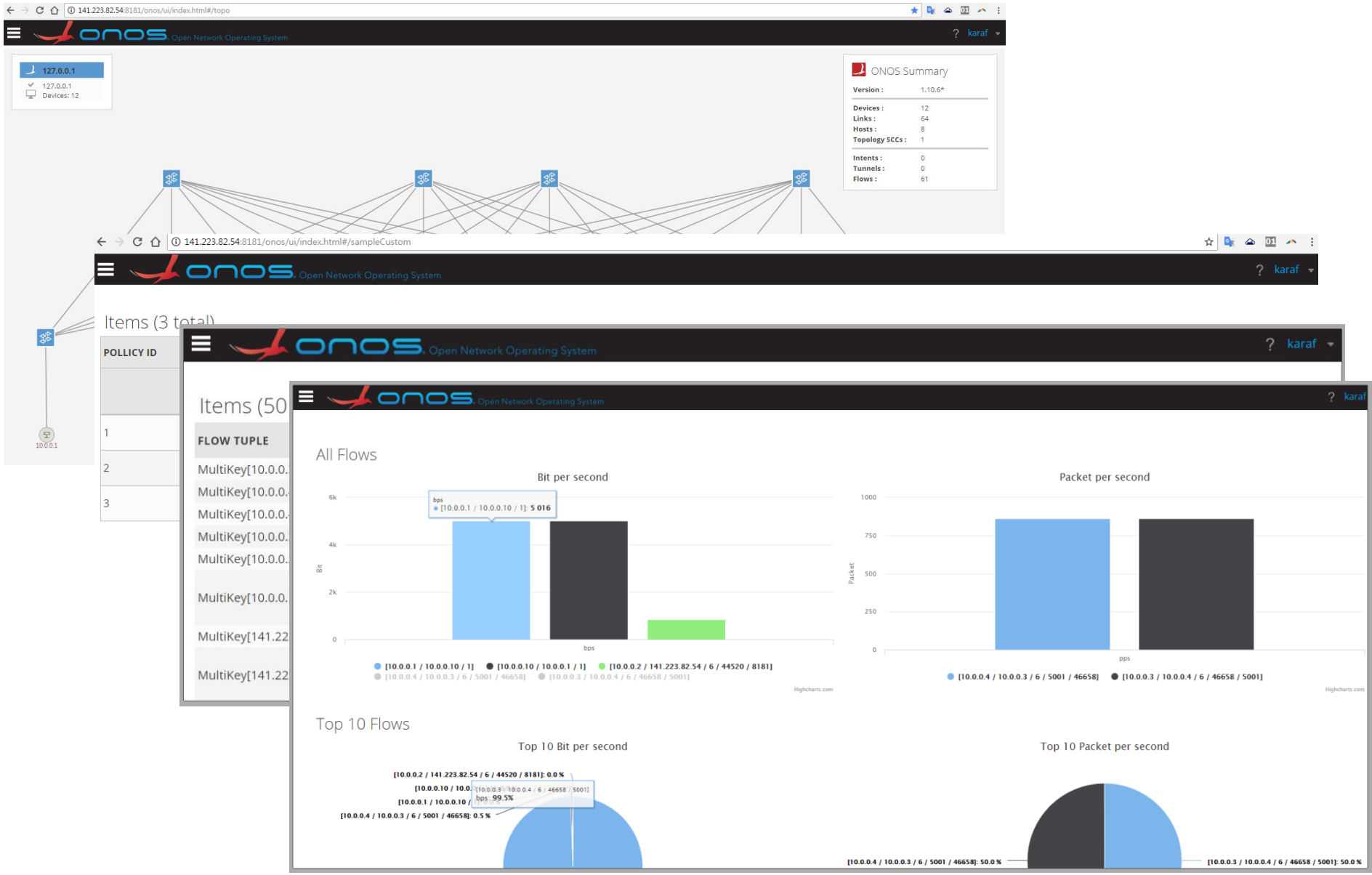


Implementation

- Control plane: ONOS (SDN controller), OpenFlow
- Data plane: Open vSwitch, DPDK (accelerate packet copy & fwd.)



Implementation



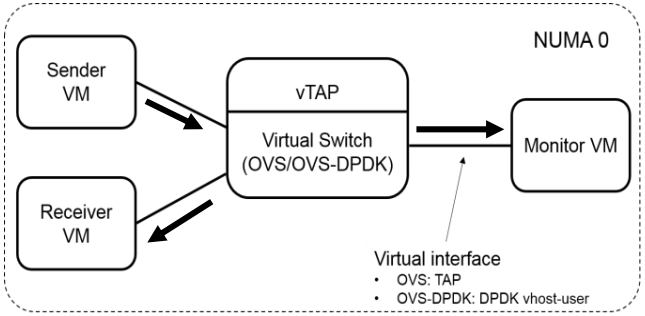
- Visualization UI -

Evaluation and Conclusion

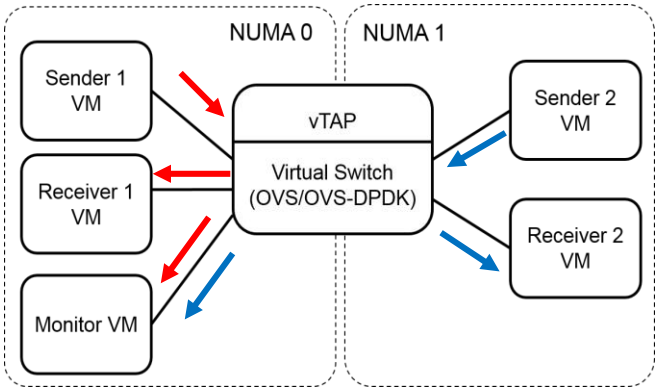
Evaluation

❖ Experiment Environment

- Hardware
  - 2.67 GHz Intel Xeon CPU and 24GB RAM
    - 6 cores * 2 NUMA nodes
- Software
  - QEMU-KVM (hypervisor)
  - Open vSwitch 2.7.2
  - DPDK 16.11
  - ONOS 1.10 (OpenFlow 1.3)
  - pktgen (packet generator)



- Testbed #1 -



- Testbed #2 -

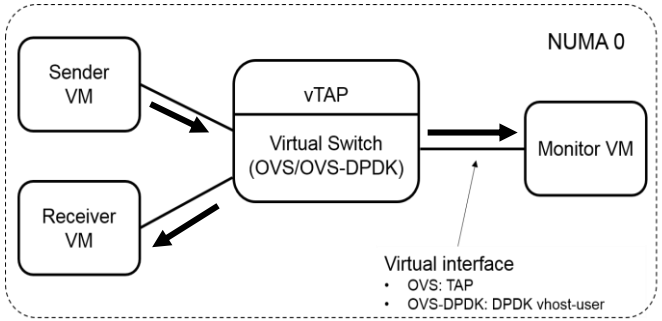
❖ Testbed setups

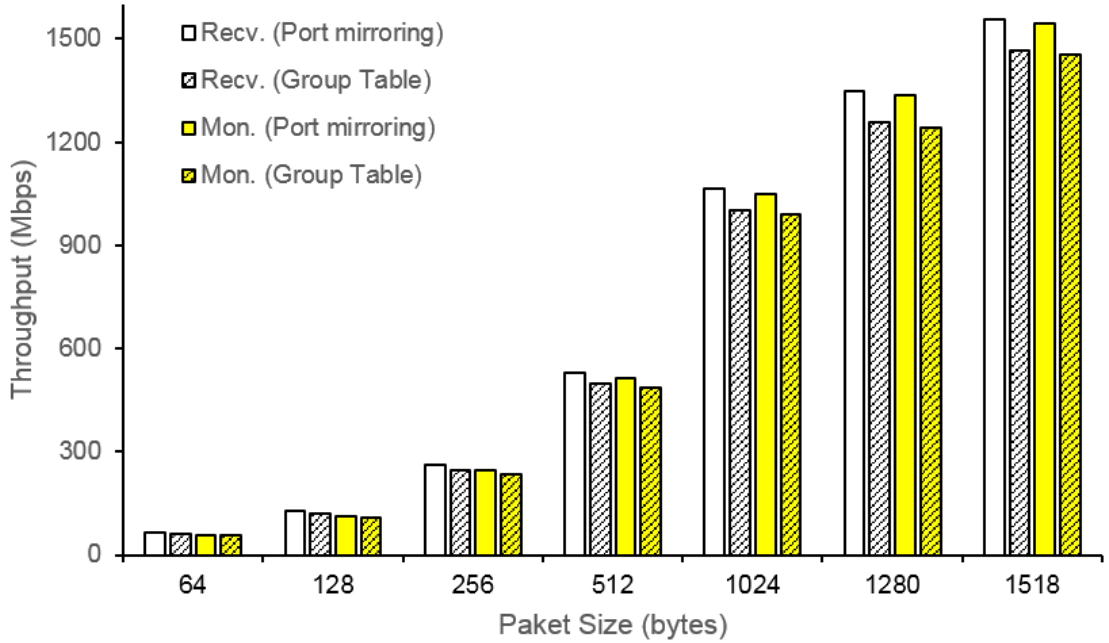
- 1 vCPU and 1GB RAM for all the VMs
- Testbed #1
  - One pair of (Sender, Receiver, Monitor)
- Testbed #2
  - Two pairs of (Sender, Receiver) and a single Monitor
  - Deploy each pair in a different NUMA node for local memory access (DPDK)

Evaluation

❖ Experiment 1

- Port mirroring vs. OF Group Table
  - Default Open vSwitch (OVS)
- No meaningful differences
  - Better flexibility and expandability for vTAP via Group Table

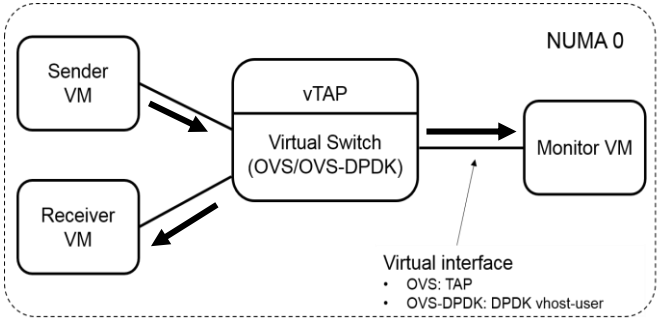


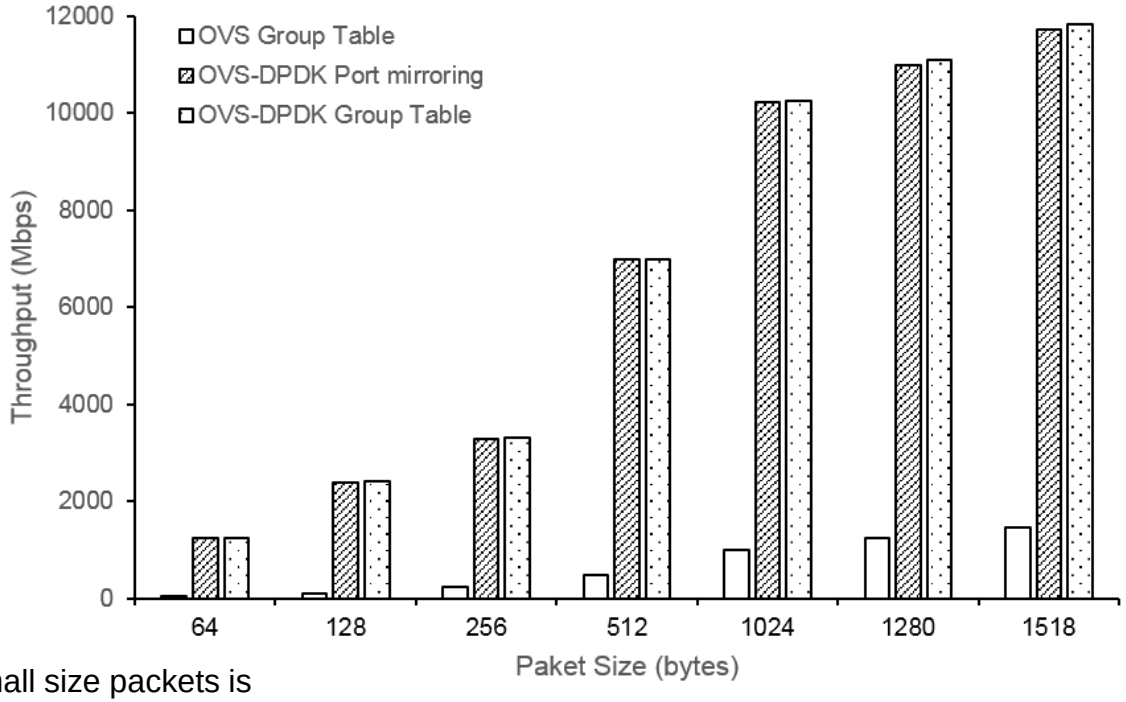


Evaluation

❖ Experiment 2

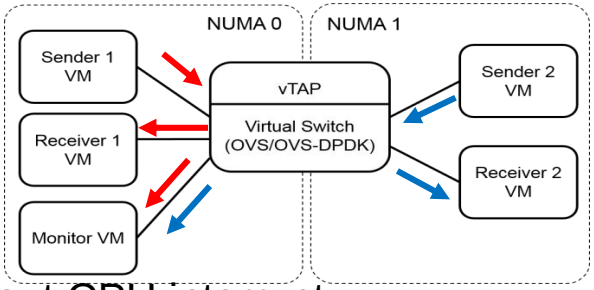
- OVS vs. OVS-DPDK for vTAP
  - Dedicated cores to OVS-DPDK
- OVS-DPDK outperforms
  - x25 increase for 64 byte packets
  - x8 increase for 1518 byte packets





*** Overall throughput for small size packets is much lower due to many packet drops in sink nodes

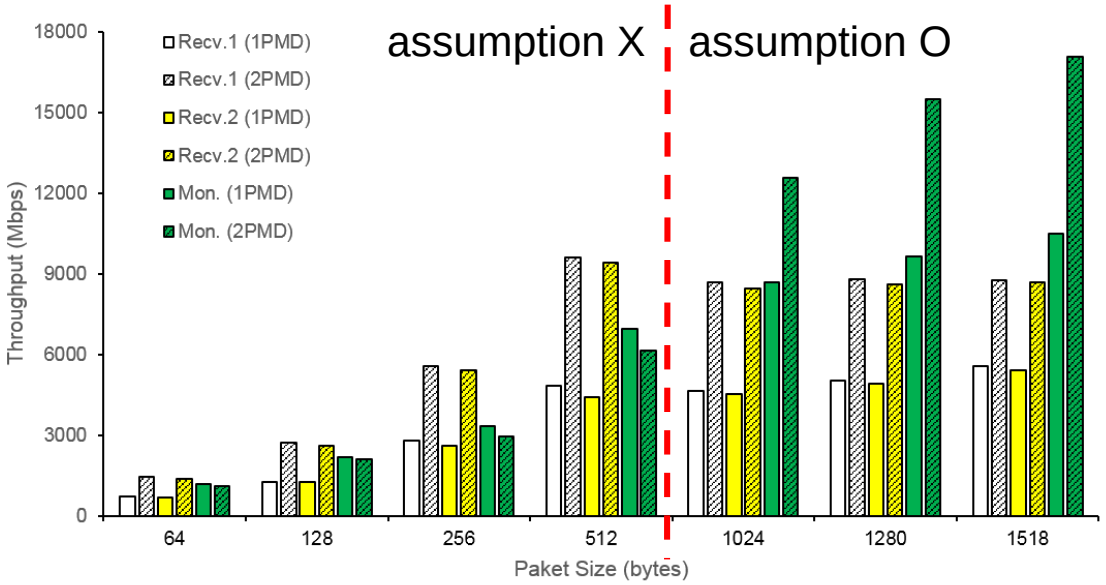
Evaluation



❖ Experiment 3

- Effect of # of DPDK PMD threads on vTAP
  - PMD (Poll Mode Driver) → direct packet fetching without CPU interrupt
  - One more PMD thread → one more CPU core
- 2 PMD shows throughput improvement
  - + 85% in Receivers, + 35% in Monitor
    - Assumption: 2 PMD shows about 2x throughput at Monitor
    - More small size packets → CPU interrupts → vCPU saturation → many packet drops

➔ To maximize vTAP performance, consider VM's resources also



Conclusion

❖ Summary

- Design and implementation of vTAP for monitoring of traffic among VMs using
  - Most popular open source virtual switch (Open vSwitch)
    → Programmability
  - Separate control and data plane concept of SDN (OpenFlow protocol)
    → Centralized management
  - Dedicated packet processing framework (DPDK, Data Plane Development Kit)
    → Performance improvement
- Performance evaluation of pack et mirroring and vTAP
  - 8~25 times throughput increase according to packet size
  - At the cost of using additional CPU cores for DPDK

❖ Future Work

- Apply our vTAP in OpenStack environment
- Consider various applications of fast packet capturing by vTAP
  - ML model for network behavior, DPI (Deep Packet Inspection), etc.

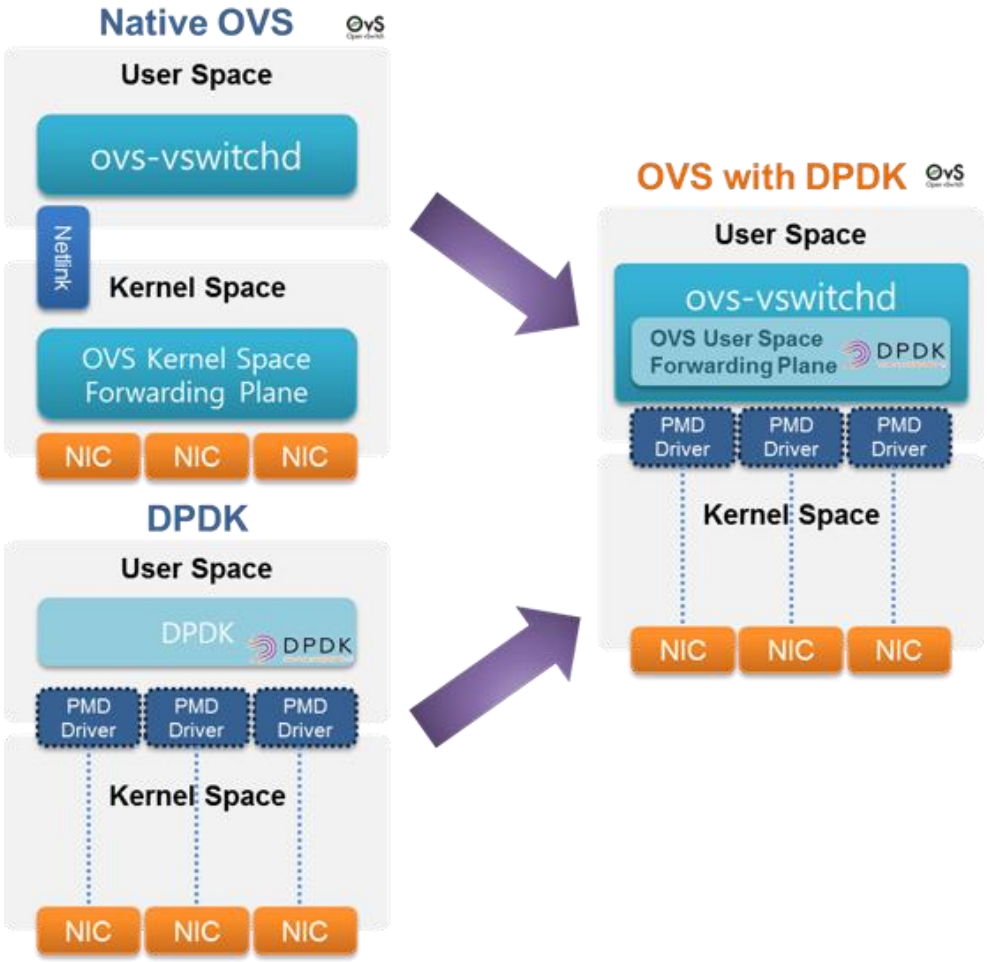
Q&A

감사합니다

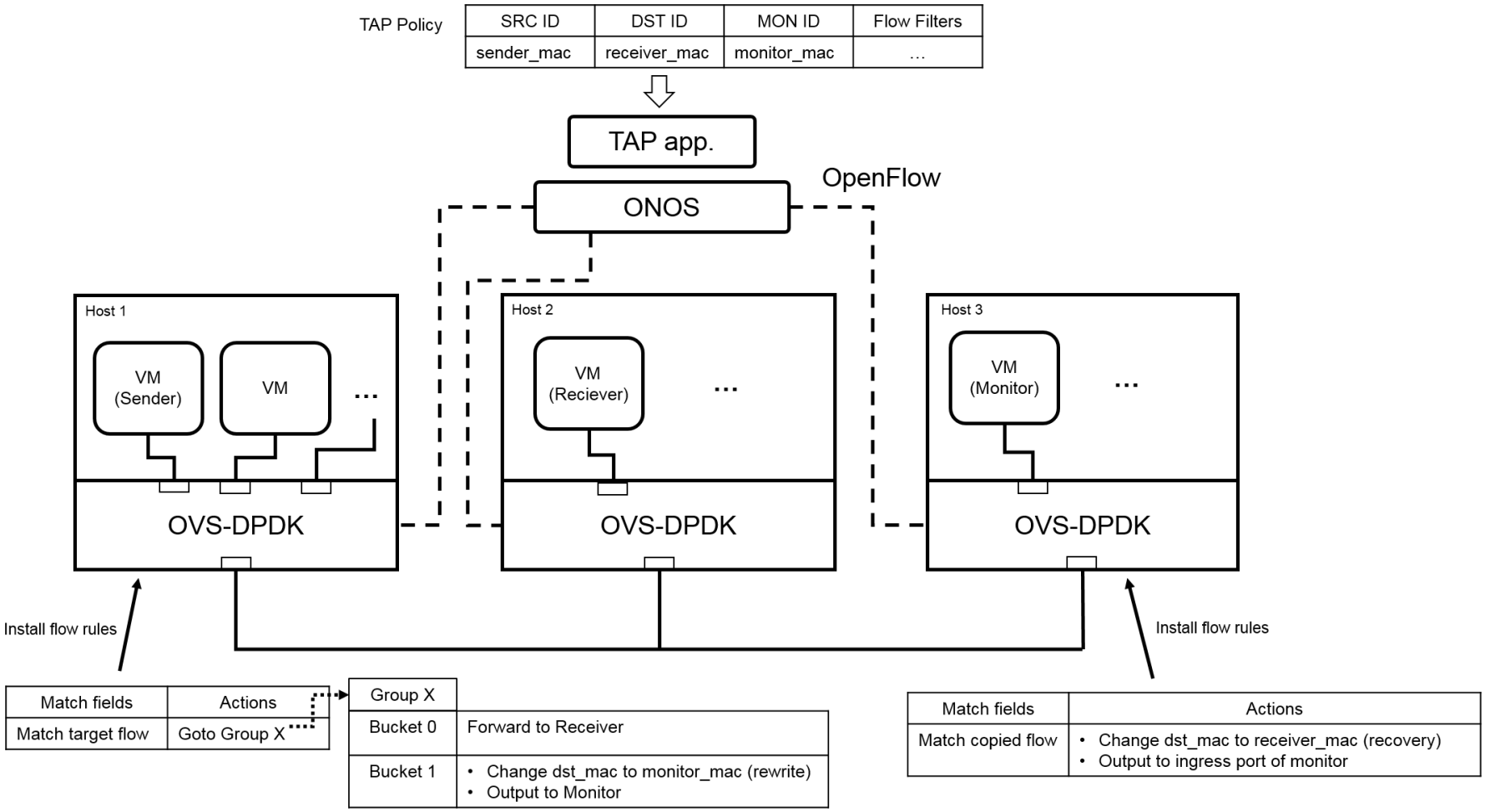
Appendix

❖ Open vSwitch with DPDK (OVS-DPDK)

- Re-implementation of OVS using DPDK features in userspace



❖ Overall Design Architecture



Appendix

❖ TAP Policy Provisioning Algorithm

Algorithm 1 : TAP Policy Apply

Require: Input (*policy*, *deviceSrv*, *hostSrv*, *topologySrv*)

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

Appendix

❖ Experiment #1, #2

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

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

Appendix

❖ Experiment #3

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

Packet size (bytes)	Receiver 1		Receiver 2		Monitor	
	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)

Packet size (bytes)	Receiver 1		Receiver 2		Monitor	
	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