



Virtual Network Isolation: Are We There Yet?

Kashyap Thimmaraju, Gábor Rétvári and Stefan Schmid

ACM SIGCOMM 2018 Workshop on Security in Softwarized Networks: Prospects and Challenges.
August 24, Budapest, Hungary



universität
wien

Multi-tenant Virtual Networks



Amazon VPC Overview Features Pricing Getting Started Resources FAQs

Amazon Virtual Private Cloud

Provision a logically isolated section of the Amazon Web Services (AWS) Cloud where you can launch AWS resources in a virtual network that you define.

Get started with Amazon VPC

Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define. You have complete control over your virtual networking environment, including selection of your own IP address range, creation of subnets, and configuration of route tables and network gateways. You can use both IPv4 and IPv6 in your VPC for secure and easy access to resources and applications.

You can easily customize the network configuration for your Amazon VPC. For example, you can create a public-facing subnet for your web servers that has access to the Internet, and place your back-end systems such as databases or application servers in a private-facing subnet with no Internet access. You can leverage multiple layers of security, including security groups and network access control lists, to help control access to Amazon EC2 instances in each subnet.

Additionally, you can create a Hardware Virtual Private Network (VPN) connection between your corporate data center and your VPC and leverage the AWS Cloud as an extension of your corporate data center.

AWS PrivateLink

Access services hosted on AWS easily and securely.

Learn about AWS PrivateLink

Benefits

SECURE

Amazon VPC provides advanced security features, such as security groups and network access control lists, to enable isolated and controlled traffic at the

SIMPLE

You can create a VPC quickly and easily using the AWS Management Console. You can select one of the common network configurations that have been used in the past

ALL THE SCALABILITY AND RELIABILITY OF AWS

Amazon VPC provides all of the same benefits as the rest of the AWS platform. You can instantly scale your resources up



Data Centre > **Virtualization**

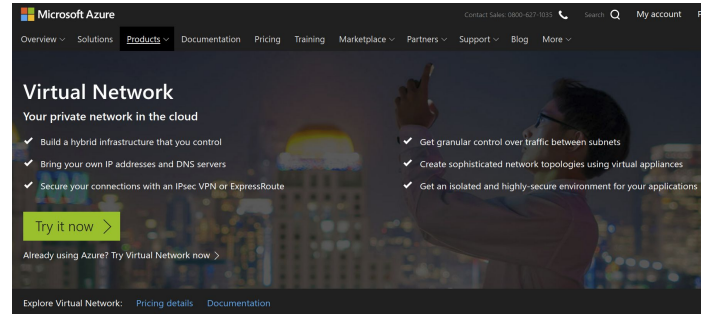
VMware preps NSX network virtualization for smaller customers

Q1 2019 beats expectations, full-year guidance raised

By Simon Sharwood 1 Jun 2018 at 02:54

2 SHARE

VMware's mainstream server virtualization users haven't felt a lot of love in recent years. The company's focussed on the cloud, containers and software, but failed to woo large enterprises.



Microsoft Azure Contact Sales 0800-627-9338 Search My account

Overview Solutions Products Documentation Pricing Training Marketplace Partners Support Blog More

Virtual Network

Your private network in the cloud

- ✓ Build a hybrid infrastructure that you control
- ✓ Bring your own IP addresses and DNS servers
- ✓ Secure your connections with an IPsec VPN or ExpressRoute
- ✓ Get granular control over traffic between subnets
- ✓ Create sophisticated network topologies using virtual appliances
- ✓ Get an isolated and highly-secure environment for your applications

Try it now >

Already using Azure? Try Virtual Network now >

Explore Virtual Network: Pricing details Documentation

Customers running Virtual Network



Enhance security and isolation

Azure Virtual Network gives you an isolated and highly-secure environment to run your virtual machines and applications. Use your private IP addresses and define subnets, access control policies, and more. Use Virtual Network to treat Azure the same as you would your own datacenter.

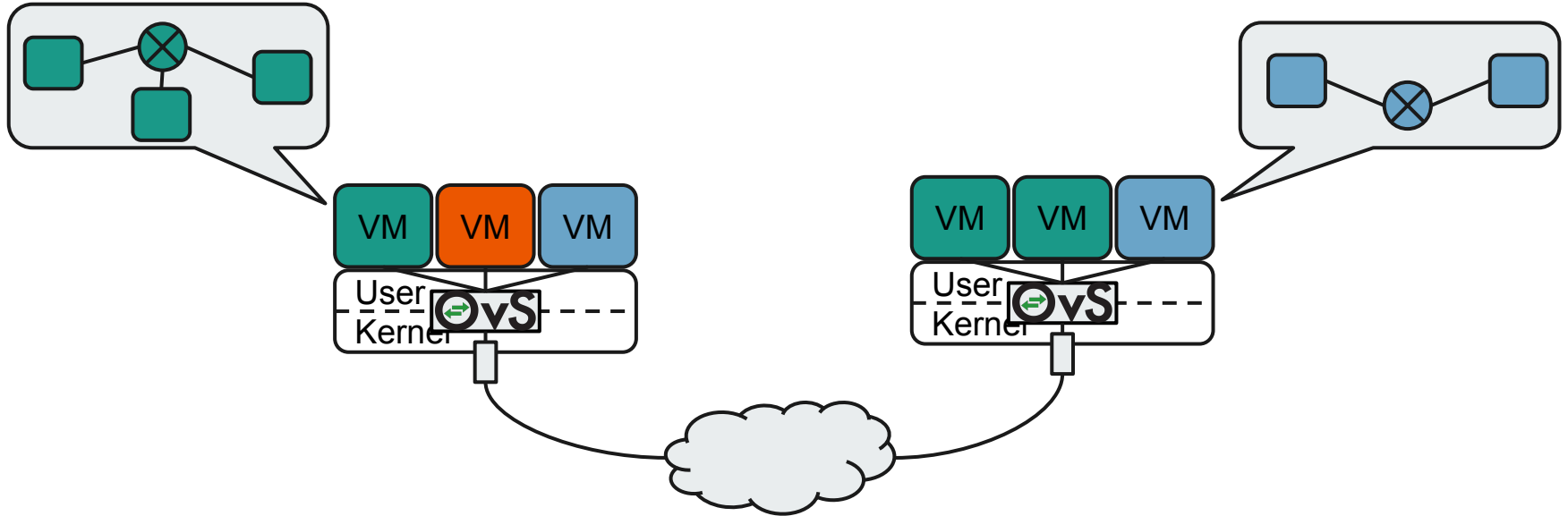


<https://aws.amazon.com/vpc/>

https://www.theregister.co.uk/2018/06/01/vmware_q1_2019/

<https://azure.microsoft.com/en-us/services/virtual-network/>

Virtual Networks in the Cloud



Virtual Switches

Andromeda: Performance, Isolation, and Velocity at Scale in Cloud Network Virtualization

Authors:

Michael Dabran, David Schultz, Jacob Adamsen, Ahsan Azeem, Anshuman Gupta, Brian Fahs, Dima Zubitskii, Enrique Cauch, Ziemens Erik Rubov, James Alexander Drouot, Jesse Aplet, Jing Ai, Jin Ohno, Kevin DeChoober, Marc de Kruif, Nathan Lewis, Nikhil Kosmudhan, Riccardo Crepaldi, Srinivas Krishnan, Subbath Venkata, Yossi Richter, Uday Nall, and Amin Vahidi, Google Inc.

Abstract:

This paper presents our design and experience with Andromeda, Google Cloud Platform's network virtualization stack. Our production deployment poses several challenging requirements, including performance isolation among customer virtual networks, scalability, rapid provisioning of large numbers of virtual hosts, bandwidth and latency largely indistinguishable from the underlying hardware, and high feature velocity combined with high availability.

Andromeda is designed around a flexible hierarchy of flow processing paths. These are mapped to a programming path dynamically based on feature and performance requirements. We introduce the Hoverboard programming model, which uses gateways for the long tail of low bandwidth flows, and enables the control plane to program network connectivity for tens of thousands of VMs in seconds. The on-host datapath is based around a high-performance OS kernel software packet processing path. CPU-intensive per packet operations with higher latency targets are executed on coprocessor threads. This architecture allows Andromeda to decouple feature growth from fast path performance, as many features can be implemented solely on the coprocessor path. We demonstrate that the Andromeda datapath achieves performance that is competitive with hardware while maintaining the flexibility and velocity of a software-based architecture.

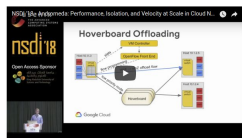
NSDI '18 Open Access Videos Sponsored by
King Abdullah University of Science and Technology (KAUST)

Open Access Media

USENIX is committed to Open Access to the research presented at our events. Papers and proceedings are freely available to everyone once the event begins. Any video, audio, and/or slides that are posted after the event are also free and open to everyone. Support USENIX and our commitment to Open Access.

Download PDF View the slides

Bit.ly/



<https://www.usenix.org/node/21124>

Virtual Networking



Accelerate Application Agility

Bring speed, security, and simplicity to today's application-centric businesses.

Learn More

Featured Products All Products Benefits Learn More

Cisco Nexus 1000V Switch for Microsoft Hyper-V

Cisco Nexus 1000V Switch for VMware vSphere

Cisco Nexus 1000V Switch for KVM

Cisco Application Virtual Switch

Cloud Networking Services

Virtual Application Cloud Segmentation

Cisco Virtual Security Gateway for Nexus 1000V Switch

Cisco Cloud Services Router 1000V Series

<https://www.cisco.com/c/en/us/products/switches/virtual-networking/index.html>

VFP: A Virtual Switch Platform for Host SDN in the Public Cloud

Authors:

Daniel Firestone, Microsoft

Abstract:

Many modern scalable cloud networking architectures rely on host networking for implementing VM network policy - e.g. tunneling for virtual networks, NAT for load balancing, stateful ACLs, QoS, and more. We present the Virtual Filtering Platform (VFP) - a programmable virtual switch that powers Microsoft Azure, a large public cloud, and provides this policy. We define several major goals for a programmable virtual switch based on our operational experiences, including support for multiple independent network controllers, policy based on connections rather than only on packets, efficient caching and classification algorithms for performance, and efficient offload of flow policy to programmable NICs, and demonstrate how VFP achieves these goals. VFP has been deployed on >1M hosts running IaaS and PaaS workloads for over 4 years. We present the design of VFP and its API, its flow language and compiler used for flow processing, performance results, and experiences deploying and using VFP in Azure over several years.

NSDI '17 Open Access Videos Sponsored by
King Abdullah University of Science and Technology (KAUST)

Open Access Media

USENIX is committed to Open Access to the research presented at our events. Papers and proceedings are freely available to everyone once the event begins. Any video, audio, and/or slides that are posted after the event are also free and open to everyone. Support USENIX and our commitment to Open Access.

Presentation Video



<https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/firestone>

Virtual Switches (Non-Exhaustive List)

Name	Ref.	Year	Emphasis	Co-Location Kernel	User	Ext. Parsing	Comments
OvS	[22]	2009	Flexibility	✓	✓	✓	Baseline
Cisco NexusV	[36]	2009	Flexibility	✓	✓	✗	Commercial
VMware vSwitch	[37]	2009	Centralized control	✓	✓	✗	Commercial
Vale	[25]	2012	Performance	✓	✓	✗	Using HPFP to increase performance
Research prototype	[14]	2012	Isolation	✗	✓	✓	Place OvS in a VM [14].
Hyper-Switch	[24]	2013	Performance	✓	✓	✓	Fast path in the Xen hypervisor
MS HyperV-Switch	[18]	2013	Centralized control	✓	✓	✗	Commercial
NetVM	[13]	2014	Performance, NFV	✓	✗	✓	Using HPFP to increase performance.
sv3	[31]	2014	Security	✓	✗	?	Can run multiple sv3 switches on the Host, isolation via processes.
fd.io	[32]	2015	Performance	✓	✗	✓	Uses Vector Packet Processing, e.g., see Choi et al. [6].
mSwitch	[12]	2015	Performance	✓	✓	✗	Using HPFP to increase performance.
BESS	[4]	2015	Programmability, NFV	✓	✗	✓	Similar to the Click modular router [15].
PISCES	[29]	2016	Programmability	✓	✓	✓	Uses a domain specific language to customize parsing.
OvS with DPDK	[26]	2016	Performance	✓	✗	✓	Using HPFP for performance; sw. countermeasures, e.g., canaries and ASLR may not be used.
ESwitch	[19]	2016	Performance	✓	✗	✓	Proprietary.
MS VFP	[8]	2017	Performance, flexibility	✓	✓	✓	Commercial.
Mellanox BlueField	[17]	2017	CPU offload	✗	✓	✓	Runs full fledged OvS on CPU in NIC. Server leased, but provider controls the network.
Liquid IO	[21]	2017	CPU offload	✗	✓	✓	Runs full fledged OvS on CPU in NIC.
Stingray	[10]	2017	CPU offload	✗	✓	✓	Runs full fledged OvS on CPU in NIC.
GPU-based OvS	[35]	2017	Acceleration	✓	✓	✓	Leverages the GPU for packet processing.
MS AccelNet	[9]	2018	Performance, flexibility	✓	✓	✗	Packet processing and flow rules offloaded to an FPGA-based NIC.
Google Andromeda	[7]	2018	Flexibility and performance	✓	✗	✓	OvS-based software switch with hardware offloads.

Screenshot from the paper.

Taking Control of SDN-based Cloud Systems via the Data Plane

Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann and Stefan Schmid

SOSR'18

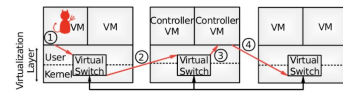


Figure 2: An overview of the security implications of current virtual switch designs.

trusted components, are commonly not protected with an additional intrusion detection system.

Unified packet parser: Once a virtual switch receives a packet it parses its headers to determine if it already has a matching flow rule. If this is not the case it will forward the packet to an intermediate data path (slow path) that processes the packet further in order to request a new flow table entry. In this step, the virtual switch commonly extracts all header information from the packet, e.g., MPLS and application layer information, before requesting a flow table entry from the controller. Parsing is the switch’s responsibility as centralizing this task would not scale. The additional information from higher-level protocols is needed for advanced functionality like load balancing, deep packet inspection (DPI), and non-standard forwarding (see Section 5 for an overview of related technologies using these features in their implementation). However, with *protocol parsing in the data plane* the virtual switch is as susceptible to security vulnerabilities as any daemon for the parsed protocol. Thus, the attack surface of the data plane increases with any new protocol that is included in parsing.

Untrusted input: Virtual switches are commonly deployed in data centers at the network edge. This implies that virtual switches receive network packets directly from the virtual machines, typically unfiltered, see Section 2. This can be abused by an attacker. She can—via a virtual machine—send arbitrary data to a virtual switch¹. Indeed, the virtual switch is typically the *first* data plane component to handle any packet from a VM. This enables attackers to take advantage of data plane vulnerabilities in virtual switches.

Summary: In combination, the above observations demonstrate why data plane attacks are a *feasible* threat and how they can spread throughout a cloud setup, see Fig. 2. By renting a VM and weaponizing a protocol parsing vulnerability an attacker can start her attack by taking over a single virtual switch (Step 1). Thus, she also takes control of the physical machine on which the virtual switch is running due to hypervisor co-location. Next (Step 2), she can take control of the Host OS where the VM running the network—and in most

¹Depending on the implementation, the *Dom0* IP stack may ensure that the IP part of all packets are well-formed.

cases cloud-controller is hosted due to the direct communication channel. From the controller (Step 3), the attacker can leverage the logically centralized design to, e.g., manipulate flow rules to violate essential network security policies (Step 4). Alternatively, the attacker can change other cloud resources, e.g., modify the identity management service or change a boot image for VMs to contain a backdoor.

3.2 Attacker Models for Virtual Switches

With these vulnerabilities and attack surfaces in mind, we revisit existing threat models. We particularly focus on work starting from 2009 when virtual switches emerged into the virtualization market [63]. We find that virtual switches are not appropriately accounted for in existing threat models, which motivates us to subsequently introduce a new attacker model.

Existing threat models: Virtual switches intersect with several areas of network security research: Data plane, network virtualization, software defined networking (SDN), and the cloud. Therefore, we conducted a qualitative analysis that includes research we identified as relevant to attacker models for virtual switches in the cloud. In the following we elaborate on that.

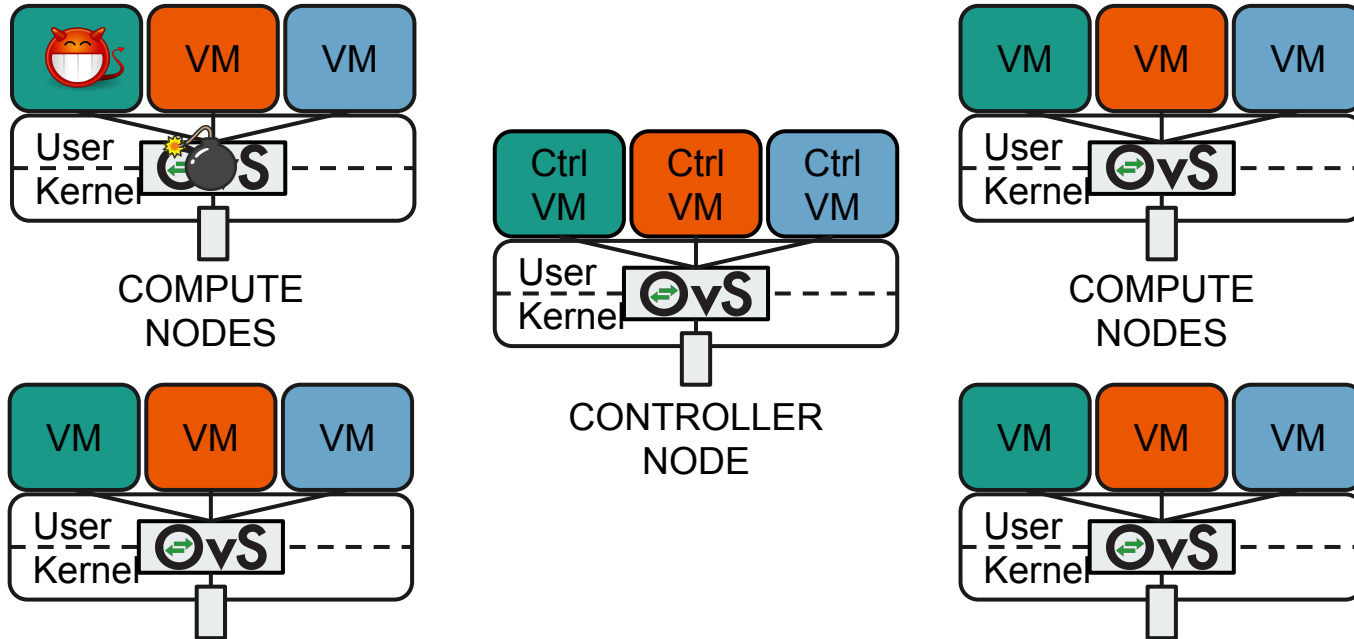
Qubes OS [78] in general assumes that the networking stack can be compromised. Similarly, Dhawan et al. [20] assumed that the Software Defined Network (SDN) data plane can be compromised. Jero et al. [36] base their assumption on a malicious data plane in an SDN on Pickett’s BlackHat briefing [65] on compromising an SDN hardware switch.

A conservative attacker model was assumed by Paladi et al. [55] who employ the Dolev-Yao model for network virtualization in a multi-tenant cloud. Grobauer et al. [28] observed that virtual networking can be attacked in the cloud without a specific attacker model.

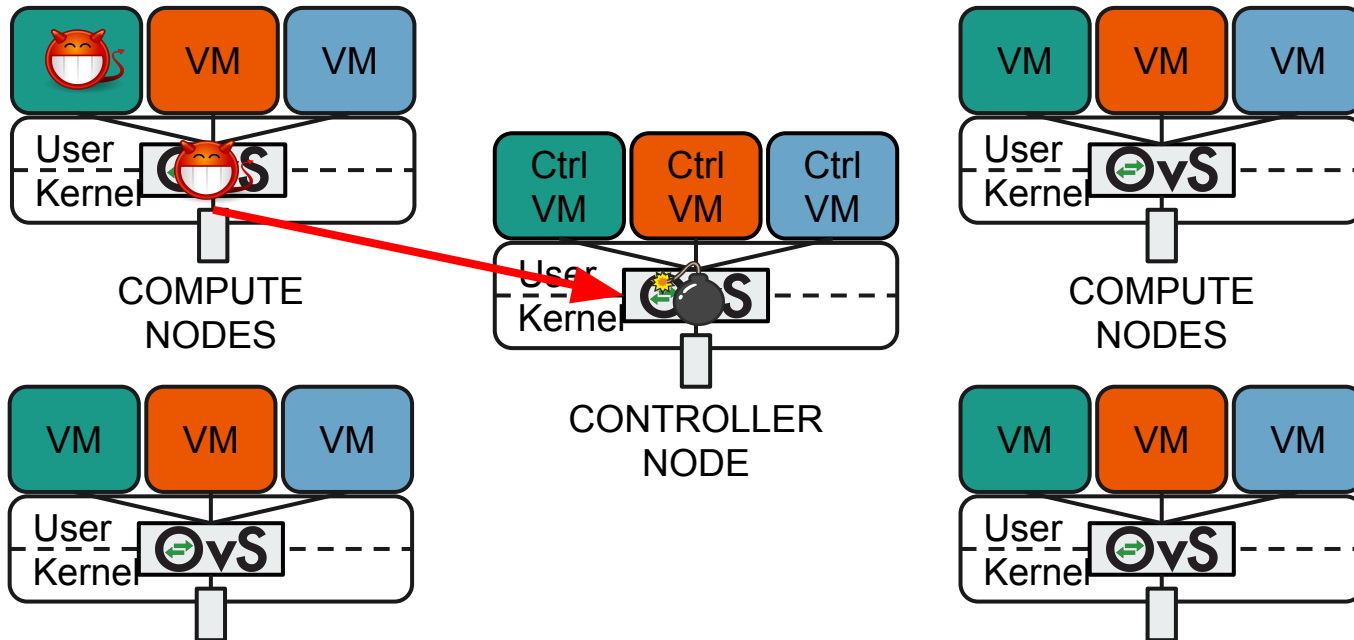
Jin et al. [37] accurately described two threats to virtual switches: Virtual switches are co-located with the hypervisor; and guest VMs need to interact with the hypervisor. However, they stopped short of providing a concrete threat model, and underestimated the impact of compromising virtual switches. Indeed at the time, cloud systems were burgeoning. However, only recently Alhebaishi et al. [9] proposed an updated approach to cloud threat modelling wherein the virtual switch was identified as a component of cloud systems that needs to be protected. However, in both cases, the authors overlooked the severity, and multitude of threats that apply to virtual switches.

Motivated by a strong adversary, Gonzales et al. [22], and Karmakar et al. [40] accounted for virtual switches, and the data plane. Similarly Yu et al. [97], Thimmaraju et al. [90] and Feldmann et al. [24] assumed a strong adversarial model,

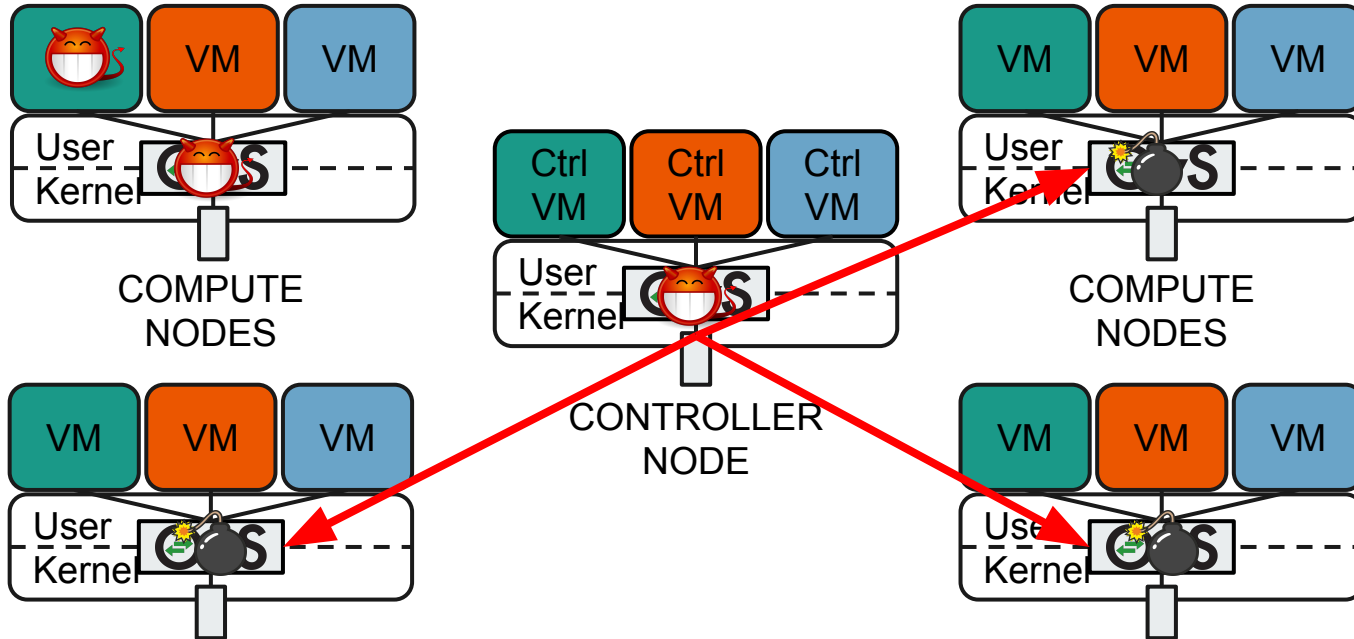
Compromising the Cloud



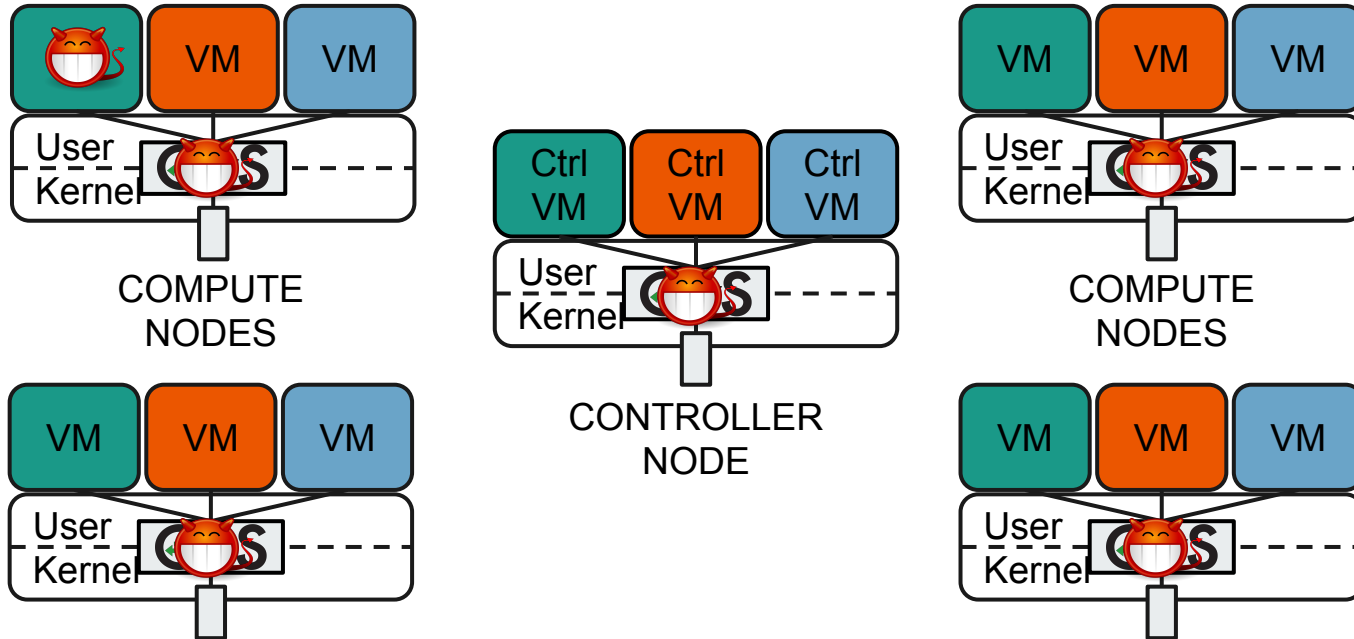
Compromising the Cloud



Compromising the Cloud



Compromising the Cloud





Taking Control of SDN-based Cloud Systems via the Data Plane

Kashyap Thimmaraju, Bhargava Shastry, Tobias Fiebig, Felicitas Hetzelt, Jean-Pierre Seifert, Anja Feldmann and Stefan Schmid

SOSR'18

there is no impact of the user-land protection mechanisms in the fast path, see Fig. 5b.

Throughput Evaluation: For the throughput evaluation we use a constant stream of packets replayed at a specific rate. We opted for small packets to focus on the packets per second (pps) throughput rather than the bytes per second throughput. Indeed, pps throughput indicates performance bottlenecks earlier [34] than bytes per second. As in the latency experiments, we opted to use packets that are 60B long. Each experimental run lasts for 1000 seconds and uses a specific replay rate. Then we reset the system and start with the next replay rate. Our evaluation focuses on the last 900 seconds. For the slow path, the replay rates start from 10k to 40k packets per second, in steps of 1k pps. For the fast path, the replay rates start from 300k to 900k packets per second, in steps of 10k pps. For better readability we show the slow path plot from 10k to 35k pps.

An overview of the results for the slow and fast path throughput measurements are depicted in Figures 5c and 5d resp. In the slow path, packet loss for the vanilla kernel first sets in just after 18k pps, while the experiments on the grsecurity enabled kernel already exhibit packet loss at 14k pps. In the fast path, grsec exhibits packet loss from 350k pps whereas the vanilla kernel starts to drop packets at 690k pps. Hence, we note that the grsecurity kernel patch does have a measurable impact on the forwarding throughput in the slow and fast path of OvS. With respect to the user-land security features, we observe an overhead only in the slow path of approximately 4-15%.

Summary: Our measurements demonstrate that user-land mitigations do not have a large impact on OvS's forwarding performance. However, grsecurity kernel patches do cause a performance overhead for latency as well as throughput. Given that cloud systems support a variety of workloads, e.g., low latency or high throughput, kernel-based mitigations may or may not be used. However, cloud systems such as the one studied by Pfaff et al. [64] can adopt the user-land and kernel software mitigations described in this paper.

It is only a question of time until the next wormable vulnerability in a virtual switch is discovered. As software mitigations can be more easily deployed than a fully re-designed virtual switch ecosystem, we strongly recommend the adoption of software countermeasures, until a more securely designed virtual switch platform can be rolled out.

Moreover, our security analysis underlines the need for networking researchers to include software countermeasures in their design, implementation, and evaluation of novel networking components. As indicated by our analysis of

7 DESIGN COUNTERMEASURES

Specific attacks against virtual switches may be prevented by software countermeasures. However, the underlying problems of co-location and a worm-friendly system design remain. Hence, in this section, we present mitigation strategies that detect, isolate, and prevent the spread of attacks via the data plane and thus reduce the attack surface we identified. We do so not only for cloud based systems and OvS but also in the more general context of SDN.

Virtualized/Isolated data plane: One essential feature of the identified attack surface is the co-location of data plane and hypervisor (see Section 3). Addressing this problem in OpenStack is non-trivial due to the sheer number of interacting components and possible configurations, e.g., virtualized/non-virtualized, integrated/distributed, redundant/merging, virtual controllers [69].

One way to design a system with stronger separation is to virtualize the data plane components, thereby decoupling it from the virtualization layer. For virtual switches one example of such a proposal is to shift the position of the virtual switch from the host to a dedicated guest as proposed by Jin et al. [37]. However, the IOMMU of the host must be used to restrict access of the network cards to the network interfaces. Otherwise the physical host and the operating system running there are left vulnerable to direct memory access (DMA) attacks [86]. Such a design reduces the host OS's Trusted Computing Base (TCB) and, thereby, the attack surface of the virtual switch. We note that Arrakis [59] and IX [12] are promising proposals for HPFFs that would allow for designing such a system. Note, that while Arrakis utilizes the IOMMU, the authors of IX left this for further work.

Furthermore, to reduce the attack surface of hypervisors, Szefer et al. [87] suggest that the hypervisor should disengage itself from guest VMs, and the VM should receive direct access to the hardware (e.g., NIC). In conjunction with our suggestion of transferring the virtual switch into a virtual machine, the approach of Szefer et al. results in a more secure data plane that can no longer attack the hypervisor.

Control plane communication firewalls: Another method to contain and prevent attacks like the worm is tight firewalling of the control plane. In contrast to "normal" Internet traffic, control plane traffic has characteristics that enable a tighter and more secure firewall design: (i) The control plane *traffic volume* should be *significantly smaller* than regular network traffic. (ii) Nodes should only *communicate via the controller and not among each other*. Hence, there is a *central location* for the firewall. (iii) On the control channel



Outline

- Motivation
- Security Landscape of Virtual Switches
- Secure Virtual Switch Design
- Conclusion
- Discussion
 - Performance Evaluation

Security Landscape of Virtual Switches

Name	Ref.	Year	Emphasis	Co-Location	Kernel	User	Ext. Parsing	Comments
OvS	[22]	2009	Flexibility	✓	✓	✓	✓	Baseline
Cisco NexusV	[36]	2009	Flexibility	✓	✓	✗	?	Commercial
VMware vSwitch	[37]	2009	Centralized control	✓	✓	✗	✗	Commercial
Vale	[25]	2012	Performance	✓	✓	✗	✗	Using HPPF to increase performance
Research prototype	[14]	2012	Isolation	✗	✓	✓	✓	Place OvS in a VM [14].
Hyper-Switch	[24]	2013	Performance	✓	✓	✓	✓	Fast path in the Xen hypervisor
MS HyperV-Switch	[18]	2013	Centralized control	✓	✓	✗	?	Commercial
NetVM	[13]	2014	Performance, NFV	✓	✗	✓	?	Using HPPF to increase performance.
sv3	[31]	2014	Security	✓	✗	✓	?	Can run multiple sv3 switches on the Host, isolation via processes.
fd.io	[32]	2015	Performance	✓	✗	✓	✗	Uses Vector Packet Processing, e.g., see Choi et al. [6].
mSwitch	[12]	2015	Performance	✓	✓	✗	✓	Using HPPF to increase performance.
BESS	[4]	2015	Programmability, NFV	✓	✗	✓	✗	Similar to the Click modular router [15].
PISCES	[29]	2016	Programmability	✓	✓	✓	✗	Uses a domain specific language to customize parsing.
OvS with DPDK	[26]	2016	Performance	✓	✗	✓	✓	Using HPPF for performance; sw. countermeasures, e.g., canaries and ASLR may not be used.
ESwitch	[19]	2016	Performance	✓	✗	✓	✗	Proprietary.
MS VFP	[8]	2017	Performance, flexibility	✓	✓	✗	✓	Commercial.
Mellanox BlueField	[17]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC. Server leased, but provider controls the network.
Liquid IO	[21]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC.
Stingray	[10]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC.
GPU-based OvS	[35]	2017	Acceleration	✓	✓	✓	✓	Leverages the GPU for packet processing.
MS AccelNet	[9]	2018	Performance, flexibility	✓	✓	✗	✓	Packet processing and flow rules offloaded to an FPGA-based NIC.
Google Andromeda	[7]	2018	Flexibility and performance	✓	✗	✓	✓	OvS-based software switch with hardware offloads.

A security analysis of a non exhaustive list of virtual switches
(screenshot from the paper).

Name	Ref.	Year	Emphasis	Co-Location	Kernel	User	Ext. Parsing	Comments
OvS	[22]	2009	Flexibility	✓	✓	✓	✓	Baseline
Cisco NexusV	[36]	2009	Flexibility	✓	✓	✗	?	Commercial
VMware vSwitch	[37]	2009	Centralized control	✓	✓	✗	✗	Commercial
Vale	[25]	2012	Performance	✓	✓	✗	✗	Using HPPF to increase performance
Research prototype	[14]	2012	Isolation	✗	✓	✓	✓	Place OvS in a VM [14].
Hyper-Switch	[24]	2013	Performance	✓	✓	✓	✓	Fast path in the Xen hypervisor
MS HyperV-Switch	[18]	2013	Centralized control	✓	✓	✗	?	Commercial
NetVM	[13]	2014	Performance, NFV	✓	✗	✓	?	Using HPPF to increase performance.
sv3	[31]	2014	Security	✓	✗	✓	?	Can run multiple sv3 switches on the Host, isolation via processes.
fd.io	[32]	2015	Performance	✓	✗	✓	✗	Uses Vector Packet Processing, e.g., see Choi et al. [6].
mSwitch	[12]	2015	Performance	✓	✓	✗	✓	Using HPPF to increase performance.
BESS	[4]	2015	Programmability, NFV	✓	✗	✓	✗	Similar to the Click modular router [15].
PISCES	[29]	2016	Programmability	✓	✓	✓	✗	Uses a domain specific language to customize parsing.
OvS with DPDK	[26]	2016	Performance	✓	✗	✓	✓	Using HPPF for performance; sw. countermeasures, e.g., canaries and ASLR may not be used.
ESwitch	[19]	2016	Performance	✓	✗	✓	✗	Proprietary.
MS VFP	[8]	2017	Performance, flexibility	✓	✓	✗	✓	Commercial.
Mellanox BlueField	[17]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC. Server leased, but provider controls the network.
Liquid IO	[21]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC.
Stingray	[10]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC.
GPU-based OvS	[35]	2017	Acceleration	✓	✓	✓	✓	Leverages the GPU for packet processing.
MS AccelNet	[9]	2018	Performance, flexibility	✓	✓	✗	✓	Packet processing and flow rules offloaded to an FPGA-based NIC.
Google Andromeda	[7]	2018	Flexibility and performance	✓	✗	✓	✓	OvS-based software switch with hardware offloads.

A security analysis of a non exhaustive list of virtual switches.
Preliminary work on isolation.

Name	Ref.	Year	Emphasis	Co-Location	Kernel	User	Ext. Parsing	Comments
OvS	[22]	2009	Flexibility	✓	✓	✓	✓	Baseline
Cisco NexusV	[36]	2009	Flexibility	✓	✓	✗	?	Commercial
VMware vSwitch	[37]	2009	Centralized control	✓	✓	✗	✗	Commercial
Vale	[25]	2012	Performance	✓	✓	✗	✗	Using HPPF to increase performance
Research prototype	[14]	2012	Isolation	✗	✓	✓	✓	Place OvS in a VM [14].
Hyper-Switch	[24]	2013	Performance	✓	✓	✓	✓	Fast path in the Xen hypervisor
MS HyperV-Switch	[18]	2013	Centralized control	✓	✓	✗	?	Commercial
NetVM	[13]	2014	Performance, NFV	✓	✗	✓	?	Using HPPF to increase performance.
sv3	[31]	2014	Security	✓	✗	✓	?	Can run multiple sv3 switches on the Host, isolation via processes.
fd.io	[32]	2015	Performance	✓	✗	✓	✗	Uses Vector Packet Processing, e.g., see Choi et al. [6].
mSwitch	[12]	2015	Performance	✓	✓	✗	✓	Using HPPF to increase performance.
BESS	[4]	2015	Programmability, NFV	✓	✗	✓	✗	Similar to the Click modular router [15].
PISCES	[29]	2016	Programmability	✓	✓	✓	✗	Uses a domain specific language to customize parsing.
OvS with DPDK	[26]	2016	Performance	✓	✗	✓	✓	Using HPPF for performance; sw. countermeasures, e.g., canaries and ASLR may not be used.
ESwitch	[19]	2016	Performance	✓	✗	✓	✗	Proprietary.
MS VFP	[8]	2017	Performance, flexibility	✓	✓	✗	✓	Commercial.
Mellanox BlueField	[17]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC. Server leased, but provider controls the network.
Liquid IO	[21]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC.
Stingray	[10]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC.
GPU-based OvS	[35]	2017	Acceleration	✓	✓	✓	✓	Leverages the GPU for packet processing.
MS AccelNet	[9]	2018	Performance, flexibility	✓	✓	✗	✓	Packet processing and flow rules offloaded to an FPGA-based NIC.
Google Andromeda	[7]	2018	Flexibility and performance	✓	✓	✓	✓	OvS-based software switch with hardware offloads.

A security analysis of a non exhaustive list of virtual switches.
New NICs offering isolation.

Name	Ref.	Year	Emphasis	Co-Location	Kernel	User	Ext. Parsing	Comments
OvS	[22]	2009	Flexibility	✓	✓	✓	✓	Baseline
Cisco NexusV	[36]	2009	Flexibility	✓	✓	✗	?	Commercial
VMware vSwitch	[37]	2009	Centralized control	✓	✓	✗	✗	Commercial
Vale	[25]	2012	Performance	✓	✓	✗	✗	Using HPPF to increase performance
Research prototype	[14]	2012	Isolation	✗	✓	✓	✓	Place OvS in a VM [14].
Hyper-Switch	[24]	2013	Performance	✓	✓	✓	✓	Fast path in the Xen hypervisor
MS HyperV-Switch	[18]	2013	Centralized control	✓	✓	✗	?	Commercial
NetVM	[13]	2014	Performance, NFV	✓	✗	✓	?	Using HPPF to increase performance.
sv3	[31]	2014	Security	✓	✗	✓	?	Can run multiple sv3 switches on the Host, isolation via processes.
fd.io	[32]	2015	Performance	✓	✗	✓	✗	Uses Vector Packet Processing, e.g., see Choi et al. [6].
mSwitch	[12]	2015	Performance	✓	✓	✗	✓	Using HPPF to increase performance.
BESS	[4]	2015	Programmability, NFV	✓	✗	✓	✗	Similar to the Click modular router [15].
PISCES	[29]	2016	Programmability	✓	✓	✓	✗	Uses a domain specific language to customize parsing.
OvS with DPDK	[26]	2016	Performance	✓	✗	✓	✓	Using HPPF for performance; sw. countermeasures, e.g., canaries and ASLR may not be used.
ESwitch	[19]	2016	Performance	✓	✗	✓	✗	Proprietary.
MS VFP	[8]	2017	Performance, flexibility	✓	✓	✗	✓	Commercial.
Mellanox BlueField	[17]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC. Server leased, but provider controls the network.
Liquid IO	[21]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC.
Stingray	[10]	2017	CPU offload	✗	✓	✓	✓	Runs full fledged OvS on CPU in NIC.
GPU-based OvS	[35]	2017	Acceleration	✓	✓	✓	✓	Leverages the GPU for packet processing.
MS AccelNet	[9]	2018	Performance, flexibility	✓	✓	✗	✓	Packet processing and flow rules offloaded to an FPGA-based NIC.
Google Andromeda	[7]	2018	Flexibility and performance	✓	✗	✓	✓	OvS-based software switch with hardware offloads.

A security analysis of a non exhaustive list of virtual switches.
Least privilege packet processing.



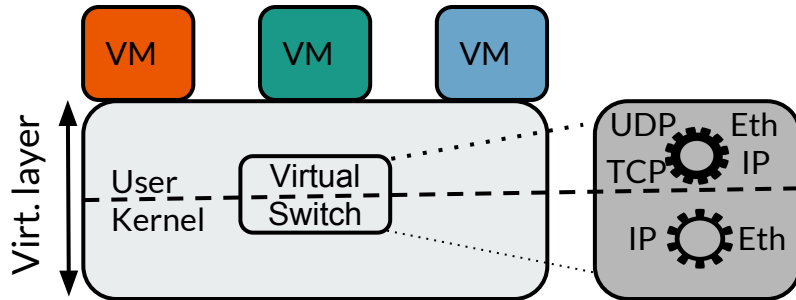
Position

- Adapt the design principles by Saltzer and Schröder [1] to Virtual Switches
- Secure Design Principles for Virtual Switches
 - Isolate the host from the vSwitch
 - Isolate the tenant vSwitches from each other
 - Least privilege packet processing
 - Reduce the Trusted Computing Base (TCB)

The Secure Virtual Switch Vision

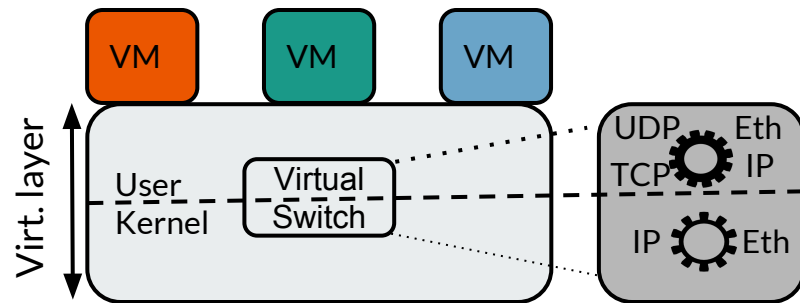


Vision

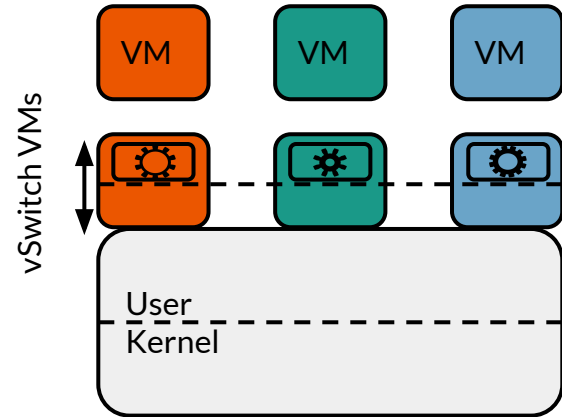


Existing vSwitch
design

Vision



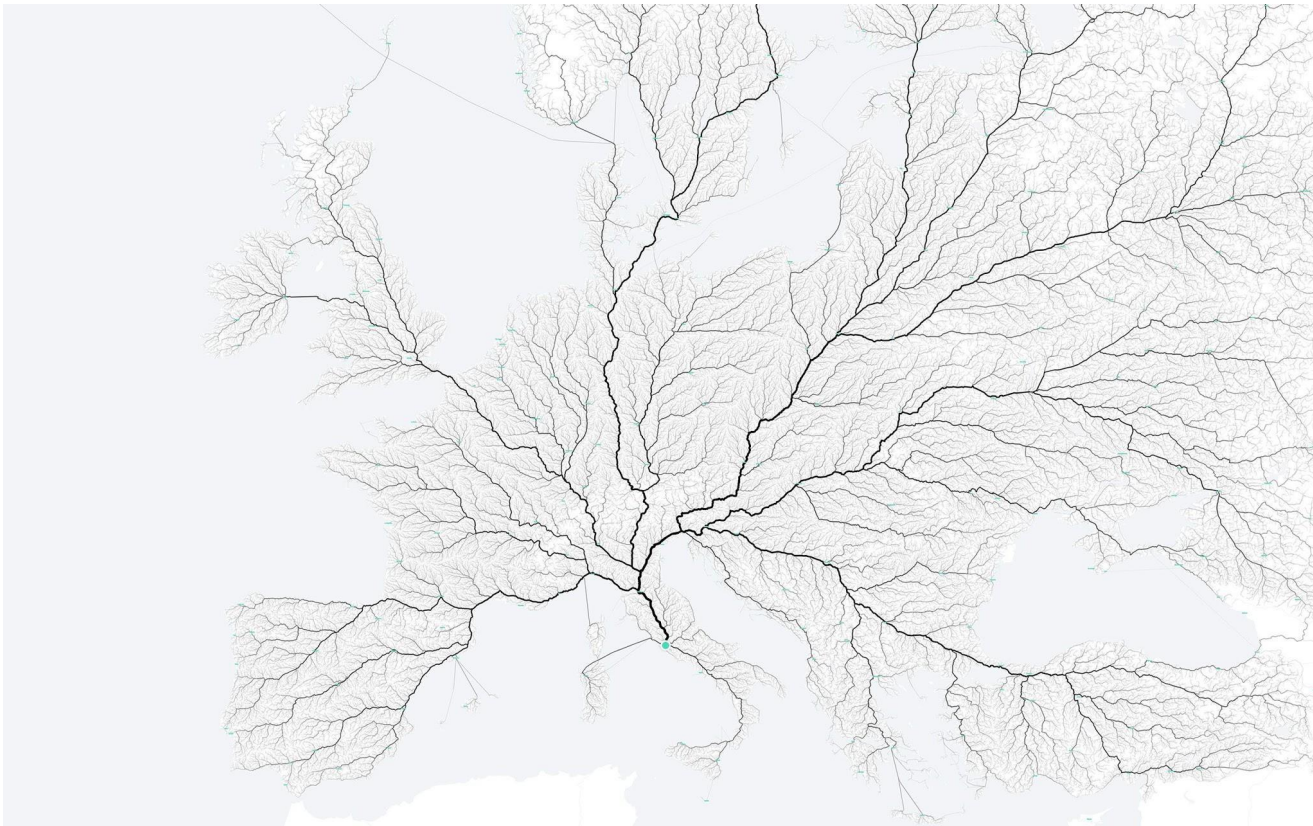
Existing vSwitch design



A Secure vSwitch design

Design Space

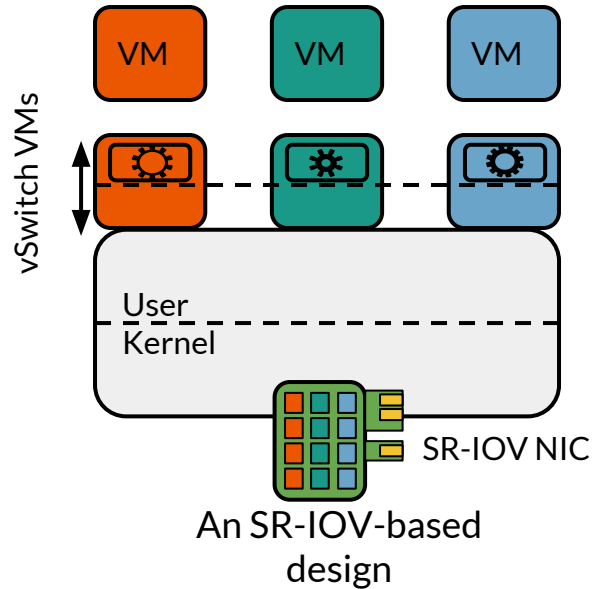




All Roads Lead to Rome

Figure credit: moovel Lab,
<https://lab.moovel.com/projects/roads-to-rome>

Design Space

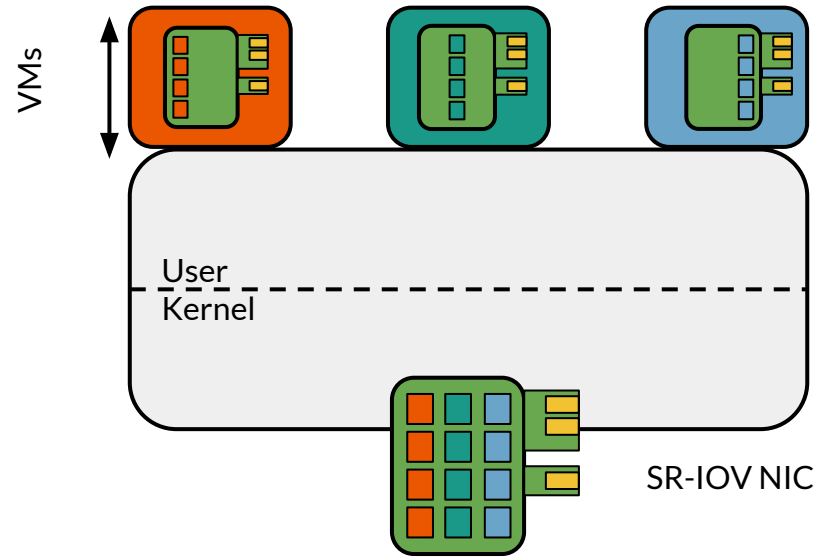


- Isolate the virtual switch from the Host
 - VMs, Container, Process
- Isolate tenant virtual switches
 - VMs, Containers, Process
- Least privilege packet processing
 - User-space, VM, memory safety
- Reduce the TCB
 - Separation of virtual switch from the Host
 - Limited parsing



SR-IOV

Single Root IO Virtualization





SR-IOV

Single Root IO Virtualization

- PCIe standard for IO Virtualization
- Allows a VM direct access to the NIC
- Dedicated registers for the NIC
- Physical Functions
- Virtual Functions



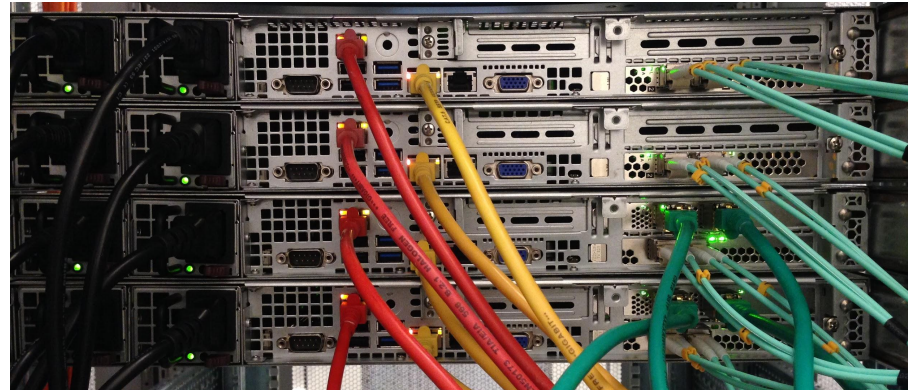
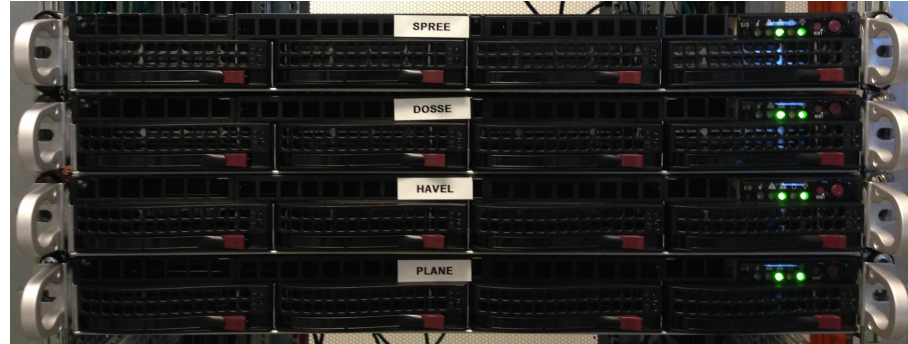
Challenges

- Reachability/Connectivity when the vSwitch is in the VM
 - L2
 - L3
 - ARP
- Drivers
- Resources
- Management
- Security of SR-IOV



First Experience

Servers: Supermicro (Intel Xeon(R)
E5-2609 v4 Single Socket 8 Cores)
NICs: SolarFlare, Mellanox, Netronome





First Experience

- + It works!
- + Easy to configure
- + Good documentation
- + Each NIC is different (duh!)
- + Mellanox ftw!
- + Differences in PF and VF
- + Security features offered
- SolarFlare VFIO and DPDK in VM not yet supported
- PF driver not in VM
 - SolarFlare can do PF-IOV in 8 VMs only
- Layer 1 issues
 - Netronome only 10G, not 1G
 - Multi-mode vs Single-mode Splitter
- Should have got an Intel NIC as well
- IPv6 Multicast when VFs come up?



Conclusion

- 4 Security weaknesses with existing virtual switches
- 3 Secure design principles for virtual switches
- Introduced a first secure virtual switch design



Future Work

- Extend the evaluation
- Explore the design space
- Security of SR-IOV



Contact

Kashyap Thimmaraju

Email: kash@sect.tu-berlin.de

Web: www.fgsect.de/~hashkash

Fingerprint: 5FFC 5589 DC38 F6F5 CEF7 79D8
A10E 670F 9520 75CD



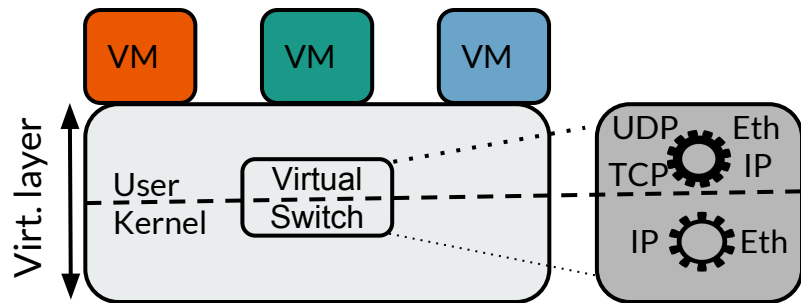
References

1. Saltzer, Jerome H., and Michael D. Schroeder. "The protection of information in computer systems." Proceedings of the IEEE 63.9 (1975): 1278-1308.

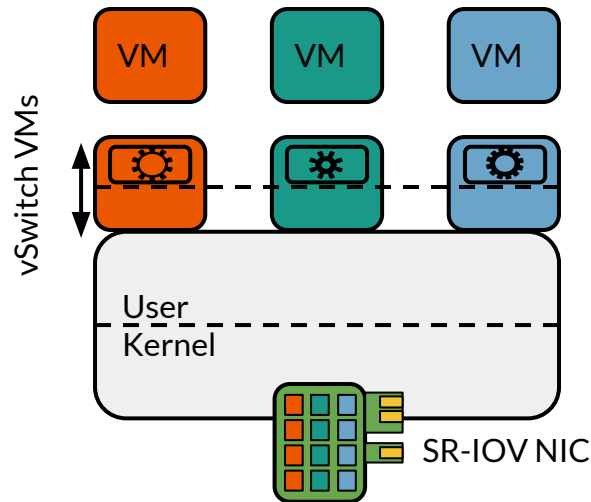
Backup



Vision



Existing vSwitch design



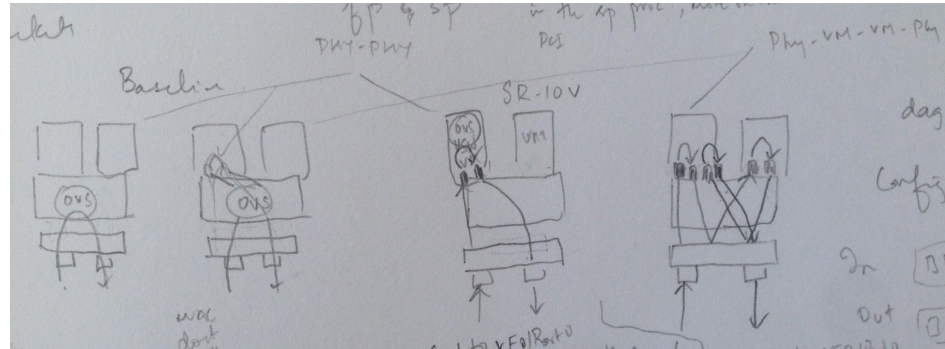
An SR-IOV-based design

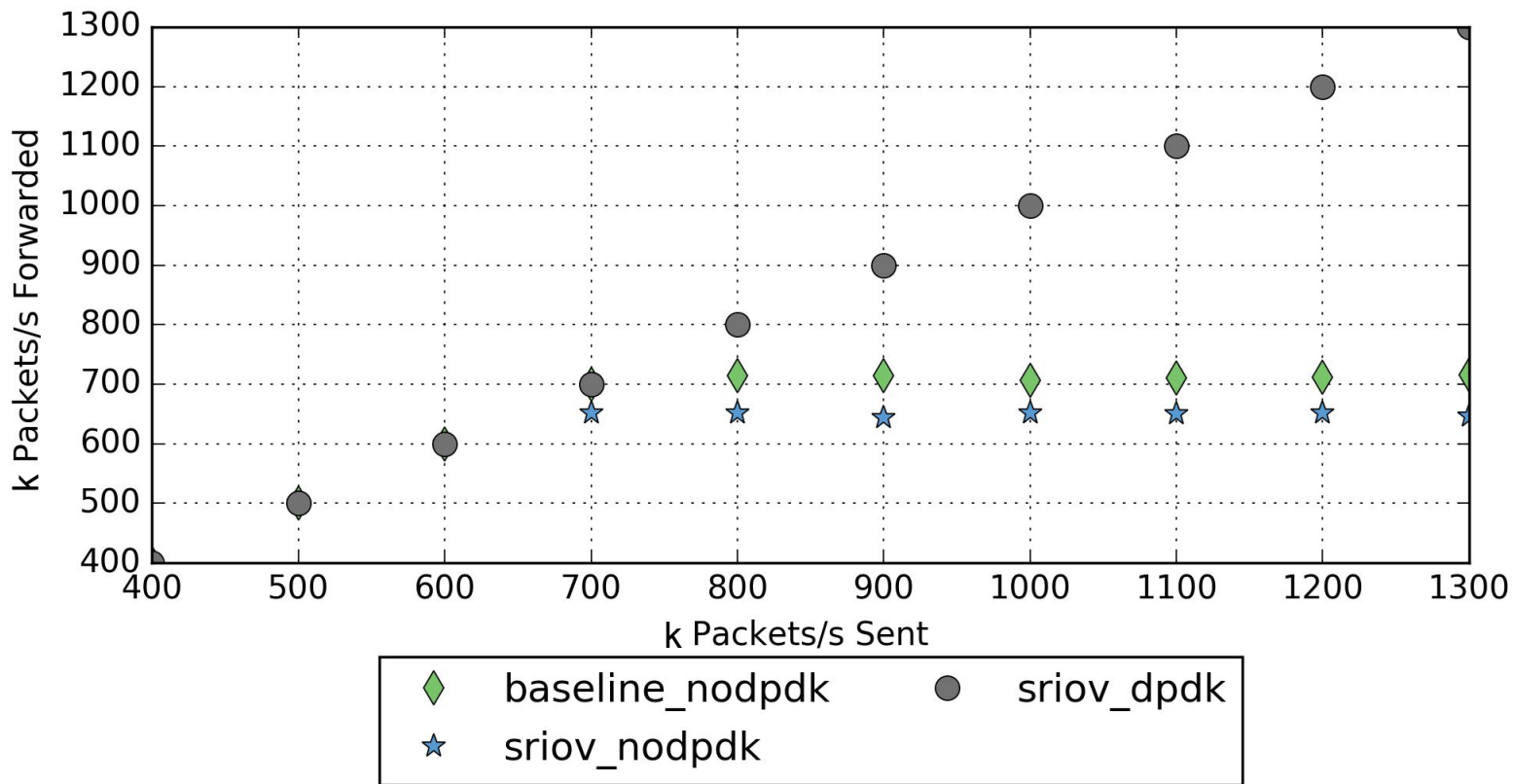


First Measurements

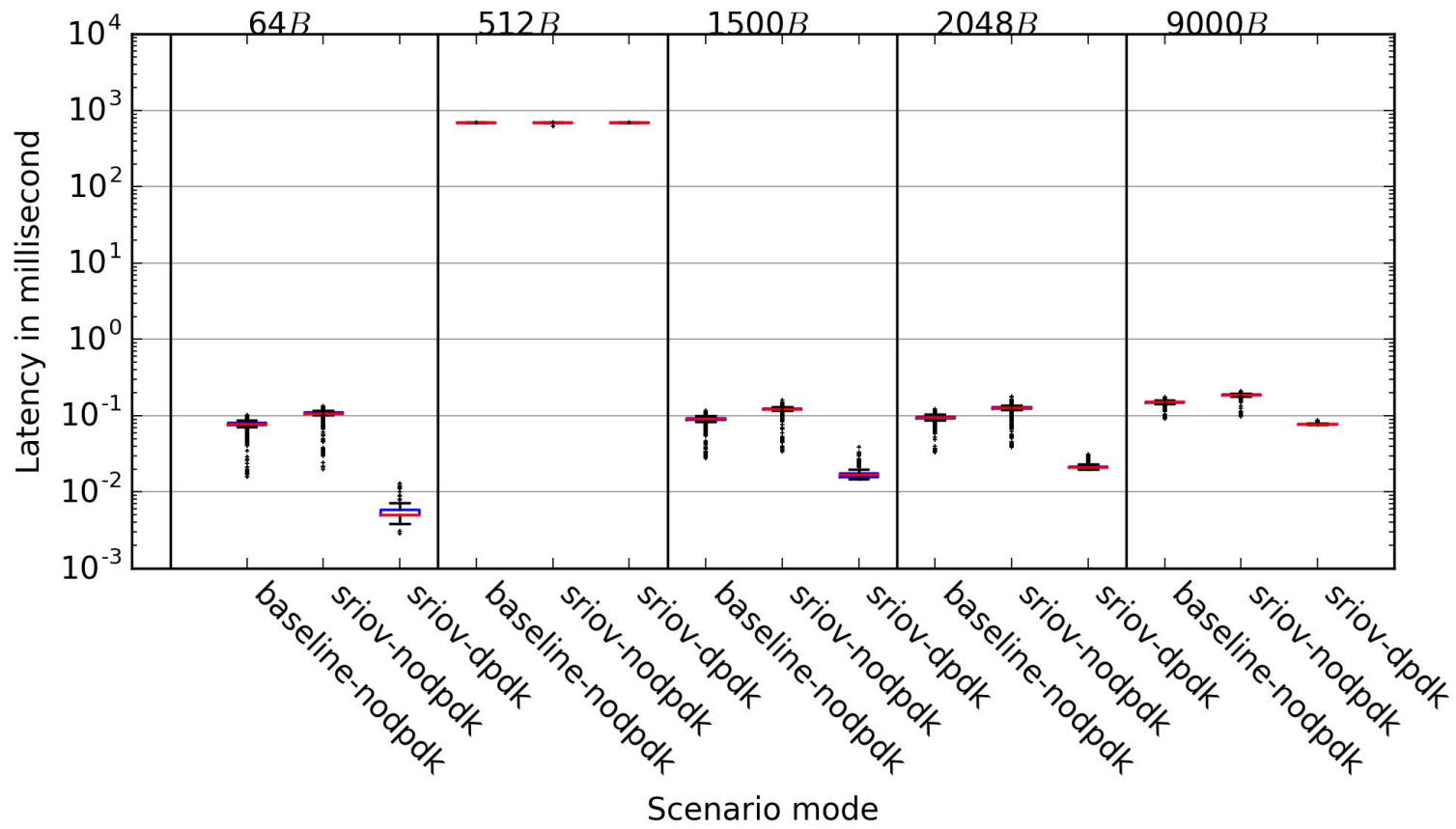
Measurement Setup

- Broadly 2 topologies
 - PHY-PHY
 - PHY-VM-PHY
- Uni-directional

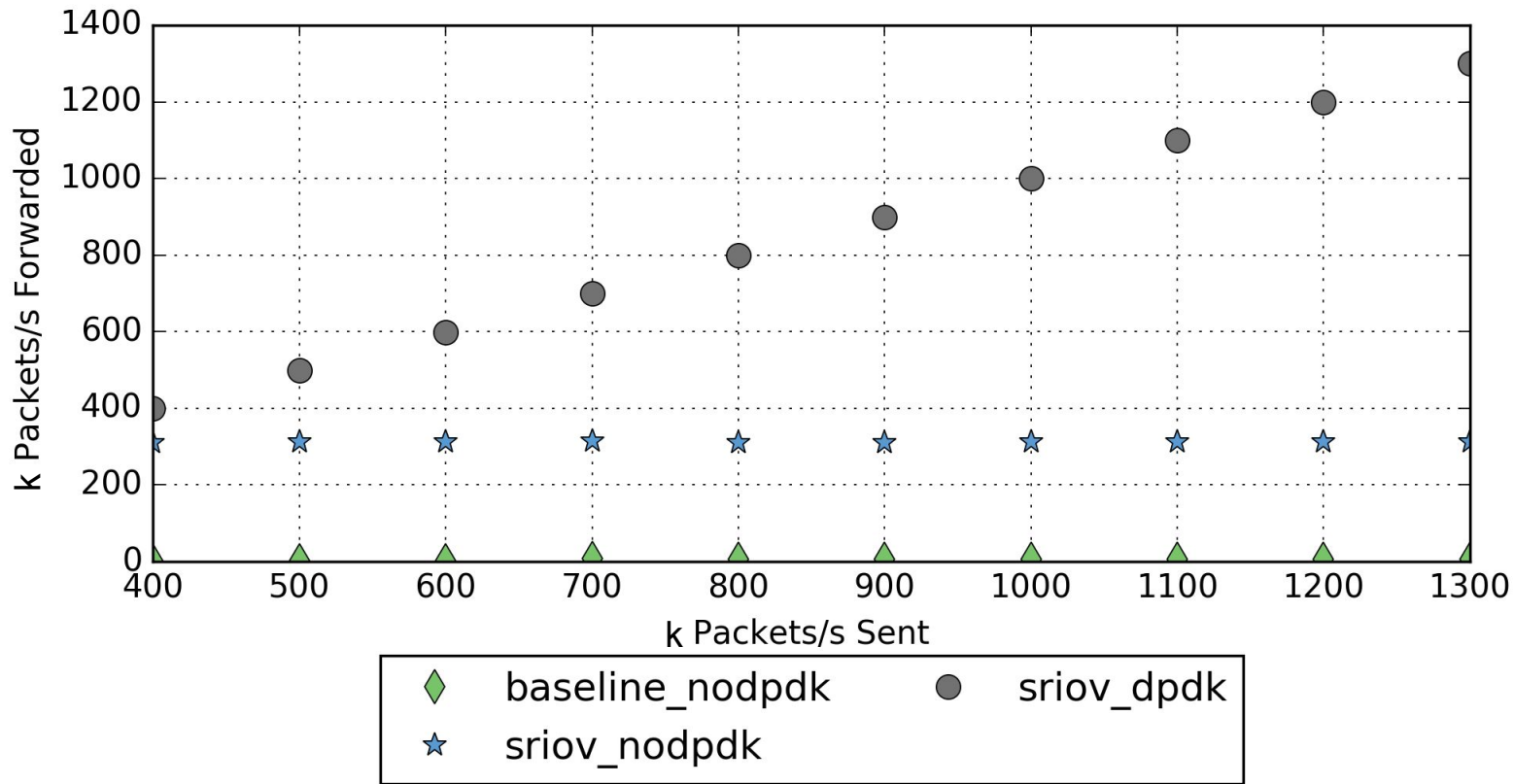




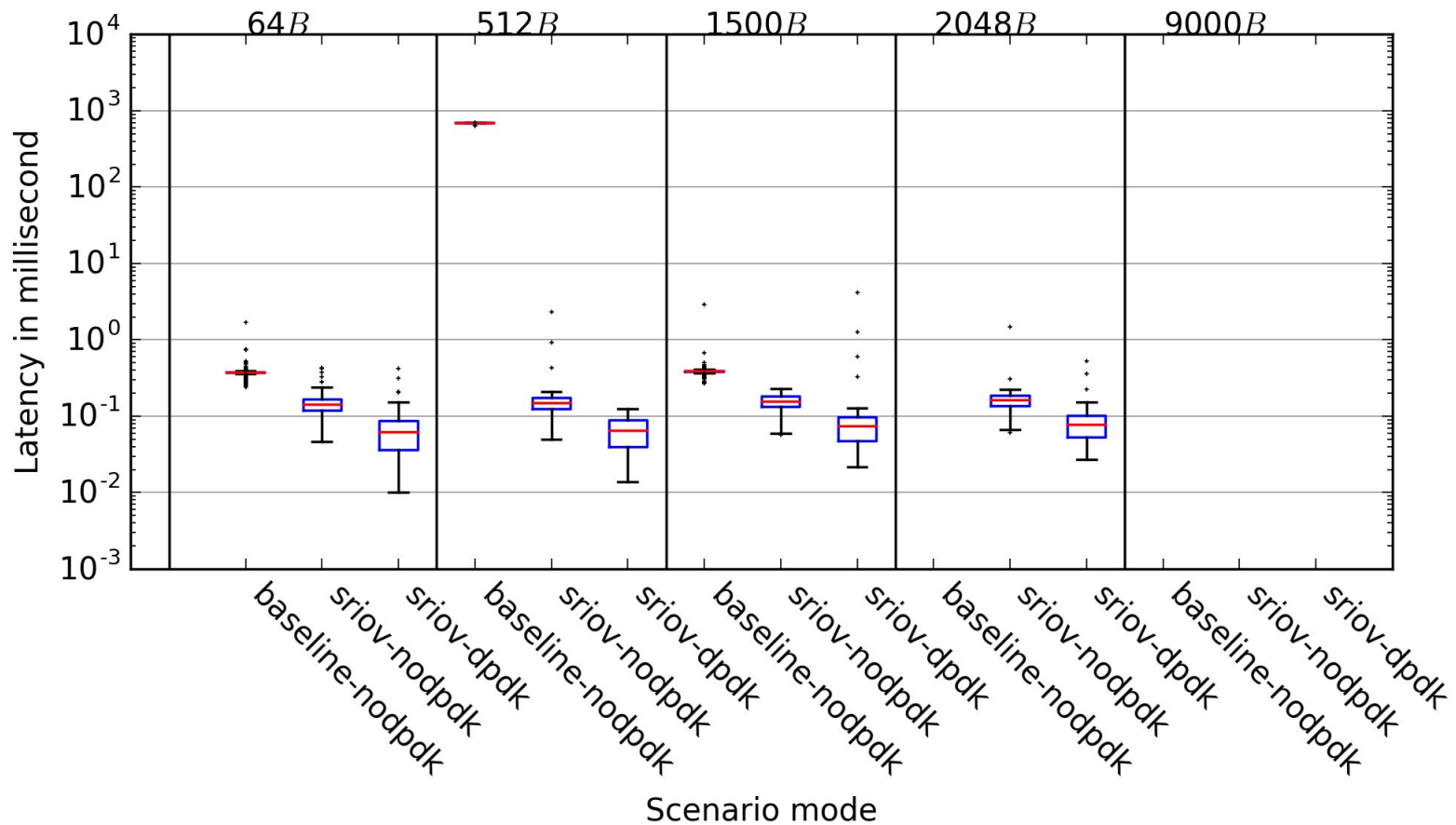
Phy-Phy 1 Tenant Uni-directional Aggregate Throughput



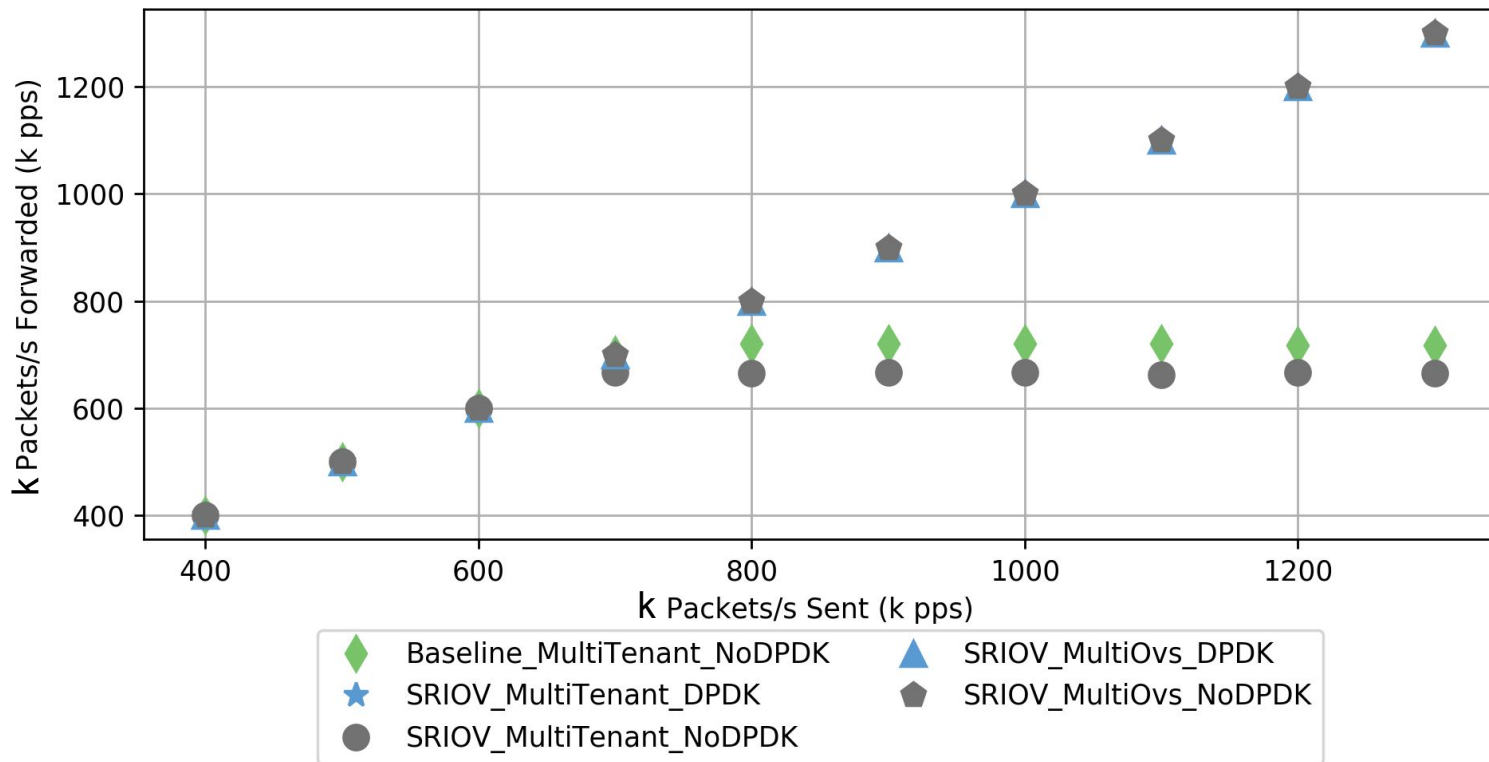
Phy-Phy 1 Tenant Uni-directional Aggregate Latency



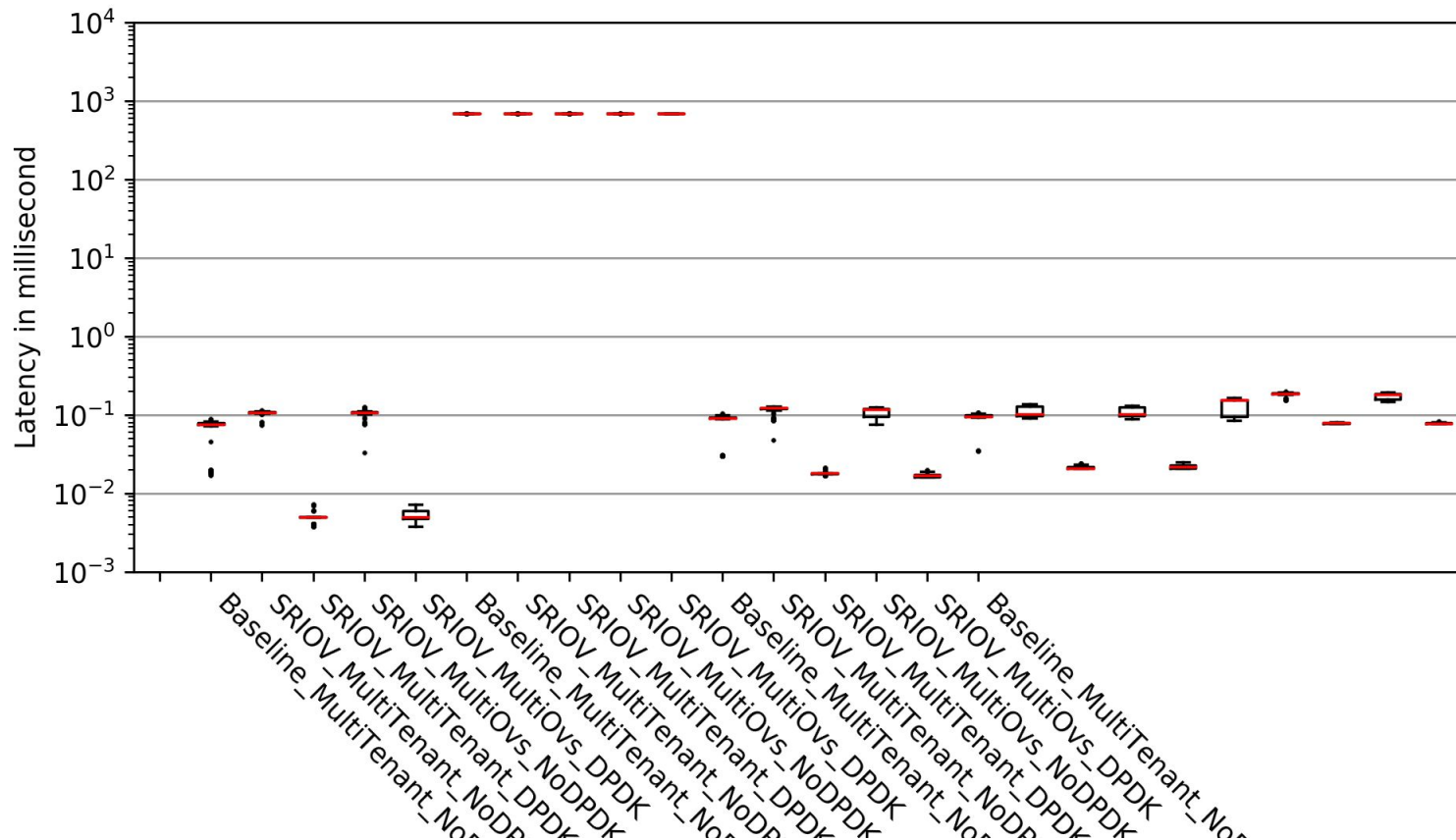
Phy-VM-VM-Phy 1 Tenant Uni-directional Aggregate Throughput



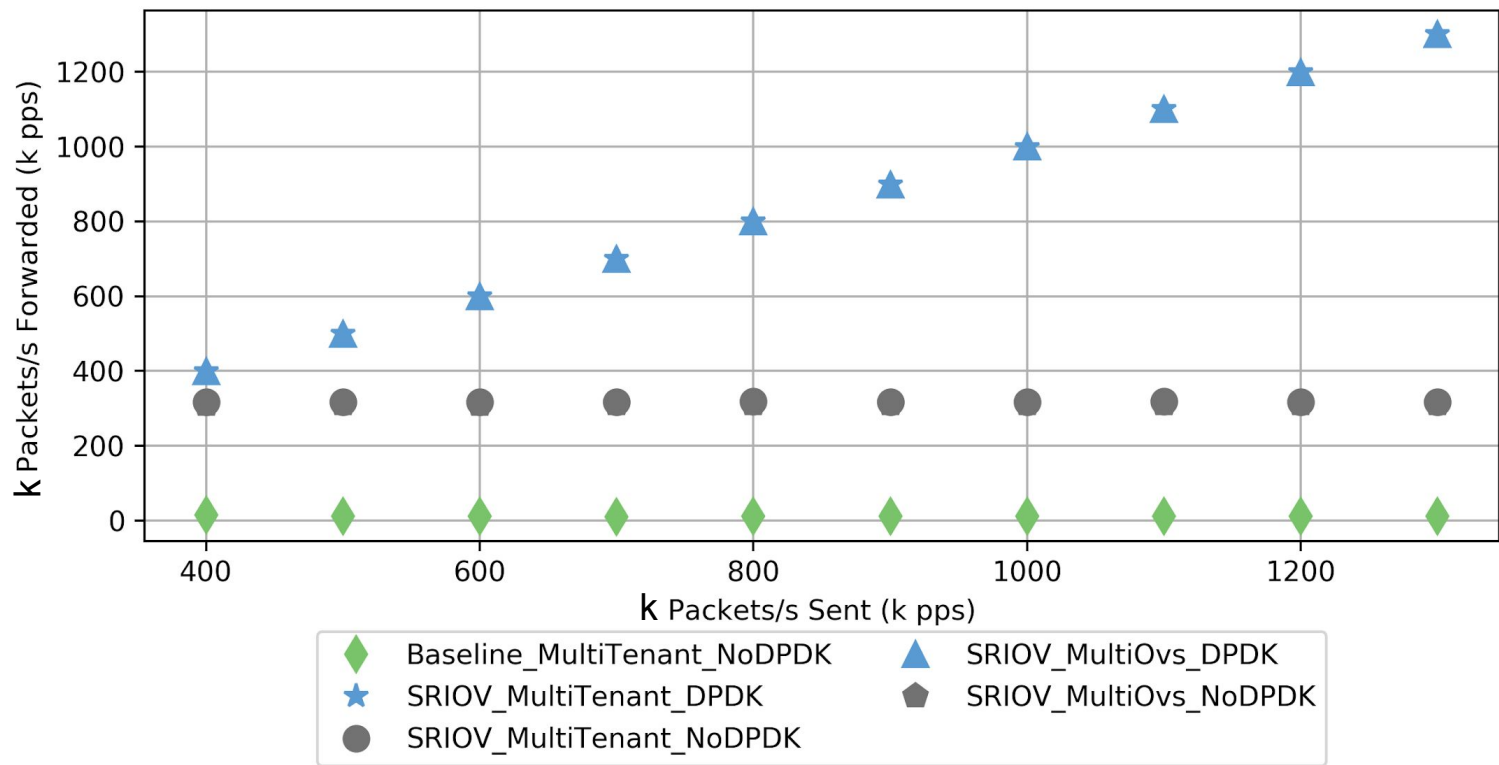
Phy-VM-VM-Phy 1 Tenant Uni-directional Aggregate Latency



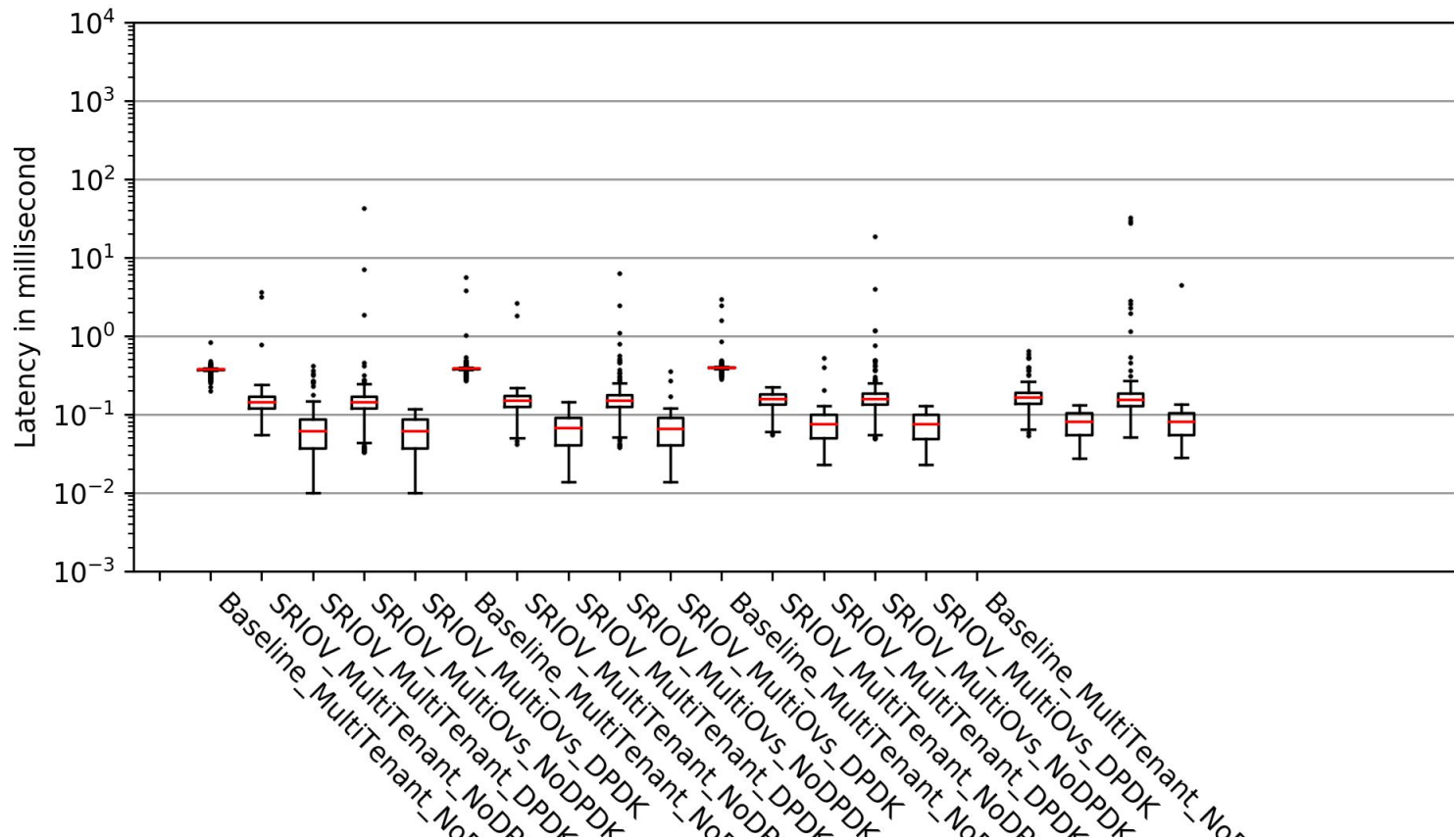
Phy-Phy 2 Tenants Uni-directional Aggregate Throughput



Phy-Phy 2 Tenant Uni-directional Aggregate Latency



Phy-VM-VM-Phy 2 Tenants Uni-directional Aggregate Throughput



Phy-VM-VM-Phy 2 Tenant Uni-directional Aggregate Latency



Lessons Learned



What have we learned so far?

- Performance
- Resources
- Management