Network Virtualization in a Shared TCP Fast-Path

by

Mehrshad Lotfi Foroushani

Master Thesis

Software Engineering Chair
Faculty of Natural Sciences and Technology I
Department of Computer Science
Saarland University

Supervisor
Antoine Kaufmann

Advisor
Antione Kaufmann

Reviewers
Antione Kaufmann
Anja Feldmann

January 30, 2023

UNIVERSITÄT DES SAARLANDES

Declaration of Authorship

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Datum/Date: ______________________________

Unterschrift/Signature: ______________________________

SAARLAND UNIVERSITY
Software Engineering Chair
Department of Computer Science

Abstract

Network Virtualization
in a Shared TCP Fast-Path
by Mehrshad Lotfi Foroushani

Content

Chapter 1

Introduction

Virtualization has gain a lot of popularity during the last two decades. Enabling fast provisioning, increasing the availability time, and reducing the maintenance costs for users are a few reasons of the gained popularity. Moreover, by multiplexing multiple virtual machines (VMs) on a same physical server less enegry gets consumed, which makes virtualization enviromental friendly and even more popular thesedays.

Public cloud providers such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP) are expanding the range of workloads which can be spilled over virtualized environments. These vendors, in addition to providing high number of VMs with increased performance, have made it possible for public tenants to migrate their home workloads without changing the on-premises networking configurations. Furthermore, the users of these environments benefit from supplied address spaces, security groups and ACLs, scalable load balancers, bandwidth metering, QoS, and more [1].

The workloads applied by cloud tenants are bursty, which require high throughput and low latecy network access. It was this requirement, which motivated cloud providers to increase their networking speeds by more than 40x and more in a span of only a few years [2]. While networking speed is growing fast, the CPU improvement has become slower, and has experienced the end of Moore's law [3]. Hence efficient packet processing is becoming more and more important.

Due to poor cache allocation, costly context switches, and resources sharing across different cores, Linux TCP packet processing is considered as inefficient for modern data center networking [4, 5]. Different techniques tried to address the issue by reducing the overheads and improving the conventional TCP stack. These techniques include bypassing kernel to enable direct NIC access from user-space [6–8] using receive side scaling (RSS) to carefuly steering and processing packets on multi-cores architecture [4, 9], and offloading packet processing to NICs with computation capabilities [2, 5, 10, 11].

Although these techniques have imporved the performance of end-host packet processing, they suffer from two fundamental implications when it comes to providing high throughput and low latency network access for applications residing on VMs.

Implication I. Majority of these techniques are proposed only for applications running on bare metal servers, and they do not provide the rich virtualization features required in multi-tentant datacenters. Hence, cloud providers hesitate to deploy the techniques for their tenants and applications still use the conventional TCP packet processing.

Implication II. Cloud tenants are only concerned with performance of their applications, and they do not want the maintenance of the network stack to fallout on their shoulders. Deploying new packet processing stacks requires extensive testing and comes with the risk of misconfigurations. Therefore, cloud tenants hesitate to deploy the techniques in trade with high throughput and low latency.

Goal In this thesis, we ask the following question: *How can we provide virtualization features on top of the modern stacks?* In response, we enrich Virt-TAS with virtualization features. Virt-TAS is a TCP acceleration service (TAS) for virtualized environments targeted at applications that require low latency and high throughput. It runs as a service alongside the host and provides a fast-path for common send and receive operations, through which it reduces the virtualization overheads incurred by the hypervisor and guest operating system [12].

To avoid reinventing the wheel, instead of implementing a virtual switch from scratch, we complement Virt-TAS by integrating Open vSwitch (OVS) [13]. OVS is a virtual switch which is deployed in many data center networks. It is also a component of VMware's NSX product, used by thousands of customers [14].

NetKernel, is another system that provides network stack as a service in the cloud [15, 16]. Like Virt-TAs, Netkernel decouples the network stack from the guest OS, (*i*) to simplify deployment and maintenance for tenants, (*ii*) to increase efficiency of TCP packet processing for cloud providers. Netkernel's goal is to show the feasibility of decoupling network stack from VMs, whereas Virt-TAS focuses on providing network virtualization features on top of decoupling network stack, and offering the same isolation as current system architecture.

This thesis makes the following contributions.

- We extend Virt-TAS memory layout to support *group* terminology. Groups like target groups in Amazon EC2 [17] are a set of VMs, which trust eachother and can share memory. Each VM in Virt-TAS is assigned to a group and a group can be assigned to multiple VMs.

- We integrate OVS to steer packets between network interface card (NIC) on hypervisor, and applications residing on the VMs. By using OVS, we benefit from network programmability support added through OpenFlow interface. Each VM in the new design is connected to OVS through a shared memory maintained by its group. TCP acceleration service on the other side connects NIC to OVS through lock free queues.

- We further optimize the solution by leveraging fast-path processing. After processing the first packet of each connection in OVS, Virt-TAS stores connection information for the flow in its cache and the following packets are processes by the fast-path.

Chapter 2

Background

2.1 Network Virtualization

Network virtualization has gained an extreme importance in data center networking as it enables cloud providers to enhance the flexibility, scalability and resource utilization of the network infrastructure. With network virtualization multiple virtual networks can be deployed on a single physical infrastructure.

In this section, we first explain what network virtualization is and how it is different from the conventional non-virtualized envrionments (Section 2.1.1). We then explore different network virtualization platforms and investigate their shortcommings in providing virtualization features for low-latency and high-throughput data center applications (Section 2.1.2).

2.1.1 Non-virtualized vs. virtualized network environments

In a non-virtualized network environment, networking components are directly tied to the underlying physical hardware. Each networking component, such as a switch or router, corresponds to a specific physical device. Therefore, the network as a whole is limited by the available hardware resources. Furthermore, configuration in these environment is a hard and time-consuming task. Each networking vendor has its own interface, and configuring each device requires the corresponding expertise. In addition to that, adding or removing a machine requires multiple configurations to be set up in a box-by-box manner, which introduces an excessive operation overhead and increases the risk of misconfiguration [9, 18].

Network virtualization aims to solve these issues by separating and abstracting network infrastructure from the underlying hardware. Through a virtualization layer, users can create virtual networks, with arbitrary service models, topologies, and addressing

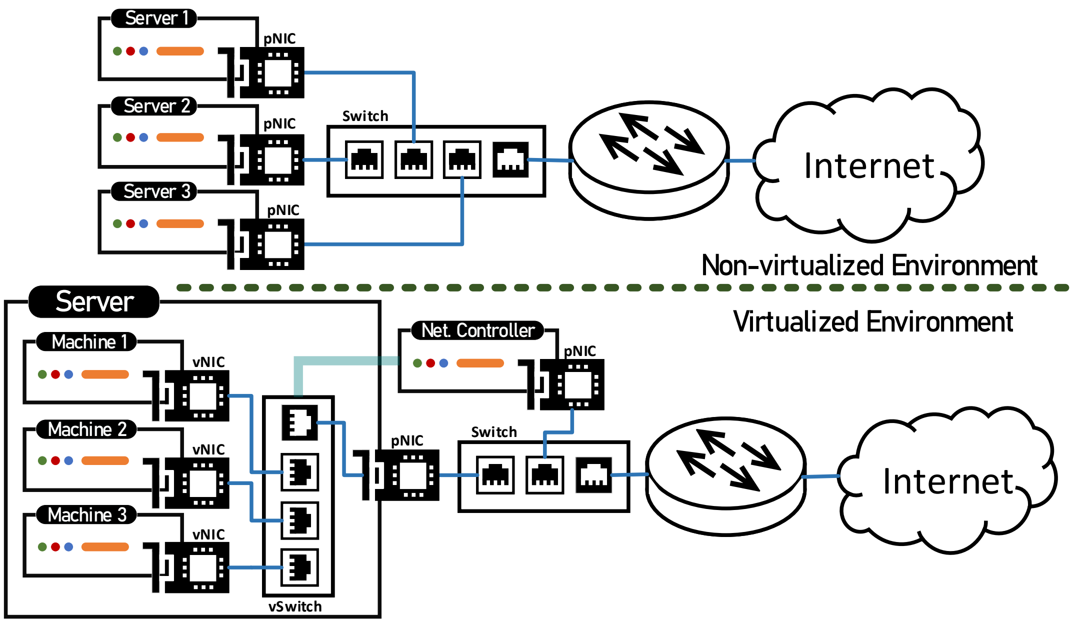

FIGURE 2.1: Non-virtualized vs. virtualized network environment

scheme, on top of the same physical network devices. Furthermore, a global abstraction enables management and configuration of these virtual networks [19].

Figure 2.1 illustrates differences between a non-virtualized and a virtualized networking environment. In a non-virtualized setup, the kernel directly routes packets from applications to a physical Network Interface Card (pNIC), binding the applications to the physical networking components. Whereas, in a virtualized setup, software-based virtual NICs (vNICs) and virtual switches (vSwitches) are used to abstract and separate packets from the underlying hardware through an encapsulation layer. Moreover, a network controller in these environment is used to setup the policies for encapsulation and forwarding decision on the virtual switches.

2.1.2 Network virtualization platforms

To provide network virtualization for cloud tenants, public cloud providers, such as Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP), have taken different approaches. While Microsoft has been following a hardware-assisted procedure to provision networking demands [1, 2], Google has been utilizing a software-only approach [9]. In this section, we first introduce two seminal open-source projects in network virtualization, namely OpenFlow (Section 2.1.3), and Open Virtual Switch (Section 2.1.4). We then elaborate on the state-of-the art network virtualization used by VMWare (Section 2.1.5), Google (Section 2.1.6), and Microsoft(Section 2.1.7).

2.1.3 OpenFlow

OpenFlow was first proposed to enhance the implementation of new protocols on top of the production networks. It consists of a standardized interface to add or remove entries from flow tables residing on the Ethernet switches [19].

Flow tables are the fundamental data structure of an OpenFlow switch. They consist of flow entries and proper actions to the packets of each flow. A flow is a set of packets that have similar characteristics. It could be a TCP connection, all packets from a particular port number, or packets with the same VLAN tag. An OpenFlow switch matches the packets with the specified flows and applies the action of that flow. For example, all packets from a particular sender could be dropped by the OpenFlow switch.

The three basic actions that are supported by an OpenFlow switch consists of (*i*) forwarding packets to a given port (or ports), (*ii*) encapsulating and forwarding packets to a controller, and (*iii*) dropping packets.

In summary, an OpenFlow switch has the following three main components:

1. *A Flow-table residing on the Ethernet switch*: This flow table maintains information about existing flows and actions required to be applied to the corresponding packets.
2. *Secure interface between a controller and the switch*: The interface is used when a packet has no corresponding entry in the flow table. In such scenarios, the packets will be forwarded through this interface to a controller. The controller can then decide for the corresponding action.
3. *Standardized OpenFlow interface to program flow tables*: With this standardized interface, a controller can update the flow-tables on Ethernet switches. Therefore, not all the packets are required to be transferred to the controller, hence higher throughput can be achieved.

Further detailed specifications of an OpenFlow switch are defined by *Open Networking Foundation* [20].

VirtTAS supports OpenFlow interface to enable network programmability at end-host.

2.1.4 Open Virtual Switch (OVS)

Conventional virtual switches are concerned only with providing a basic network connection for VMs through L2 networks. This was changed with the advent of network virtualization. Today's virtual switches provide most network services for VMs, and the physical networks only transmit IP-tunneled packets. Therefore, VMs are no longer tightly connected to physical datacenter networks and the workloads benefit from higher scalability and mobility.

As the complexity of network virtualization increased, the need for a production-level multi-layer virtual switch became a necessity. Open vSwitch (OVS) was designed and implemented to address this need. It enables network virtualization through OpenFlow and supports standard interface management protocols such as NetFlow[21], sFlow[22].

Open vSwitch has a long history of development and in order to keep up with the production workloads and increase the development velocity, the following key design choices were made by OVS community:

1. **Use tuple search classification.**. Tuple search classification supports constant time updates. Constant time updates are essential in virtualized environments, where a controller sends flow updates multiple times a second. Furthermore, it consumes memory linear in the number of flows, and it can be generalized to all combinations of packet header fields.
2. **Split implementation between Kernel and userspace.** Kernel-based networking makes the development and release cycles lengthy. Moreover, OVS would not be accepted as a contribution to upstream Linux, if it was implemented solely in Kernel. Thereby, OVS uses kernel as a match and action cache and the main classification labor is done in userspace by a virtual switch daemon.
3. **Benefit from caching.** OVS uses a two-layers caching mechanism to match packets in the Kernel. A Microflow cache layer matches on specific flows with exact headers fields, and a Megaflow cache layer matches on a wildcard of flows. These wildcards are generated by the cross-product of OpenFlow match and action tables.

Although OVS has a positive impact on making virtual switches available for the networking community, there are a few shortcomings which makes it not suitable for large scale deployments.

1. **Explicit tunnel interface.** OVS uses an extra interface to setup tunneling actions, and it hardcodes models of tunneling to dataplane, rather than enabling a controller to specify details of tunneling. Therefore, adding complicated tunneling logics, such as ECMP routing, is difficult with OVS.

2. **No support for multi-controller model.** In a multi-controller software-defined network, a virtual switch transmits in-bound packets in the reverse direction of out-bound packets to make the states consistent, whereas OVS only supports forward transmission.

3. **No support for stateful actions** Actions are not stateful, there are trackers supported by OVS but it does not support stateful actions.

4. **Performance** Packets that are not matched in the fast-path cache are handled by the slowpath and in specific cases, these packets are forwarded to a controller. This makes OVS vulnerable to DDoS attacks and cache polution.

Despite the limitations of OVS, we decided to integrate OVS into our shared TCP fast-path, and take the benefits of its long history of development and deployment. Addressing the limitations of OVS is beyond the scope of this thesis and we believe the OVS community can address these issues in the future works.

2.1.5 Network Virtualization Platform (NVP)

Network Virtualization Platform (NVP) is one of the first commercial virtualization platform based on SDN that appeared with the market demand for network virtualization and research on Software Defined Networking (SDN) [23]. NVP was proposed to address virtualization primitives such as VLAN (Virtualized L2 domain), VRFs (Virtualized L3 FIB), NAT (Virtualized IP address space), and MPLS (Virtualized Path) needed by VMware's users.

NVP allows users to setup, and configure different virtual networks, each with different addressing space, independent services model, and topologies on top of the same physical network. It hides the topology of the underlaying physical network from the users, and specific aspects of the forwarding devices becomes unknown for the users. In NVP, a hypervisor layer translations each tenant's network configurations to low-level instruction set, which can be installed by virtual switches. NVP deploys OVS as its virtual switch. The packets are tunneled by virtual switches over the physical network, and the physical switch only transmit tunneled packets between hypervisors and gateways.

Virt-TAS like NVP, enables virtualization primitivs through use of OVS at end-host. NVP, focuses on providing network connectivity to VMs and network programmability for operators, whereas Virt-TAS aims to supply VMs with high throughput and low latency by streamlining packet processing as a service on hypervisor in addition to providing the same features.

2.1.6 Google Snap Platform

As one of the primary cloud providers with internet-based products that are used by a wide range of users, Google has prioritized the following three main requirements for host networking in their data centers.

1. **Fast development and release cycles.** The network bandwidth is continuously increasing and delivering novel approaches for edge switching and bandwidth management is inevitable. Thus, a system that enables fast delivery of innovative paradigms is required.
2. **Virtualization features.** Rich virtualization features must be provided on top of host networking to enable cloud computing.
3. **Low latency and high throughput.** Distributed data-intensive applications require low latency and high throughput for their communication. As a consequence, the host networking system should be optimized with regard to these applications.

To reach the requirements of data-center networking and to mitigate the drawbacks of using the kernel in the development of end-host network stack, Google has proposed Snap. Snap is a userspace networking system, which was initially inspired by microkernel architecture. By residing in userspace, the development of new features has become faster and upgrades to the networking stack can be applied transparently. Moreover, in comparison to the Linux Kernel networking stack, it provides higher throughput.

2.1.7 Virtual Filtering Platform (VFP)

Microsoft Azure has deployed Virtual Filtering Platform (VFP) as its programmable virtual switch on more than 1M hosts to provide virtualization features [1]. VFP extends Microsoft Hyper-V's switch [24] with the switch logic, and it implements match and actions tables with multiple layers to support multi-controller programmability. VFP takes advantage of a programming model, which unlike OpenFlow consists of complex actions.

The key advantage of VFP is its performance which was made because of just in time compiler and metadata parsing and touching packet once all decisions are made.

Unfortunately the implementation details of VFP is not publicly available and we cannot evaluate performance benefits of extending VFP to streamline packet processing on endhost. Nevertheless, we beleive the main idea is still applicable to VFP. We believe by providing packet processing as a service at hypervisor, the tenants do not need to provision higher than their demands.

2.2 TCP acceleration

TCP acceleration techniques are used in data center networking to increase TCP's performance on high-bandwidth, low-latency networks. Some of these techniques are as follows:

- **TCP/IP Offload Engine (TOE).** TOEs are hardward-based solutions that offloads TCP/IP processing from the host CPU to a dedicated hardware on network interface card (NIC), therefore lowering CPU usage and accelerating network performance [5, 25–27].
- **Kernel bypass.** Kernel bypass is another technique used to directly access the NIC from the application in userspace. This technique enables low latency communication between the application and the NIC, as it eliminates the need for the data to pass through the kernel's network stack [4, 9, 28].
- **Congestion Control.** Different congestion control are used to manage network traffic and to prevent congestion and data loss [29, 30].

In this section, we focus on kernel bypass technique and we elaborate the TCP Acceleration as Service (TAS) project [4].

2.2.1 TCP Acceleration as Service (TAS)

TAS is an acceleration service, which splits TCP packet processing into two components: a light weight fast-path optimized for common case client-server RPCs, and a heavier stack slow-path which drives congestion control, connection steup.

The fast-path benefits from different streamlining opportunity. The user can scale the dedicated cores for efficient and fast processing of packets.

2.3 Related works

2.3.1 NetKernel

Netkernel is another system that offers network stack as a service in virtualized environments. Similar to VirtTAS, Netkernel decouples the network stack from the VMs. Thus, (*i*) tenants benefit from fast deployment and maintenance of the network stack, and (*ii*) cloud providers are able improve the efficiency of TCP stack processing stacks transparently.

In the design of Netkernel, the BSD socket APIs are redirected to a full NetKernel socket implementation through library called *GuestLib*. The library is deployed as a kernel patch, and it is the only modification made to the users' virtual machines. Network stacks are set up by the operator on different virtual machines inside the same server as *Network Stack Modules (NSMs)*. The NSMs are connected from one side to users' virtual machines through lock-free queues on shared-memory between virtual machines and from the other side to a virtual switch through conventional tap devices.

The design of NetKernel is limited by processing network stack inside virtual machines. Furthermore, the full performance benefits claimed by NetKernel is only achieved when cloud providers deploy Single Root I/O virtualization (SR-IOV) in their datacenter. SR-IOV ties applications to the underlying hardware, and is not favorable to all cloud providers. Whereas, VirtTAS focuses on improving the performance of packet processing for application residing on user virtual machines through streamlining the packet processing on the hypervisor. VirtTAS benefits from the multi-core achitecture and uses kernel-bypass to achieve line-rate performance.

Chapter 3

Design

3.1 Objectives

Virt-TAS is designed to fulfill the following goals:

- **Low latency and high throughput** Modern datacenters host distributed data-intensive applications, that require low latency and high throughput. Workloads of these applications consist of short-lived flows with stringent low latency requirements, as well as long-lived flows with high throughput needs. VirtTAS should provide the latency and throughput needs of these applications, and it should enable cloud providers to maintain Service Level Agreements (SLAs).
- **Compatibility** The networking interfaces should be unmodified so that applications residing on the VMs could benefit from VirtTAS without any modification. Moreover, to make VirtTAS programmable through existing state-of-the-art network controllers, it should support OpenFlow API.
- **Mobility** VMs could be migrated to different locations, thus VirtTAS should allow VMs to continue communicating over the network, despite their migration. VirtTAS should decouple applications from physical networking infrastructures at their location.
- **High utilization** As VirtTAS shares fate with the hypervisor, resource conversation is highly critical. VirtTAS should maximize resource utilization to maintain precious resources for the main goal of the hypervisor, running user workloads.
- **Scalability** Scalability becomes a challenging problem when different tenants with different network topologies and networking demands reside on the same hypervisor. VirtTAS should keep up with the increasing number of flows. It should also support an increasing number of flow updates from a controller.

- **Isolation** Applications residing on one VM must be prevented to access packets from other VMs. They should not be able to violate other tenants' SLAs. Thus, VirtTAS should provide the same isolation as the current architecture.

3.2 Challenges

3.3 Design principles

To achieve the aforementioned goals, we benefit from a set of design principles to provide network virtualization features in a shared TCP fastpath.

3.4 Virt-TAS Architecture Overview

Similar to TAS, Virt-TAS devides the network stack into three different components: (*i*) A fast-path running on hypervisor, (*ii*) a slow-path that has components both on the hypervisor and the VMs, and (*iii*) an application library, which runs entirely on the VM side.

Figure X depicts the overview architecture of Virt-TAS. The packets are first received by the fast-path component from a physical NIC or through shared memory from applications running on VMs. The fast-path either has received instruction from OVS on how to handle the received packet based on its flow or it has not. In the first scenario, the fast-path applies the action provided by OVS. The actions ranges from dropping the packet to forwarding it to specific application connected to Virt-TAS or to the physical NIC. In the second scenario, where fast-path has no information about the flow of the received packet, the packet is delivered to OVS through lock-free shared memory. OVS can then decide how to treat packets. On the other side, the application interface has the responsibility to initialize the connection to Virt-TAS and implement POSIX network socket API for applications.

By integrating OVS, we can control and program the forwarding dataplane through OpenFlow protocol. OpenFlow enables network operators to add, remove, update entries, and to monitor statistics on flow tables. OVS takes OpenFlow tables from an SDN controller, matches the received packets to these flow tables, and applies all actions needed to be taken. The outcome is then cached by the fast-path component of Virt-TAS. This significantly simplifies the fast-path, as it allows the fast-path to be agnostic to OpenFlow specifics. One the other side, from the perspective of an OpenFlow

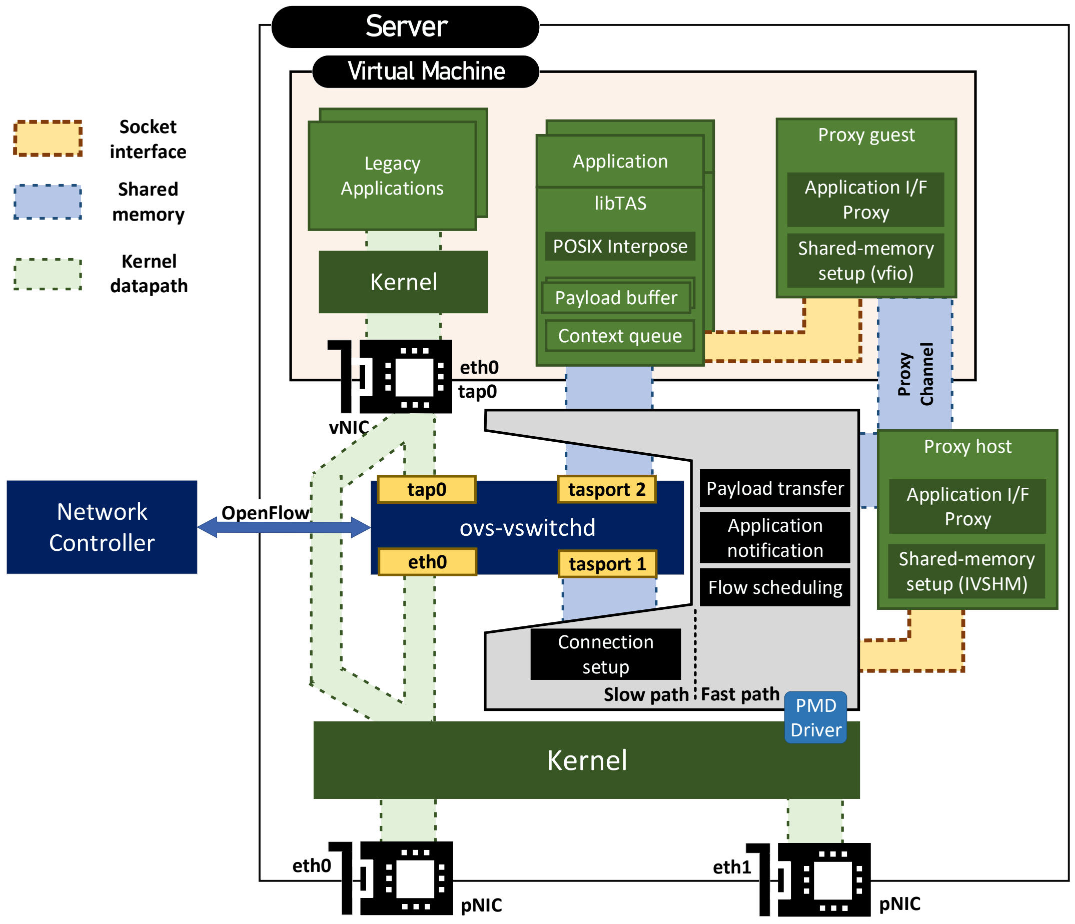

FIGURE 3.1: Virt-TAS big picture

controller, processing packets in Virt-TAS is an invisible implementation detail. In its perspective, all packets are matched against multiple flow tables, and the corresponding actions are applied among entries that satisfy all conditions and have the highest priority.

A userspace library redirects the POSIX socket API transparently, so that applications remain unmodified. Through this library the sockets calls are transmitted to the hypervisor instead of being transfered to the VMs network stack. Furthermore, By rewriting applications to use low level API of libTAS higher performance can be achieved.

3.5 Slow Path

3.5.1 Virtual switching

3.5.2 Proxy Host on hypervisor

The Proxy Host is a service that operates on the hypervisor as an intermediary between virtual machines and the TAS. It uses the libtas library to connect to the running TAS instance on the hypervisor. The Proxy Host determines the number of supported groups from the TAS instance and sets up a UNIX socket for each group. The path of the created socket for the desired group is then used to launch virtual machines with fast-path support.

```
qemu-system-x86_64 \
...
-chardev socket,path="PROXY_SOCKET_PATH",id="tas" \
-device ivshmem-doorbell,vectors=1,chardev="tas" \
...
```

Listing 3.1: QEMU system x86-64 command with ivshmem-doorbell and socket interface for connection to proxy host

runs a request to a server, it is sent to the proxy server first. The proxy server then makes a request to the server on behalf of the client, and returns the response back to the client.

3.5.3 Proxy guest on VMs

3.5.4 Application Interface

3.6 Fast Path

3.6.1 Connection setup

3.6.2 OpenFlow action caching

3.7 Application library

Chapter 4

Implementation

4.1 Slow Path

4.1.1 Open vSwitch Extension

4.1.2 Proxy server

4.1.3 Proxy guest

4.2 Fast Path

Chapter 5

Evaluation

5.1 Setup

5.2 Single flow overhead

Throughput

Latency

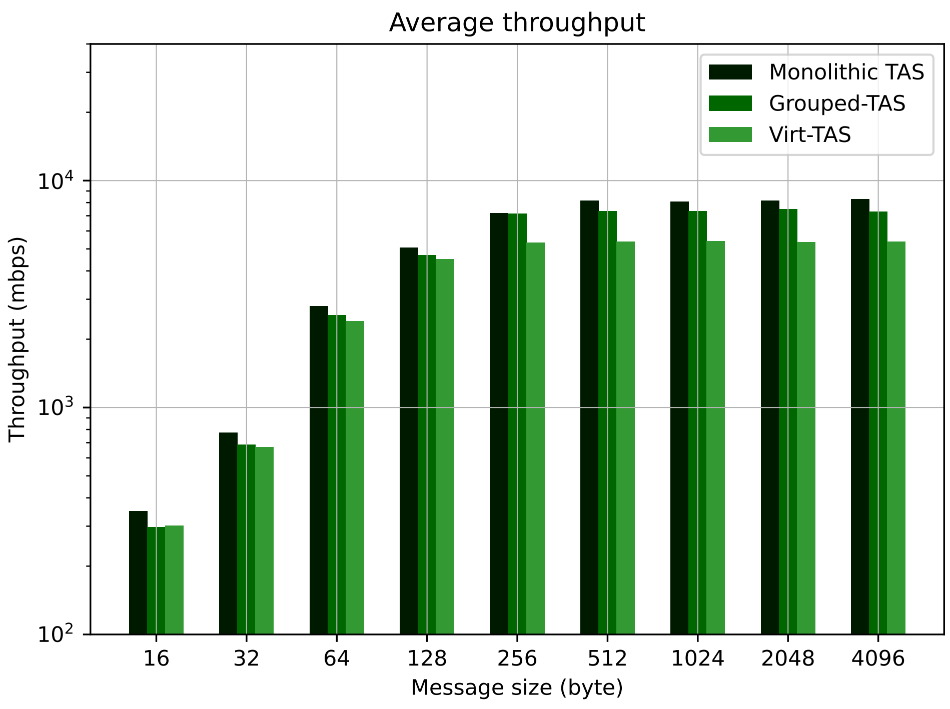

Figure 5.1: Single context overhead.

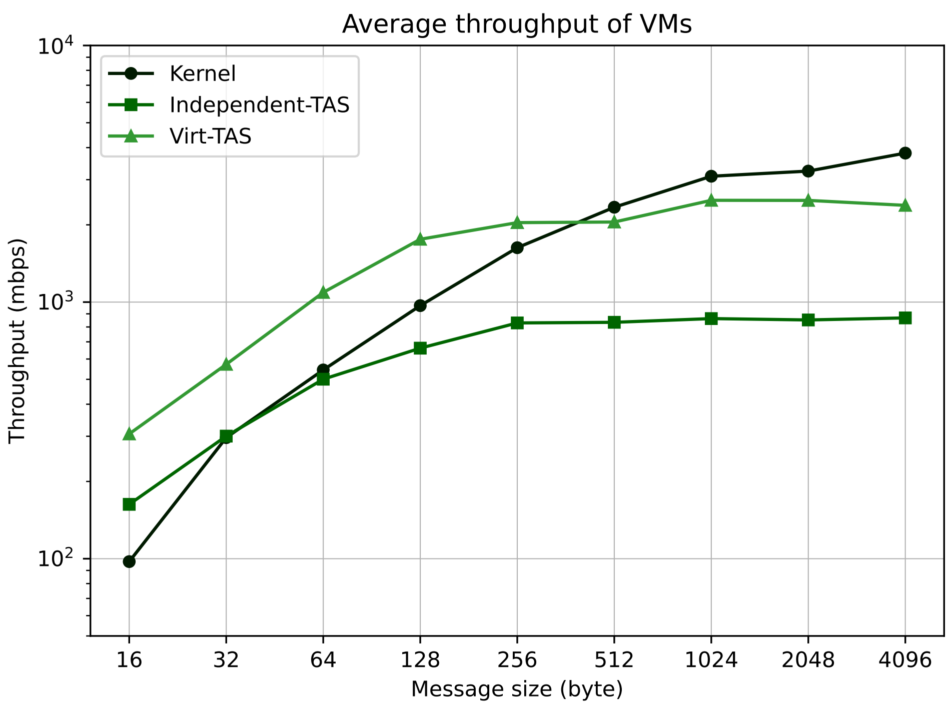

FIGURE 5.2: Using a shared TCP fast-path increase the resource efficiency in a multi-tenant environment

5.3 Multiplexing

By using a shared TCP fast-path among multiple virtual machines, tenants would benefit from the multiplexing benefits. To

5.4 Flow scaling

Chapter 6

Conclusion

List of Figures

List of Tables

Bibliography

[1] Daniel Firestone. {VFP}: A virtual switch platform for host {SDN} in the public cloud. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 315–328, 2017.

[2] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. Azure accelerated networking:{SmartNICs} in the public cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 51–66, 2018.

[3] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th annual international symposium on Computer architecture*, pages 365–376, 2011.

[4] Antoine Kaufmann, Tim Stamler, Simon Peter, Naveen Kr Sharma, Arvind Krishnamurthy, and Thomas Anderson. Tas: Tcp acceleration as an os service. In *Proceedings of the Fourteenth EuroSys Conference 2019*, pages 1–16, 2019.

[5] Rajath Shashidhara, Tim Stamler, Antoine Kaufmann, and Simon Peter. {FlexTOE}: Flexible {TCP} offload with {Fine-Grained} parallelism. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 87–102, 2022.

[6] Adam Belay, George Prekas, Ana Klimovic, Samuel Grossman, Christos Kozyrakis, and Edouard Bugnion. {IX}: a protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 49–65, 2014.

[7] EunYoung Jeong, Shinae Wood, Muhammad Jamshed, Haewon Jeong, Sunghwan Ihm, Dongsu Han, and KyoungSoo Park. {mTCP}: a highly scalable user-level {TCP} stack for multicore systems. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 489–502, 2014.

[8] George Prekas, Marios Kogias, and Edouard Bugnion. Zygos: Achieving low tail latency for microsecond-scale networked tasks. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 325–341, 2017.

[9] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C Evans, Steve Gribble, et al. Snap: A microkernel approach to host networking. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 399–413, 2019.

[10] Mina Tahmasbi Arashloo, Alexey Lavrov, Manya Ghobadi, Jennifer Rexford, David Walker, and David Wentzlaff. Enabling programmable transport protocols in {High-Speed}{NICs}. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 93–109, 2020.

[11] Jiaxin Lin, Kiran Patel, Brent E Stephens, Anirudh Sivaraman, and Aditya Akella. {PANIC}: A {High-Performance} programmable {NIC} for multi-tenant networks. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 243–259, 2020.

[12] Florian Bauckholt. virt-tas: Virtualization capabilities for the tcp acceleration service, 2021.

[13] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of open {vSwitch}. In *12th USENIX symposium on networked systems design and implementation (NSDI 15)*, pages 117–130, 2015.

[14] William Tu, Yi-Hung Wei, Gianni Antichi, and Ben Pfaff. Revisiting the open vswitch dataplane ten years later. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 245–257, 2021.

[15] Zhixiong Niu, Qiang Su, Peng Cheng, Yongqiang Xiong, Dongsu Han, Keith Winstein, Chun Jason Xue, and Hong Xu. Netkernel: Making network stack part of the virtualized infrastructure. *IEEE/ACM Transactions on Networking*, 2021.

[16] Zhixiong Niu, Hong Xu, Dongsu Han, Peng Cheng, Yongqiang Xiong, Guo Chen, and Keith Winstein. Network stack as a service in the cloud. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks*, pages 65–71, 2017.

[17] Target groups for your application load balancers. `https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-target-groups.html`. Accessed: 2023-01.

[18] DW Cearley, D Scott, J Skorupa, and TJ Bittman. Top 10 technology trends, 2013: cloud computing and hybrid it drive future it models. *Gartner, February*, 2013.

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling

innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[20] OpenFlow Switch Specification. Version 1.0. 0 (wire protocol 0x01). *Open Networking Foundation*, 2009.

[21] Benoit Claise. Cisco systems netflow services export version 9. Technical report, 2004.

[22] Mea Wang, Baochun Li, and Zongpeng Li. sflow: Towards resource-efficient and agile service federation in service overlay networks. In *24th International Conference on Distributed Computing Systems, 2004. Proceedings.*, pages 628–635. IEEE, 2004.

[23] Teemu Koponen, Keith Amidon, Peter Balland, Martín Casado, Anupam Chanda, Bryan Fulton, Igor Ganichev, Jesse Gross, Paul Ingram, Ethan Jackson, et al. Network virtualization in multi-tenant datacenters. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 203–216, 2014.

[24] Hyper-v virtual switch. `https://technet.microsoft.com/en-us/library/hh831823(v=ws.11).aspx`. Accessed: 2023-01.

[25] Zhong-Zhen Wu and Han-Chiang Chen. Design and implementation of tcp/ip offload engine system over gigabit ethernet. In *Proceedings of 15th International Conference on Computer Communications and Networks*, pages 245–250. IEEE, 2006.

[26] Krishna Kant. Tcp offload performance for front-end servers. In *GLOBECOM'03. IEEE Global Telecommunications Conference (IEEE Cat. No. 03CH37489)*, volume 6, pages 3242–3247. IEEE, 2003.

[27] Douglas Freimuth, Elbert C Hu, Jason D LaVoie, Ronald Mraz, Erich M Nahum, Prashant Pradhan, and John M Tracey. Server network scalability and tcp offload. In *USENIX Annual Technical Conference, General Track*, pages 209–222, 2005.

[28] Ruining Chen and Guoao Sun. A survey of kernel-bypass techniques in network stack. In *Proceedings of the 2018 2nd International Conference on Computer Science and Artificial Intelligence*, pages 474–477, 2018.

[29] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. *ACM SIGCOMM Computer Communication Review*, 45(4):537–550, 2015.

[30] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan MG Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, et al. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 514–528, 2020.