Network Performance Influences of Software-defined Networks on Micro-service Architectures

Axel Busch
axel.busch@ibm.com
IBM Germany Research & Development
Boeblingen, Germany

Martin Kammerer
martin.kammerer@de.ibm.com
IBM Germany Research & Development
Boeblingen, Germany

ABSTRACT

Modern business applications are increasingly developed as micro-services and deployed in the cloud. Due to many components involved micro-services need a flexible and high-performance network infrastructure. To ensure highly available and high performance applications, operators are increasingly relying on cloud service platforms such as the OpenShift Container Platform on Z. In such environments modern software-defined network technologies such as Open vSwitch (OVS) are used. However, the impact of their architecture on network performance has not yet been sufficiently researched although networking performance is particularly critical for the quality of the service. In this paper, we analyse the impact of the OVS pipeline and selected OVS operations in detail. We define different scenarios used in the industry and analyse the performance of different OVS configurations using an IBM z14 mainframe system. Our analysis showed the OVS pipeline and its operations can affect network performance by up to factor 3. Our results show that even the use of virtual switches such as OVS, network performance can be significantly improved by optimizing the OVS pipeline architecture.

CCS CONCEPTS

• **Networks** → **Network performance analysis**; *Network management*; Programmable networks.

KEYWORDS

micro services, virtual switches, openshift on z, performance, openflow, open vswitch

ACM Reference Format:
Axel Busch and Martin Kammerer. 2021. Network Performance Influences of Software-defined Networks on Micro-service Architectures. In *Proceedings of the 2021 ACM/SPEC International Conference on Performance Engineering (ICPE '21), April 19–23, 2021, Virtual Event, France.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3427921.3450236

1 INTRODUCTION

Flexible software services are nowadays increasingly designed based on the concept of micro-service architectures. Micro-services have the advantage, among other things, of functionally representing small, self-contained components to be used in broader contexts. These services are made available via interfaces to the outside world to be used for instance in Native Cloud Applications (NCA).

Smaller components providing their services via interfaces advantage in being more fail-safe and scalable. These services are typiclly running on container technologies to be used in cloud environments. Container technologies are often used as the basis for deploying services in separate resource areas, monitoring their availability and scaling them. However, container technologies and their possibilities for (automatic) monitoring and scaling create new requirements, especially for the cloud provider's network infrastructure. When running containers with different services on the same physical resources, various aspects have to be considered such as managing network security.

Both requirements on network infrastructure and security can be implemented and automated using Software Defined Networks (SDN). SDNs introduce a virtual network layer providing the automatic setup of subnets for network isolation of services, control of packet flow within subnets, load balancing of resources, automatic assignment of IP addresses to containers or providing security functions such as anti-ARP spoofing.

Such requirements tend to influence each other. Some of them can influence each other in a negative manner. Let us consider operationalized quality requirements such as security features. Such requirements have been shown their negative influence e.g. on usability and performance [2]. Additional virtual layers of the SDN, however, introduce overhead in terms of CPU time and thus influence reponse time and throughput. Especially in micro-service architectures with many interdependent services, the latency of the individual components adds up quickly and thus leads to higher response times of the entire service. Therefore the latency (and also the throughput) of each individual component is particularly critical to meet quality requirements regarding service response time.

The RedHat OpenShift Container Platform[1] (OCP) enables the automatic deployment of container-based services in the cloud. OpenShift's SDN provides a variety of features to ensure network isolation and automatic network management. One of the central units is the Open vSwitch[2] (OVS), which introduces a virtual network switch on top of the phsyical network layer to dynamically manage networks and coordinate data flows. The additional layers and features introduce overhead and thus can increase response time and reduce throughput. Therefore in performance critical services a trade-off must be made between features and performance.

ICPE '21, April 19–23, 2021, Virtual Event, France

ACM ISBN 978-1-4503-8194-9/21/04…$15.00
https://doi.org/10.1145/3427921.3450236

[1] https://www.openshift.com/
[2] https://www.openvswitch.org/

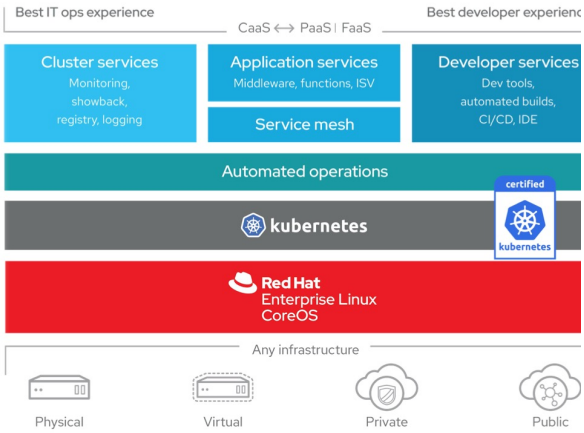

Figure 1: RedHat OpenShift Container Platform archit ture overview [5]

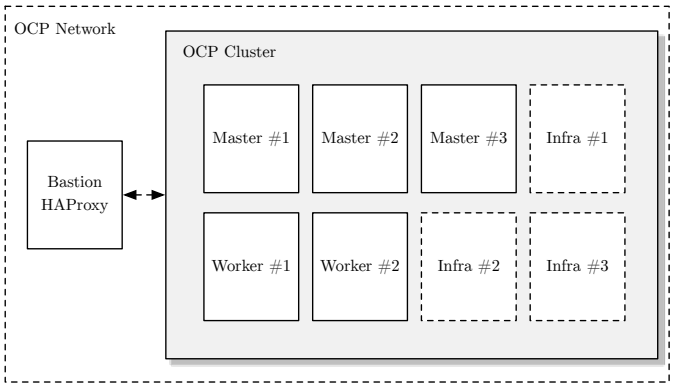

Figure 2: OCP nodes setup schematical overview

In this paper, we analyse the influence of the OVS architect used in OpenShift's SDN and some of its features on the netw latency and throughput. As our contribution we demonstrate detail how the OVS architecture of typical SDN setups influe performance of network workloads, which can be used as a basis drawing conclusions about the performance of micro-services. hardware and software stack we use a state-of-the-art mainfra IBM system z14 running uperf[3]. uperf is an in industry widely u network performance analysis tool, used as load driver for netw performance measurements.

The remainder of this paper is as follows: With Section 2 we troduce OCP and its internal architecture. In Section 3 we introduce software-defined networks with the focus on OVS used in OCP4. Section 4 describes the experiment setup. We introduce infrastructure and software setup as well as workloads and metrics used for the analysis. In Section 5 we perform the baseline analysis and continue with the performance analysis of the Open vSwitch and OpenFlow, its features and thus the flow of data packets through the pipeline. Section 6 presents related work, while we conclude in Section 7 and present future work.

2 REDHAT OPENSHIFT CONTAINER PLATFORM

The OpenShift Container Platform orchestrates cloud technologies enabling on-premise platform services. It allows and supports typical tasks such as the development, deployment, management, and operation of cloud applications without having the expert knowledge around container technologies and the management and orchestration of Kubernetes services. This section introduces the software architecture and hardware architecture of OCP recommended by RedHat. Large parts of the description were collected through expert interviews, domain knowledge or workshops or from referenced sources.

[3]http://uperf.org/

2.1 Software Architecture

Figure 1 shows an overview about the OCP software architecture. The architecture is comprised of 5 layers. These layers enable the 4 core tasks mentioned before. The developer services support the software development with automated builds and CI/CD build pipelines. Different programming languages can be combined and different core repositories can be used.

Using Kubernetes and various automated operations, i.e. layer 2 and 3, applications are automatically fetched from the code repositories, built and rolled out using containers. When the application is running, it is automatically scaled to handle peak loads and replicated to improve service reliability.

Layer 1 is about managing the system: Resources such as physical machines, virtual machines, or resources provided by public or private clouds are automatically managed and the software infrastructure is set up. In addition, infrastructure services are automatically installed and configured, such as the virtual network (OpenShift-SDN).

Various cluster services and application services support the operation of the cluster and applications by monitoring and logging of software and hardware. The network and its topology are configured so that various network policies and services are automatically available at network level.

2.2 Hardware Architecture

OCP requires several machines to provide its services. Figure 2 shows a high level overview of the OCP nodes setup as recommended by RedHat. At least 3 so-called master nodes and at least 5 worker nodes are required. The *master* nodes can also be used as *worker* nodes at the same time, resulting in a minimum requirement of 2 additional worker nodes. However, to improve performance master nodes should not be used as worker nodes at the same time. For optimal performance there is a third class of nodes namely the *infrastructure* nodes running CPU and memory intensive services such as the monitoring services or router. In the following we describe the types of nodes in detail:

- *Master nodes:* Master nodes must be replicated for load balancing and availability purposes. Therefore, 3 master nodes are required for running OCP. The core components of the

OCP cluster run on the master nodes. These components are responsible for configuring and managing the cluster, and carry out the deployment and replication of applications on worker nodes. One of the core components is *etcd*. etcd is a database that stores the state of the master nodes. The state stored in etcd is used by other services to synchronize their own state. [8]

- *Worker nodes:* On worker nodes OCP schedules user applications and services. Resources spent for worker nodes are therefore critical in terms of the performance of services. OCP requires at least two dedicated machines running as worker nodes. If no (optional) infrastructure node is present in the OCP cluster, worker nodes take over the role of the infrastructure nodes. However, since infrastructure services are resource intensive this may slow down the performance of the application containers.
- *Infrastructure nodes:* Infrastructure nodes run all the infrastructure containers such as monitoring services, routers or registry services. Especially the monitoring services are CPU and memory intensive. Therefore, when running the cluster without infrastructure nodes these resource intensive containers place an additional resource demand on the worker nodes. This additional load may slow down the workers. Infrastructure nodes can reduce that effect. However infrastructure nodes are optional, if possible, at least 3 of them should run in the cluster.

In addition to the previously mentioned master, worker, and infrastructure nodes, OCP needs a load balancer in front of the cluster. All traffic and access to the cluster (e.g. by command line interface) is handled by the load balancer. Either hardware load balancers or software load balancers such as HAProxy[4], i.e. a high performance TCP/HTTP load balancer can be used. HAProxy requires a dedicated machine running outside the cluster. This machine is called *Bastion* node. The Bastion node serves as a central access point, a perimeter network access point for incoming and outgoing traffic as well as access point for the configuration of the cluster.

3 ROLE OF SOFTWARE-DEFINED NETWORKS

The term software-defined network has been developing since 1996, originally used to describe the user-managed control of forwarding packets to network nodes. Since then, the term and the concept SDN have been evolved and adapted to the requirements of modern networks.

Network configurations and installations became more and more complex, difficult to understand and therefore cost expensive. Approaches to simulate network topologies became more and more complex and required special knowledge [21]. Recently, functional and quality requirements such as highly dynamic networks coupled with container technologies, resource-efficient scaling of hardware and service replication to improve reliability have made traditional network concepts impossible. In addition containers of cloud services come and go dynamically and allowed routes, i.e. the logical paths between containers and the outside world, change over time. Such flexible networks necessitate an automated, dynamically adaptive network layer.

[4]http://www.haproxy.org/

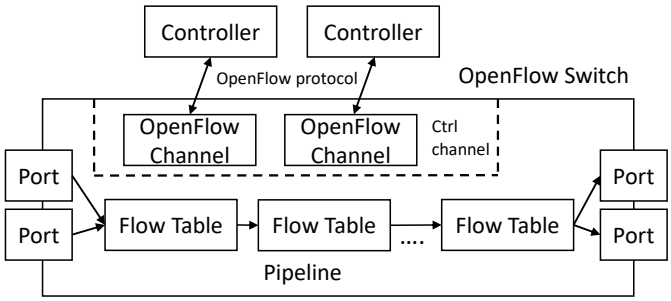

Figure 3: Schematical representation of OpenFlow switch

Modern SDN technologies fill this gap. They allow highly dynamic and flexible network topologies, which can be automatically adapted to the conditions of the current cluster configurations (e.g., with the help of Open vSwitch). In addition, new protocols, such as OpenFlow, allow rules to be defined to control the packet flow [16]. This means that services previously defined at application level, such as firewalls [15], can be implemented directly in the network layer. However the concept of SDN is highly driven by different implementations such as Open vSwitch. Using the Open Network Foundation's SDN [18] definition we derive four key features of SDNs:

(1) *Separation of networking concerns:* Data plane is separated from control plane what means forwarding of packets and control of packet flow is separated.
(2) *Controller:* Centralized controller manages and configures the switch.
(3) *Abstraction:* Network infrastracture layer separated from application layer.
(4) *Application layer functions:* Application layer functions such as firewalls can be integrated directly into network layer.

These features automatically allow networks to dynamically adapt to changing environments. In the following we introduce one of the main components of the OpenShift-SDN namely the Open vSwitch implementing the OpenFlow switch.

3.1 OpenFlow Switch

Figure 3 shows a schematical representation of the OpenFlow switch. The main parts relevant for this paper are the *control channel* and the *pipeline*. In the *control channel* the controller communicates with the OpenFlow channel via the OpenFlow protocol. Through this connection the controller manages and configures the OpenFlow switch. There can be several controllers and several switches configured by the controller for switching in distributed networks.

The *pipeline* is comprised of flow tables. Each packet passes through the pipeline from incoming ports to outgoing ports. Each table contains any number of rules that are applied to the packets one by one. The rules define how packets are forwarded to the next pipeline level. By applying rules one by one the packet finds its way to its destination. To keep rules updated the OpenFlow channel updates the OpenFlow rules in the flow tables of the pipeline.

3.2 OpenFlow Rules

OpenFlow rules determine how packets should be routed through the switch. Rule 1 shows the example of an OpenFlow rule. An OpenFlow rule is comprised of three parts namely *priority* (green), *match* (orange), and *action* (blue). The key word *table* indicates the flow table in which the rule is applied.

$$table = 0, priority = 100, nw\_src = 10.0.0.1, \\ nw\_dst = 10.0.0.2, ct\_state = -trk, action = ct(table = 1) \quad (1)$$

In the following we describe the properties of OpenFlow rules in detail:

- *Priority:* Priority determines the order in which rules are applied to incoming packets. Any number of rules with different priorities can be placed in a flow table. The rules with highest priority that matches the packet is applied to the packet. A higher value means a higher priority.
- *Match:* Match defines properties a packet must satisfy so that a rule can apply. The match section can be defined from a very generic expression to a very specific expression.
- *Action:* Action defines the activities applied to the packets on match.

3.2.1 OpenFlow Match. The *match* section can be comprised of several fields to allow a fine-grained definition of OpenFlow rules. The OpenFlow switch specification defines more than 40 fields on the match of packets [18]. This paper focusses on packet source *nw_src* and destination *nw_dst*, incoming port *in_port*, and connection tracking state *ct_state*. Fields important for this paper are desribed in the following:

- *nw_src & nw_dst:* nw_src and nw_dst define addresses from which packets enter and leave the switch respectively. With this field packet flows can be controled on service granularity.
- *in_port:* in_port defines on which incoming port of the virtual switch the packet arrives. With this field packet flows can be controlled on machines/nodes or subnet granularity.
- *ct_state:* ct_state is a feature implemented by OVS. It supports tracking of packets on stateful or stateless protocols. ct_state supports the tracking for new, already existing, and related (to an existing) connections. Further reply (i.e. the way back), invalid and already tracked packets can be matched.

3.2.2 OpenFlow Action. The *action* section defines what to do with packets or connections matching the rule. The OpenFlow switch specification defines more than 25 fields on the action on matching packets. For this paper *connection tracking* changes (ct(..)) and goto actions are relevant:

- *Contrack action:* The ct action activates the connection tracker that can be used to match on the TCP, UDP, ICMP state of connections. The following arguments are relevant for this paper:
  - *commit:* Commits the connection of a packet in the connection tracker over the lifetime of the package.
  - *table=number:* Creates a copy of the current packet in the specified table and enables connection tracking for that packet.

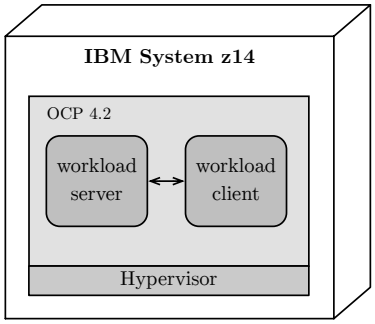

Figure 4: High level overview of experiment setup

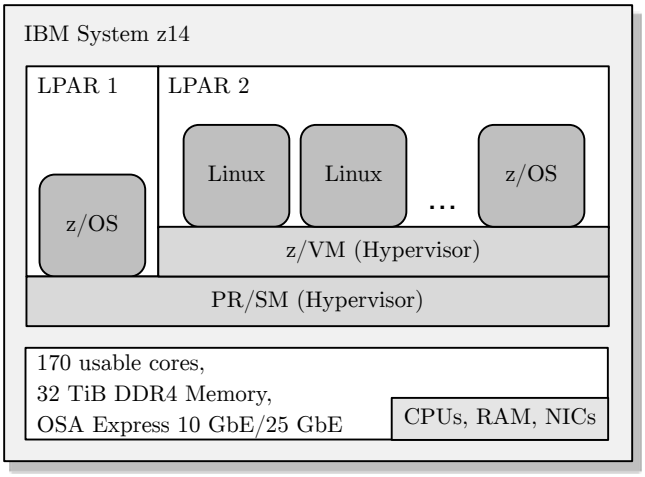

Figure 5: Schematical representation of IBM z14 mainframe

- *Goto action:* Goto action copies packets in the specified table without connection tracker involved.

4 EXPERIMENT SETUP

The setup of the experiment is designed to analyse the network performance of an OpenFlow switch in modern cloud environments and to determine the costs of this technology in terms of performance. This results in several requirements regarding hardware, software and workloads used.

Figure 4 shows a high level overview of the setup. We use an IBM z14 mainframe system, as state of the art cloud platform. Our software stack is RedHat's OpenShift Container Platform (OCP) 4.2 on Z. We use a benchmark as workload driver to simulate different network workloads, whose network patterns are representative for micro-service architectures. The benchmark is running in two instances (one as client and one as server instance) in (different) containers inside OCP. OCP runs in an environment virtualized in multiple stages by several hypervisors.

4.1 Infrastructure Setup

Figure 5 shows a schematical representation of the infrastructure setup [3]. To run our OCP 4.2 cluster we use an IBM z14 mainframe system equipped with a total of 170 cores usable for running workloads. In total the machine comes with 32 TiB DDR4 memory. Each core has a clock speed of 5.2 GHz. The processing speed of the instructions can be further accelerated with simulataneous multithreading (SMT) and single instruction multiple data (SIMD). From the network side the z14 comes with several different Network Interface Cards (NICs). Examples are NICs from Mellanox and IBM Open Systems Adapter-Express (OSA-Express) devices with 10 GbE and 25 GbE respectively [12]. In our setup, we use OSA-Express devices.

System Z is a highly virtualized system to make the best possible use of the resources available. In the case of our OCP setup, two virtualization levels are used, namely the *PR/SM hypervisor* and the *z/VM hypervisor*.

4.1.1 PR/SM Hypervisor. The *PR/SM* hypervisor enables logical partitioning of the central processor complex, i.e. the resources of a Z mainframe. A logical partition (LPAR) contains several CPUs, memory and I/O paths. Furthermore it can be determined whether processing resources are available exclusively or shared. In general applications cannot be run natively but only on LPARs. In LPARs different operating systems can be installed, such as the Z mainframe operating system z/OS, a Linux on IBM Z adapted for the Z mainframe from different distributors or other hypervisors such as z/VM [11].

4.1.2 z/VM Hypervisor. z/VM allows the virtualization of hundreds to thousands of virtual servers within an LPAR. A common use case for the z/VM hypervisor is therefore the consolidation of servers. New instances of virtual servers can be quickly installed, which are particularly necessary for cloud infrastructures. The z/VM hypervisor can virtualize a mix of operating systems such as z/OS or Linux on Z in parallel.

z/VM provides a virtual network that is used to connect the virtual servers, i.e. z/VM guests, to each other and to the outside world. Especially for the communication between the virtual servers, there is no additional load on the physical network hardware, e.g. NICs. For operation, OCP requires several server instances, which we virtualize in our infrastructure with z/VM.

4.1.3 Network. In our infrastructure setup, we have two types of networks: the z/VM VSWITCH included in z/VM, a virtual switch for all guests to communicate with each other, and physical NICs for external communication. The z/VM VSWITCH is connected to the NIC to enable external communication, i.e. outgoing and incoming network traffic from and to the OCP cluster. z/VM can use a physical NIC either shared or in an exclusive manner.

4.2 Cluster Configuration

Figure 6 shows a schematical representation of the machine configuration. The graphic shows two logical parts: On the left side there is the z/VM LPAR with 5 z/VM guests. The cluster has three master nodes and two worker nodes. The operation system of each node is a CoreOS 4, a lightweight operating system for cloud services based on RedHat Enterprise Linux (RHEL) 8.1. In addition, the z/VM VSWITCH is set up within z/VM as a virtual network layer that virtualizes the OSA5 NIC. This z/VM VSWITCH virtualization has the advantage that traffic within the OCP network, i.e. between master nodes or worker nodes, does not put load on the physical NIC and is handled in software. Outgoing traffic is forwarded from the z/VM VSWITCH via the OSA5 NIC to the Bastion node.

The Bastion node (right) is a RHEL 8.1 installation on a separate LPAR installation. This system serves among other things as load balancer for the cluster network. For load balancing we use the software solution HAProxy. The Bastion node uses a dedicated OSA6 card. All traffic out of the cluster and into the cluster is routed via the Bastion node. For better performance we have chosen a higher performance OSA6 card since we expect much traffic passing the Basion node. This ensures that there is no resource contention due to the Bastion node's NIC.

4.2.1 LPAR Setup. Figure 6 shows two LPARs involved in the machine configuration. The LPARs are configured with the following hardware resources:

- *z/VM LPAR:* The LPAR with the z/VM installation running 5 OCP guests can use 28 cores with SMT resulting in 56 virtual cores. The z/VM LPAR can use in total 96 GiB memory. Each z/VM guest running an OCP master or worker node can use parts of these resources. Each master node is configured so that it uses 4 cores each and 16 GiB memory. Worker nodes can use 8 cores and 32 GiB memory each.
- *Bastion LPAR:* The Bastion LPAR is equipped with 8 cores and 32 GiB memory. As mentioned before it runs a RHEL 8.1 as operating system.

4.3 Open vSwitch & OpenFlow Configuration

Open vSwitch and OpenFlow are the main parts of the OpenShift-SDN. OVS defines a bridge where all virtual cables of the containers are plugged. The bridge is running in a so called fail-safe mode *secure*. Secure fail-safe mode means the SDN controller takes care about keeping OpenFlow rules up to date and consistent.

The SDN-controller defines several pipeline levels to route traffic inside the cluster. In our scenario OVS defines six pipeline levels each packet has to go through in order to reach the packet destination.

Figure 7 shows the six pipelines schematically. OVS pipeline levels are organised as tables. *Table 0* is the entry point all packets arriving at the switch. Then they are forwarded according to the rules they match. By applying rules to packets they find their way through the pipeline and finally to their destination. The table identifier such as *table 0* or *table 20* can be freely defined.

In our setup the SDN-controller injects about 120 OpenFlow rules in the pipelines. For our scenario, about 10 rules are relevant (see Figure 7) and apply to packets sent from client containers to server containers on the same node. The rules are shown in the following:

$$table = 0, priority = 300, ct\_state = -trk, ip, \\ actions = ct(table = 0) \quad \text{(Rule 1)}$$

$$table = 0, priority = 100, ip, actions = goto\_table : 20 \quad \text{(Rule 2)}$$

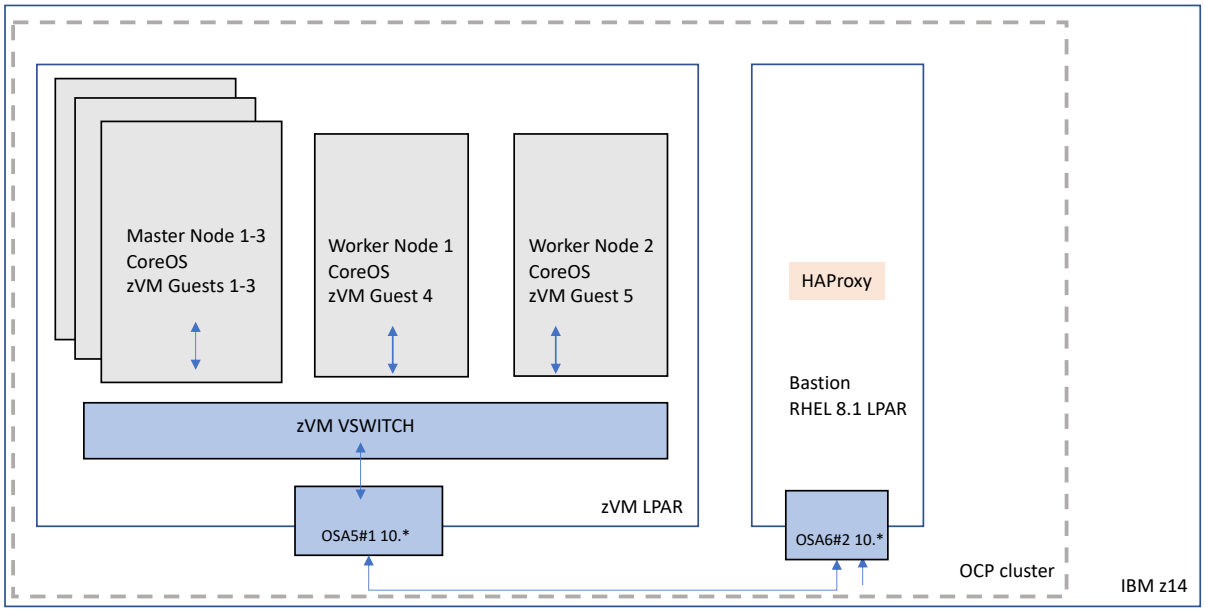

Figure 6: Overview of machine configuration

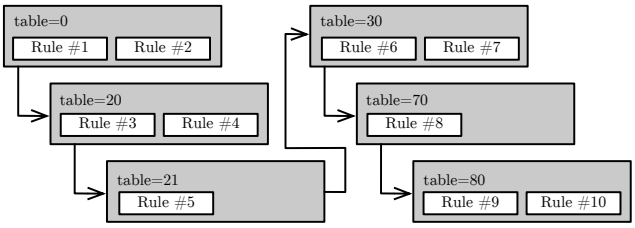

Figure 7: Overview of OVS pipeline and rules

$$table = 20, priority = 100, ip, in\_port = 5, nw\_src = 10.131.0.4, actions = load : 0- > NXM\_NX\_REG0[], goto\_table : 21 \quad \text{(Rule 3)}$$

$$table = 20, priority = 100, ip, in\_port = 6, nw\_src = 10.131.0.5, actions = load : 0- > NXM\_NX\_REG0[], goto\_table : 21 \quad \text{(Rule 4)}$$

$$table = 21, priority = 200, ip, nw\_dst = 10.128.0.0/14, actions = ct(commit, table = 30) \quad \text{(Rule 5)}$$

$$table = 30, priority = 300, ct\_state = +rpl, ip, nw\_dst = 10.131.0.0/23, actions = ct(table = 70, nat) \quad \text{(Rule 6)}$$

$$table = 30, priority = 200, ip, nw\_dst = 10.131.0.0/23, actions = goto\_table : 70 \quad \text{(Rule 7)}$$

$$table = 70, priority = 100, ip, nw\_dst = 10.131.0.4, actions = load : 0- > NXM\_NX\_REG1[], load : 0x5- > NXM\_NX\_REG2[], goto\_table : 80 \quad \text{(Rule 8)}$$

$$table = 80, priority = 200, ct\_state = +rpl, ip, actions = output : NXM\_NX\_REG2[] \quad \text{(Rule 9)}$$

$$table = 80, priority = 300, nw_src = 10.131.0.5, ip, actions = output : NXM\_NX\_REG2[] \quad \text{(Rule 10)}$$

4.3.1 Rule 1. Enables connection tracking for all packets that are not yet monitored by the connection tracker and copies the packet again to *table 0*.

4.3.2 Rule 2. Re-routes all packets in *table 20*.

4.3.3 Rule 3 & 4. Match on packets coming from specific ports and specific source addresses, redirect them to *table 21* and load ports into the appropriate registers for later use.

4.3.4 Rule 5. Matches on all packets to be transferred to subnet 10.128.0.0/14. The packets are committed on the connection tracker and forwarded to *table 30*.

4.3.5 Rule 6 & 7. Matches all packets with connection tracking state *+rpl*, i.e. packets that travel in the opposite direction than initiated by the connection and originally routed to subnet 10.131.0.0./23. A NAT is performed in connection tracker (NAT is necessary because the packet is traveling in the opposite direction and the packet headers need to be updated) and transfers the respective packets to table 70.

4.3.6 Rule 8. Loads ports (source and sink) for a given destination address into the corresponding registers and forwards packets to *table 80*.

4.3.7 Rule 9 & 10. Both rules route out of the packets that have been completely handled by the connection tracker and are en route to the destination address or are en route in the opposite direction. Destination port is the port that was previously loaded into the register.

4.4 Workload Driver

In this section we introduce the workload driver and the workloads we use for our analysis. The benchmarks and workloads we use help to understand typical scenarios from industry environments and also to evaluate high load scenarios.

4.4.1 Uperf Benchmark. The Unified Performance Tool for Networking (uperf), an open source software, is a network performance measurement tool that supports execution of workload profiles and offers a variety of network protocols and connection setups.

A typical measurement setup consists of a uperf client and a uperf server. Uperf has a powerfull workload definition language. The language is used to define fine grained workload profiles in order to build complex workload patterns to be able to model network responses of typical real-world applications. Using the workload definition language performance analysts define workloads that contain information like protocol, target system, amount of processes or threads, number of connections, duration and data sizes. The server will be put into listen mode and then the client initiates the workload according to this profile.

In this paper we use two main types of workloads namely *request-response* and *streaming*. While request-response simulates packet round trips sending and receiving a certain amount of data, e.g. a typical web server network workload, while streaming workloads are considered to be unidirectional, e.g. traffic from a content delivery network [10].

4.4.2 Request-Response Workloads. The transactional profile is set up such that a request of x bytes will be sent from the client to the server and a response of y bytes will be sent back from the server to the client. This request-response workload is repeated for the specified duration time and number of connections. The connections stay open during the entire test period.

In the profile for smaller transactions, which is typical for web server applications, we set the parameters to request 200 byte, response 1000 byte (TCP), number of connections to 50 or 250 and duration to 5 minutes.

In another profile for bigger transactions that could represent database queries we keep all parameters as described for small transactions except the response parameter which we increase to 30 KiB, cf. [13].

4.4.3 Streaming Workloads. The traffic of the streaming profile is passing in one direction only. Possible directions are either writing data to the server or reading data from the server. The parameters define a data direction (i.e. read or write), a counter value, and a data size of certain bytes. The streaming workload is repeated for the specified duration time and number of connections. The connections stay open during the entire test period.

We define the streaming profile with data transfer direction as write, count to 640, data size to 30 KiB, number of connections to 50 or 250, and duration to 5 minutes, cf. [13].

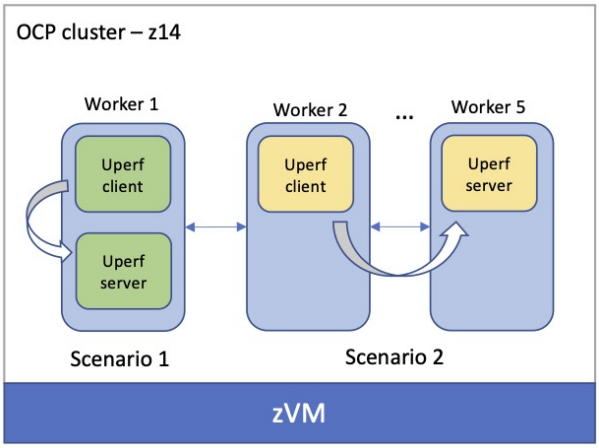

Figure 8: Overview of experiment scenarios

4.5 Metrics

The uperf benchmark provides a variety of throughput, counting, timing, and accompanying resource utilization data as results after successful execution.

For transactional workloads (request-response) we chose transaction times or latency as most important performance result, which measure how long it takes to finish one transaction completely. Similarly we could have chosen number of transactions per second. Throughput in Gbps is less interesting in this scenario as smaller transactions will not overload the network capacity of the network hardware. The result data is taken from the uperf output, *Txn*, column *avg*.

Typically, in micro-service architectures services are divided into several parts. In clusters such as OCP, these micro-services are therefore distributed in separate container instances. In order to deliver the service, many parts must work together as latencies therefore quickly add up to higher values. Micro-service architectures in particular benefit from low latencies, which is why the latency metric is particularly relevant to evaluate micro-service performance.

For streaming workloads the throughput in Mbps or Gbps as average over the entire measurement duration is one of the key aspects. We do not execute a high number of transactions but transfer the maximum possible amount of data. The result data can be gathered from the uperf output, *Run Statistics*, column *Throughput* [13].

5 PERFORMANCE INFLUENCES OF OPEN VSWITCH

This section introduces the performance influencees of OVS. We introduce the experiment setup, our scenarios and the results of our analysis.

5.1 Experiment Scenarios

Overall, we consider two scenarios. Figure 8 shows both scenarios schematically.

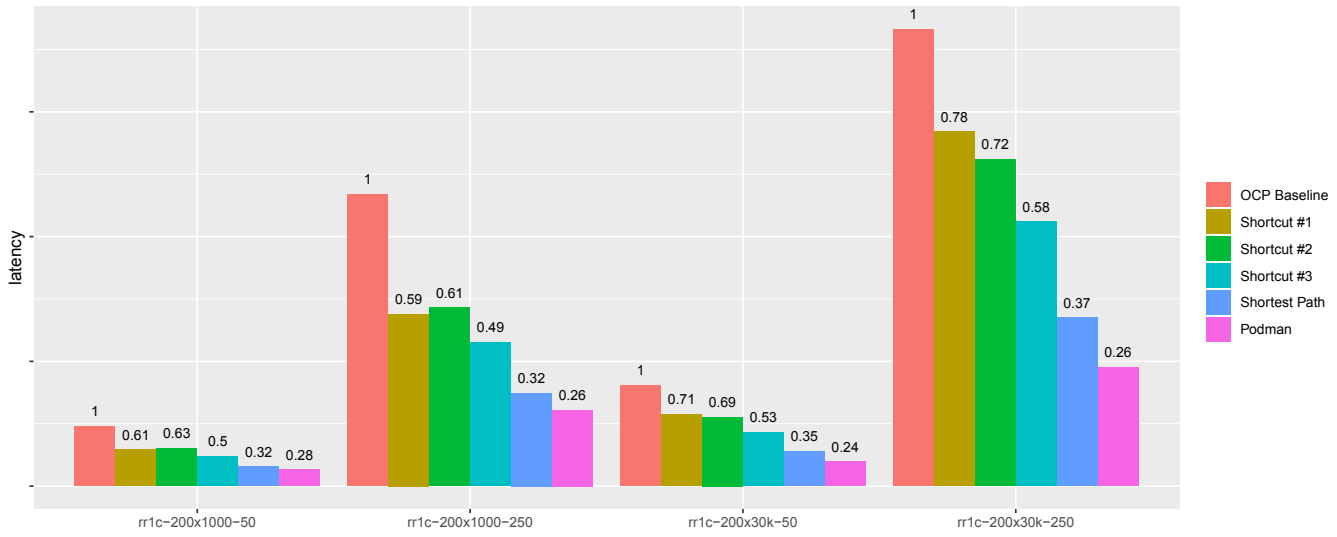

Figure 9: Latency of uperf request response workloads (rr1c), OVS pipeline levels evaluation, intra worker scenario. Each group of bars represents another workload configuration. Each scenario requests 200 bytes, while either 1000 b or 30 kb are responded. 50 or 250 connections are open in parallel (*-50 or *-250). The value of the latency of each shortcut within a group is shown relative to the OCP baseline, i.e. the OCP delivery configuration.

The first scenario considers intra-worker communication, i.e. communication within one worker node. The second scenario analyses the inter-worker, i.e. communication between two worker nodes.

5.2 Scenario I.I: OVS Shortcuts

Scenario I.I focusses on the intra-worker communication. In this kind of communication there is only the Open vSwitch involved. We evaluate six configurations regarding pipeline levels:

5.2.1 Baseline. The first configuration represents the baseline scenario. In the baseline scenario, the packets go through all pipeline levels as described in Section 4.3. By doing so, we evaluate the OCP delivery configuration after a fresh installation.

5.2.2 Shortcut #1. Rule 1 specifies all packets entered the pipeline are reinserted in *table 0* after activating connection tracking. Shortcut #1 cuts this step and skips the reinsertion of all packets in *table 0*. After activation of the connection tracker the packets are forwarded directly to *table 20*.

5.2.3 Shortcut #2. Shortcut #2 extends Shortcut #1 and skips *table 20* in addition.

5.2.4 Shortcut #3. Shortcut #3 skips in addition to Shortcut #2 *tables 21* and *30*. All packets are inserted into *table 70* after they have been processed in *table 0*. A further shortcut inserting packets from *table 0* into *table 80* is not possible (without changing rules) because otherwise ports would not be loaded into the necessary registers.

5.2.5 Shortest Path. To better classify the performance of the shortcuts, we have additionally analysed the performance of the shortest path. In the shortest path, packets arrive in OVS and are forwarded directly to their destination without beeing processed in the pipeline levels. In other words, all pipeline levels are skipped and are forwarded to the destination address.

5.2.6 Podman Loopback. The scenario *Podman Loopback* goes one step further and skips the entire OVS. This scenario shows the theroretical performance without OVS. Packets are transferred via the loopback device. Such an experiment shows the general overhead of OVS.

5.3 Scenario I.II: OVS Operations

Scenario I.II analyses several configuration options regarding connection tracking operations:

5.3.1 Baseline + no connection tracking. We analyse the performance of the connection tracker in detail. The pipeline remains unchanged (baseline), while the connection tracker is deactivated.

5.3.2 Shortcut #3 + no connection tracking. In this scenario, we build on Shortcut #3 and consider the case when connection tracking is disabled.

5.3.3 Baseline + no trk check. The scenario is based on the baseline and does not check whether packets are already monitored by the connection tracker (+trk operation). In this scenario connection tracking is enabled.

5.4 Scenario II: OVS Shortcut

Scenario II focusses on the inter-worker communication. In addition to the OVS the z/VM VSWITCH is involved. We evaluate three configuration options regarding pipeline levels:

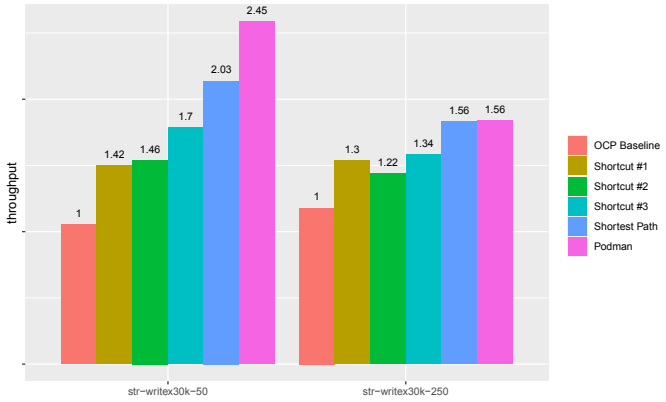

Figure 10: Throughput of uperf streaming workloads (str), OVS pipeline levels evaluation, intra worker scenario, 30 kb request size, 50 and 250 connections in parallel.

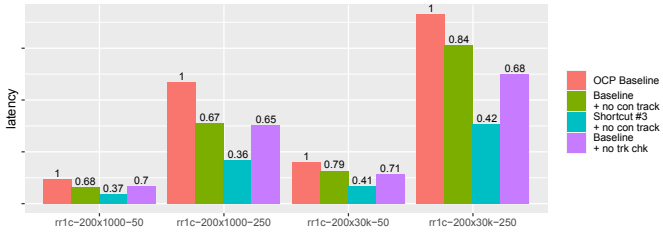

Figure 11: Latency of uperf request response workloads (rr1c), OVS operations evaluation, intra worker scenario, 1000 b and 30 kb request size, 50 and 250 connections in parallel.

5.4.1 Baseline. As before, the baseline corresponds to the standard configuration of OVS configured by the OCP's SDN-controller. The pipeline for the worker to worker communication is three pipeline levels more than the pipeline shown in Section 4.3.

5.4.2 Shortcut #3. Shortcut #3 of this scenario cuts the pipeline similar to Shortcut #3 from Section 5.2. Those three additional pipeline levels make the pipeline longer than in the first scenario.

5.5 Scenario I.I: Results

Figures 9 and 10 show the results of latency and throughput for Scenario I.I (see Section 5.2). In the analysis we focus on two workload configurations introduced in Section 4.4. The two groups of bars on the left in the graphic represent the values for a standard workload that could be found at customer side. The two groups on the right represent a heavy networking workload to stress the infrastructure at most.

Overall, shortening the OVS pipeline shows significant performance improvements by several factors, both in latency and throughput. Latency can be increased by up to a factor of 3.125, while throughput can be improved by up to a factor of 2.01. These improvements are achieved for the shortest path.

Further analysis and comparison of the shortest path with the Podman results show using OVS in general affects performance not much in terms of latency and throughput. Also there is not much additional overhead for the 250 connection streaming workload case. It is also noticeable that Shortcut #2 shows no further, or only an insignificant improvement in latency and throughput compared to Shortcut #1.

An overall analysis shows each pipeline level has a negative impact on latency and throughput. Conversely, a shortening by a level results in an average of 49 % improvement in latency and 45 % improvement in throughput for average leveled workloads. For heavy demanding workloads the improvement is about 41 % for latency and 36 % for throughput.

It is also noteworthy that Shortcut #1 represents a sweet spot within the measurement series. The functionally similar shortcut (compared to the OCP baseline) improves the latency by 41 % in the best case. The throughput can be increased by 42 % in best case, in the worst case by 30 %.

5.6 Scenario I.II: Results

Figures 11 and 12 show latency and throughput for Scenario I.II (see Section 5.3). Overall, the analysis shows that both disabling connection tracking, as well as removing the check for connection tracking (+trk) in *table 0* can improve performance, in some cases significantly. Disabling connection tracking reduces the latency between 23 % and 16 % (compared to baseline). *+trk* operates with an overhead between 29 % and 35 %.

The comparison of Shortcut #3 with and without connection tracking enabled shows interesting results: Shortcut #3 with connection tracking enabled reduces latency by about 50 % (analysing the 200x1000-50 workload), while disabling connection tracking improved latency by about 63 %. Connection tracking therefore operates with an overhead of 13 %.

We can also see differences in throughput: The baseline with connection tracking disabled improves throughput by 10 % (for heavy workloads) and by 28 % for average workloads. Disabling connection tracking check (+trk) improves throughput by between 17 % and 34 %.

The comparison of Shortcut #3 with and without activated connection tracking shows the following results: At average workload, throughput without connection tracking can be improved by 31 % (compared to the baseline).

5.7 Scenario II: Results

Figure 13 shows the throughput for Scenario II (see Section 5.4). A shortening of the pipeline can also improve throughput, namely up to a factor of 1.65 with 250 parallel connections compared to the OCP baseline. The latency, however, can only be improved slightly, namely by a factor of 1.18, when having larger packet sizes (200x30 KiB) and many connections (250 connections).

Thus, communication between nodes benefits less from shortening the pipeline. However, there may be more room for research in this area, as the pipeline is longer than for connections within a node.

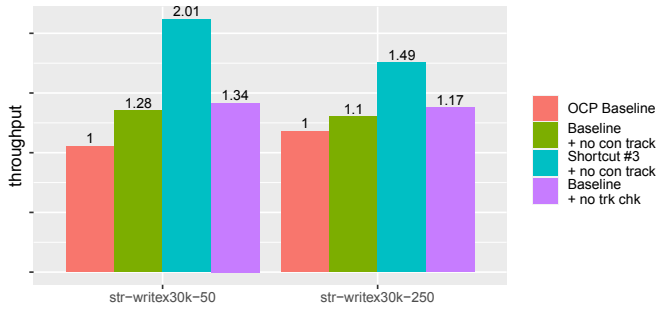

Figure 12: Throughput of uperf streaming workloads (str), OVS operations evaluation, intra worker scenario, 30 kb request size, 50 and 250 connections in parallel.

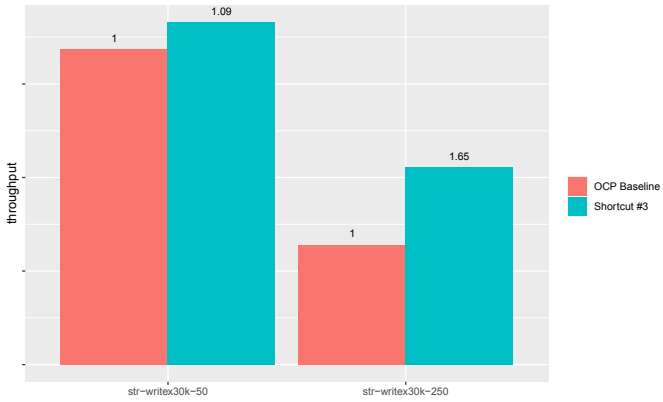

Figure 13: Throughput of uperf streaming workloads (str), OVS pipeline levels evaluation, inter worker scenario, 30 kb request size, 50 and 250 connections in parallel.

5.8 Discussion

The OVS pipeline and OpenFlow rules have a significant impact on the overall performance of typical networking workloads. In particular, we were able to show the performance impact of each pipeline level in an inter-node scenario.

The latency can be reduced by up to a factor 3 by shortening the pipeline. Most of the performance loss is obviously due to the use of deep pipelines. Depending on the architecture, this kind of setup might be mandatory. However, a trade-off decision between architecture with regard to maintainability, comprehensibility, functionality, and performance must be made.

The latency advantage becomes more apparent the more components are required by the service. In micro-service architectures, latencies quickly multiply to clearly noticeable latencies. The reduction in latency for the overall application is correspondingly high when the pipeline architecture is short.

The latency but also network throughput can be improved. The analysis shows an improvement in throughput by up to factor 2 by shortening the pipeline. Thus, the pipeline depth is also relevant for throughput intensive workloads.

The analysis of OVS features such as connection tracking shows the following: In our scenarios, connection tracking causes overhead in the range of 10-20%. Therefore, architects must weigh up whether the feature is necessary or whether it can be switched off for performance-critical applications.

In the intra-node scenario only a small improvement by shortening the pipeline can be observed with respect to latency. However, the network throughput can benefit significantly with up to factor 1.65. Throughput critical services can also benefit from optimizing the pipeline length if the network architecture can be adapted accordingly.

6 RELATED WORK

Most of related work in the field of software defined networks considers the advantages of SDNs in terms of scalability, flexibility, how they enable cloud applications, and what problems SDNs solve compared to traditional network concepts. Such topics are covered and summed up by the survey from Hu et al. [9]. They show advantages but also drawbacks of SDNs. However, the authors do not take a closer look at the network performance of SDNs. They express doubts about the QoS control implemented during their research work.

Rad et al. [19] analyse latency of SDNs in the field of high performance clouds. They study the InfiniBand Low Latency Software Defined Network (integrated in OpenStack) together with SR-IOV, a technology to minimize virtualization overhead. They conclude SR-IOV does only introduce an overhead of a fraction of a millisecond. However, they do not consider OpenFlow SDN rules and pipeline levels introduced by OVS or similar technologies.

Several papers [1],[14] consider high performance computing in general. Gupta et al. [6],[7] consider networking beside other resources. They combine several applications with different computing profiles for more efficient consolidation. During their study, they analyse performance of computing, storage and network resources using OpenStack. They use KVM as hypervisor and use different network virtualization drivers such as rtl8139, eth1000, and virtio-net. However, they do not analyse network performance in isolation but use a mix of workloads to stress several resource types at the same time.

Performance of SDNs and Open vSwitch has been investigated by several other papers [4]. Yan et al. [22] found Vxlan shows comparatibly low performance. Vxlan is used for incoming and outgoing traffic.

Sattar and Matrawy [20] analysed the delay of OVS and build a model for simulation and further analysis purposes. Yang et al. [23] proposed a similar approach to mode OVS performance including the characteristics of the system OVS runs on.

Mian et al. [17] found in their analysis an increased latency when using OVS, but did not investigate the OVS pipeline and operations in detail. Further, they rely on ping for their analysis. Using ping, however, is often not sufficient since packet sizes and number of connections can not be taken into account in required detail and has been shown to significantly influence both latency and throughput.

7 CONCLUSIONS & FUTURE WORK

In this paper, we described in detail the architecture of typical SDN setups using the OpenShift-SDN as an example. We analysed the impact of the SDN architecture on the performance of typical network workloads that are representative for micro-service applications. The analysis showed that complex OVS pipelines can highly influence both latency and throughput. Especially network performance-critical workloads or complex micro-service architectures with a high number of components may require architects to make trade-off decisions regarding pipeline length and performance. Longer pipelines quickly lead to higher latencies by several factors.

Especially our inter-node scenario confirms this statement: each extension of the pipeline length can increase the latency by 49 % on average. Similar results can be derived for throughput: each pipeline level can reduce the throughput by 45 % on average. However, for architects it is worthwhile for their workloads to analyse the pipeline length and OVS features used in more detail. Our analysis revealed at least one sweet spot that already enabled up to 41 % lower latency with only minimal changes in the architecture.

In the intra-node scenario only minor performance improvements could be shown by shortening the pipeline. However, the pipeline is longer and there is still room for research regarding shortening the pipeline. Furthermore, the OVS features have to be considered in more detail.

In addition, there is a need for further research regarding another scenario, namely the analysis including the physical network hardware. The influence of the NIC was not considered, but is relevant whenever network traffic is routed out of the cluster. Furthermore, no functions such as hardware offloading were considered, which could influence the performance in such scenarios too.

ACKNOWLEDGEMENTS

Thank you to all members of the Linux on Z Performance team in Böblingen for fruitful discussions around OpenShift performance and input to this work. Special thanks go to Christian Bram and Michael Stetter for great administrative support around this publication.

REFERENCES

[1] A. I. Avetisyan, R. Campbell, I. Gupta, M. T. Heath, S. Y. Ko, G. R. Ganger, M. A. Kozuch, D. O'Hallaron, M. Kunze, T. T. Kwan, K. Lai, M. Lyons, D. S. Milojicic, H. Y. Lee, Y. C. Soh, N. K. Ming, J. Luke, and H. Namgoong. 2010. Open Cirrus: A Global Cloud Computing Testbed. *Computer* 43, 4 (2010), 35–43.

[2] Axel Busch. 2019. *Quality-driven Reuse of Model-based Software Architecture Elements.* Ph.D. Dissertation. Institut für Programmstrukturen und Datenorganisation (IPD), Karlsruher Institut für Technologie, Karlsruhe, Germany. https://publikationen.bibliothek.kit.edu/1000097163

[3] Axel Busch, Qais Noorshams, Samuel Kounev, Anne Koziolek, Ralf Reussner, and Erich Amrehn. 2015. Automated Workload Characterization for I/O Performance Analysis in Virtualized Environments. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE '15)*. ACM, New York, NY, USA, 265–276. https://doi.org/10.1145/2668930.2688050 Acceptance Rate (Full Paper): 15/56 = 27%.

[4] H. Chen, Z. Qiao, and S. Fu. 2019. Applying SDN based data network on HPC Big Data Computing – Design, Implementation, and Evaluation. In *2019 IEEE International Conference on Big Data (Big Data)*. 6007–6009.

[5] Jaafar Chraibi. 2020. Enterprise Kubernetes with OpenShift. https://www.openshift.com/blog/enterprise-kubernetes-with-openshift-part-one. [Online; accessed 25-Aug-2020].

[6] A. Gupta, L. V. Kalé, D. Milojicic, P. Faraboschi, and S. M. Balle. 2013. HPC-Aware VM Placement in Infrastructure Clouds. In *2013 IEEE International Conference on Cloud Engineering (IC2E)*. 11–20.

[7] A. Gupta, O. Sarood, L. V. Kale, and D. Milojicic. 2013. Improving HPC Application Performance in Cloud through Dynamic Load Balancing. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*. 402–409.

[8] Red Hat. 2019. Kubernetes Infrastructure. https://docs.openshift.com/enterprise/3.0/architecture/infrastructure_components/kubernetes_infrastructure.html. [Online; accessed 25-Aug-2020].

[9] F. Hu, Q. Hao, and K. Bao. 2014. A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation. *IEEE Communications Surveys Tutorials* 16, 4 (2014), 2181–2206.

[10] IBM. 2016. *KVM Network Performance -Best Practices and TuningRecommendations.* Technical Report. International Business Machines Corporation. http://public.dhe.ibm.com/software/dw/linux390/perf/ZSW03346USEN.pdf

[11] IBM. 2019. Processor Resource/Systems Manager Planning Guide, SB10-7169-02, Level 02a. https://www.ibm.com/support/pages/node/6018640

[12] IBM. 2019. Systems Hardware Data Sheet - IBM z14. https://www.ibm.com/downloads/cas/O4VDMBV2. [Online; accessed 03-Sep-2020].

[13] IBM. 2020. IBM Knowledge Center - Uperf. https://www.ibm.com/support/knowledgecenter/linuxonibm/liaag/wkvm/wkvm_c_workload.htm. [Online; accessed 01-Sep-2020].

[14] A. Iosup, S. Ostermann, M. N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. 2011. Performance Analysis of Cloud Computing Services for Many-Tasks Scientific Computing. *IEEE Transactions on Parallel and Distributed Systems* 22, 6 (2011), 931–945.

[15] P. Krongbaramee and Y. Somchit. 2018. Implementation of SDN Stateful Firewall on Data Plane using Open vSwitch. In *2018 15th International Joint Conference on Computer Science and Software Engineering (JCSSE)*. 1–5.

[16] Y. Li and M. Chen. 2015. Software-Defined Network Function Virtualization: A Survey. *IEEE Access* 3 (2015), 2542–2553.

[17] A. N. Mian, A. Mamoon, R. Khan, and A. Anjum. 2014. Effects of Virtualization on Network and Processor Performance Using Open vSwitch and Xen Server. In *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*. 762–767.

[18] ONF. 2012. *Software-Defined Networking: The New Norm for Networks.* Technical Report. Open Networking Foundation. https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[19] P. Rad, R. V. Boppana, P. Lama, G. Berman, and M. Jamshidi. 2015. Low-latency software defined network for high performance clouds. In *2015 10th System of Systems Engineering Conference (SoSE)*. 486–491.

[20] D. Sattar and A. Matrawy. 2017. An empirical model of packet processing delay of the Open vSwitch. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*. 1–6.

[21] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao. 2013. Are we ready for SDN? Implementation challenges for software-defined networks. *IEEE Communications Magazine* 51, 7 (2013), 36–43.

[22] Y. Yan and H. Wang. 2016. Open vSwitch Vxlan performance acceleration in cloud computing data center. In *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*. 567–571.

[23] Runkai Yang, Xiaolin Chang, Jelena Mišić, and Vojislav B Mišić. 2020. Performance Modeling of Linux Network System with Open vSwitch. *Peer-to-Peer Networking and Applications* 13, 1 (2020), 151–162.