hvbench: An open and scalable SDN Network Hypervisor Benchmark

Christian Sieber, Andreas Blenk, Arsany Basta, Wolfgang Kellerer
Chair of Communication Networks
Department of Electrical and Computer Engineering
Technical University of Munich, Germany
Email: {c.sieber,andreas.blenk,arsany.basta,wolfgang.kellerer}@tum.de

***Abstract*—Software-defined networking (SDN) introduces a split between the forwarding plane and the control plane of modern network equipment. Furthermore, OpenFlow, as an open interface for SDN, fosters the development of logically centralized network operation systems (NOS). To allow multiple network operation systems accessing the same network, there is the concept of SDN hypervisors. Little is known about the performance characteristics of SDN hypervisors. Furthermore, there is a lack of scalable and realistic hypervisor benchmark tools capable of emulating dynamic load scenarios. In this paper, we present an extensible and distributed SDN hypervisor benchmarking framework based on flexible statistical request generators. The framework can be scaled out horizontally to multiple compute nodes that are centrally controlled and reconfigured at run-time. We present preliminary measurements of the CPU resource consumption of a hypervisor in a virtual environment. The results show that the performance characteristics of the hypervisor are different for dynamic load scenarios compared to static benchmarks. Furthermore, the results show that for the same overall request rate, multiple NOS increase the CPU load considerable compared to a single NOS.**

I. Introduction

Software-defined networking (SDN) is replacing classical networking concepts of integrated network devices by splitting the forwarding decision making and the actual data forwarding. Decisions are made by a logically centralized network operation system (NOS) (or *SDN controller*). The decisions are pushed in the shape of forwarding rules to the networking devices. The recent success of SDN can be attributed to the OpenFlow protocol, which is an open interface for manipulating the forwarding tables of OpenFlow-enabled networking devices. Additionally, OpenFlow provides the capability to request the features of a device, e.g., number of Ethernet ports, to receive statistics, e.g., number of Bytes received and forwarded, and to update the rules in the forwarding table.

Analogous to server virtualization where hypervisors allow multiple operation systems to share the same physical hardware, SDN hypervisors virtualize the network. For example FlowVisor [1] and OpenVirteX [2] enable multiple networking operating systems to coexist and share the control for the same physical infrastructure. SDN hypervisors use the flow space of OpenFlow forwarding rules, e.g., VLANs, IP address ranges and Ethernet ports, to split a network into different slices and each slice can be associated with a different networking operation system. Hence, an SDN hypervisor is translating rules from the NOS to allow shared access, provide statistics, topology abstraction and isolation. Use cases range from testing a new NOS on a fraction of the traffic to offering different customers the control over their forwarding decisions in a shared infrastructure [3].

978-1-4673-9486-4/16/$31.00 © 2016 IEEE

While the performance of SDN controllers and networking devices receives much attention, less is known about the performance characteristics of SDN hypervisors. All requests from the NOS and all replies or notifications from the devices have to pass through the hypervisor, which makes it a crucial part of the infrastructure. In order to know the overall performance of the system, we have to understand how SDN hypervisors handle different requests types and how the request types relate to each other in terms of computational complexity. Furthermore, it is important to understand if and how the request arrival process and request type distribution has an influence on the performance. In this work, we present an open and scalable hypervisor benchmarking tool which is able to generate dynamic workloads. This is a step towards understanding the performance characteristics of SDN hypervisors in dynamic environments.

A. Challenges

A hypervisor benchmark has to be flexible to cover the wide range of scenarios where SDN hypervisors are deployed. Deployment scenarios can range from single instance deployment to horizontally scaled out hypervisor instances and from heavy use of a single request type to different distributions of request types. Furthermore, in deployment, SDN hypervisors will face varying amount of load and changing demands. According to those requirements, we identified the following shortcomings in current OpenFlow-based SDN benchmarks: 1) They are designed as a single instance application, which limits the scalability; 2) only generate messages with a fixed inter-arrival time, hence do not represent realistic request arrival processes; 3) only generate messages of a fixed type, e.g., flow mods or echo requests, which neglects possible cross-effects between request types; 4) are not configurable at run-time and hence do not allow dynamic scenarios; and/or 5) depend on a real or emulated data plane, hence are restricted to round-trip measurements or have to intercept the connection between hypervisor and data plane.

In this work, we present a hypervisor benchmark tool that tackles the aforementioned shortcomings by 1) designing it as a distributed application with dummy data plane nodes; 2) using statistical probability distribution for message generation; and 3) allowing updates to the configuration on run-time. Subsequently we discuss the design goals of *hvbench* in detail.

B. Design Goals & Features

To aid the evaluation of the aforementioned performance characteristics of SDN hypervisors, *hvbench* offers the following notable features:

- Distributed operation of NOS controllers and data plane nodes on multiple physical machines.
- Using kafka [4] to forward packet in and packet out between the dummy data plane nodes.
- Centrally configured & controlled by using etcd [5] as directory and discovery service.
- Configurable at run-time (adding a NOS, adding nodes, changing generator configuration)
- Extensible message generators based on Poisson process and individual weight of each request type.

The source code and documentation is available on github.com under the MIT licensing terms [6].

The remainder of this paper is structured as follows. In Section II we introduce the background and related work in this field. In Section III we show the architecture of *hvbench* including the components. In Section IV we present preliminary measurement results for one SDN hypervisor. In Section V we conclude this work and outline future work.

II. Background & Related Work

Understanding the performance characteristics of SDN hypervisors is a first step towards SDN control plane resource allocation. In order to allocate hypervisor processing resources to NOS demands, a mapping between the hypervisor processing resources and the control plane request rate is required. Such mapping guarantees that, e.g., enough CPU resources are assigned to each NOS in order to provide the requested performance. The importance of such a mapping and performance evaluation benchmarks have already been shown in [7], [8]. However, the introduced benchmark focuses mostly on the maximum performance of a particular type of request, e.g., flow modifications, and does not consider request mixes or statistical arrival processes. Other existing work on hypervisors has also only provided benchmarks proofing their implementation [1], [2] based on only a few metrics, e.g., flow set-up time.

From the perspective of a controller, SDN hypervisors appear as data plane nodes. From the perspective of a data plane node, SDN hypervisors appear as an SDN controller. Hence, benchmark tools for controllers and data plane nodes are applicable for benchmarking SDN hypervisors. cbench [9], ofcbenchmark [10], SDLoad [11], PktBlaster [12] and the framework presented in [13] emulate different data plane topologies and measure performance metrics such as flow set-up times. Switch benchmark tools such as OFLOPS and OFLOPS-Turbo [14] emulate an SDN controller and measure switch specific performance metrics such as flow table capacity and the latency for rule insertion into the flow table. A comprehensive overview of further controller and data plane benchmarks can be found in [15]. In [16], the authors take a different approach and use parametrized abstract models to describe the performance of SDN controllers. However, the authors do not evaluate the abstract models for SDN hypervisors.

III. Architecture

Figure 1 depicts the architecture of *hvbench*, The architecture consists of four main modules. A configuration and directory service (1), distributed *hvbench* instances (2), a high-throughput messaging bus (3) and host monitors (4). A command-line front-end and Python bindings simplify the usage of the distributed architecture. Each (outer) box in the figure represents a software component which can be distributed to different virtual or physical machines.

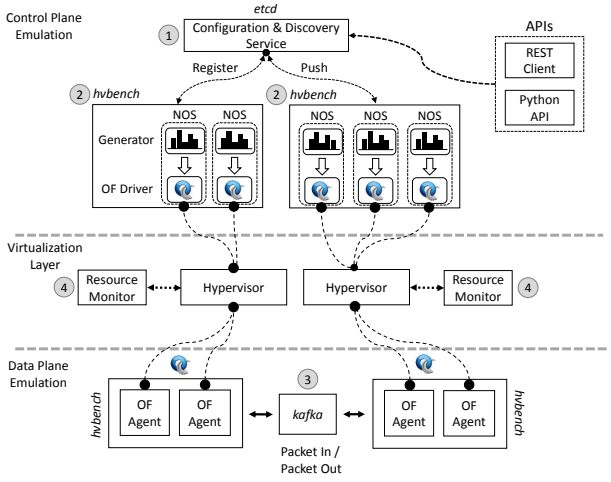

Figure 1. Architectural overview of *hvbench*. Distributed hvbench instances (2) are coordinated by a central configuration and discovery service etcd (1). kafka distributes in/out packets between the data plane nodes (3). Resource monitors collect process performance metrics (4).

hvbench instances are containers for multiple, independent, NOS and/or data plane nodes. When the distributed *hvbench* instances are started, the instances register at the configuration and directory service. We use *etcd* as configuration directory service for its simplicity and support of change notifications. From this service, they receive the run-time configuration, e.g., number of NOS per instance, and report their host configuration, e.g., the host's IP address. The configuration for a specific instance is identified by an ID given to each instances on start-up. Additionally, *etcd* pushes run-time configuration changes, e.g., an update to the average message rate, to the *hvbench* instances and the instances in return report their utilization to the directory service.

A message bus distributes injected data plane packets received from the hypervisor between the data plane nodes (*packet out*). This allows topology discovery via Link Layer

Discovery Protocol (LLDP) packets as used for example by FlowVisor and OpenVirtex. We use Apache *kafka* as message bus because of its low latency properties and scalable architecture. For handling the OpenFlow connections and message parsing, denoted as *OF driver* for the NOS and *OF agent* for the data plane node in the figure, we use a customized version of libfluid [17].

The host monitor uses a database of known SDN hypervisor names to identify and monitor hypervisor processes. From the process it collects performance metrics such as CPU utilization as reported by the underlying Linux kernel.

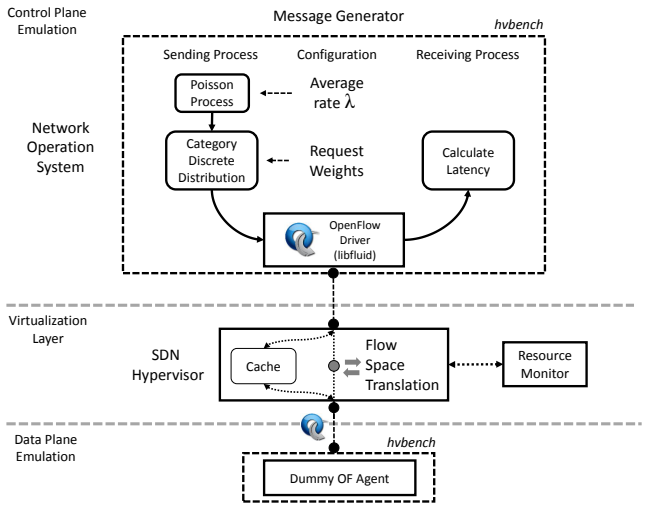

Figure 2. Detailed view of two *hvbench* instance with a single NOS including the message generator, one data plane node and the hypervisor.

Figure 2 gives a detailed overview of a set-up consisting of two *hvbench* instances and the hypervisor. A NOS is described by its OpenFlow listening port and request generator. As of now, two types of request generators are implemented for a NOS. A constant message generator generates requests of one specific type with a specific rate, e.g., 1,000 feature requests per second. A probabilistic generator generates different requests based on a set of weights and an average rate λ. At first, the sending process in the probabilistic generator chooses the point in time of the next message to send based on a Poisson process with exponential distributed inter-arrival times and an average rate of λ. Next, the type of the requests to be generated is determined by a categorical (discrete) distribution based on a set of weights. The average rate and the weight of each request type can be adjusted on run-time.

In the receiving process of the NOS, we measure the latency of requests. We use requests with replies such as echo requests to calculate the round-trip time. Measured one-way latency depends on the quality of clock synchronization when the data plane node and the NOS are not on the same physical machine. The data plane nodes emulate the OF agents and answer requests either with random values (e.g. port statistics) or with pre-configured values, e.g., feature requests.

Next we motivate *hvbench* by comparing measurements of static benchmark scenarios with dynamic workload scenarios.

IV. Measurements

Our preliminary measurements focus on three questions. First, is there a difference in computational complexity between different request types? Second, how does the statistical arrival process compare to the static arrival process with only one request type? And third, does the number of connected NOS influence the CPU load for the same request rate?

In the evaluation of the questions we use the OpenFlow hypervisor FlowVisor [1]. The hypervisor is executed inside an Ubuntu 14.04 environment virtualized by VirtualBox. One i7-4770 CPU core with 3.40 GHz and 2 GB of memory were assigned to the virtual environment. The data plane consists of one data plane node. As CPU load/utilization metric, we use the fraction of time in which the process uses the CPU in system and user mode within a time interval of one second.

For the statistical arrival process we set the following weights for the request type distribution. A probability of 0.2 for each of *flow modifications*, both types of *statistics requests*, and *packet out* messages. A probability of 0.1 for each of *feature requests* and *echo requests*. The request inter-arrival times are distributed exponentially. For the constant arrival process, the inter-arrival times are constant based on the specified average rate.

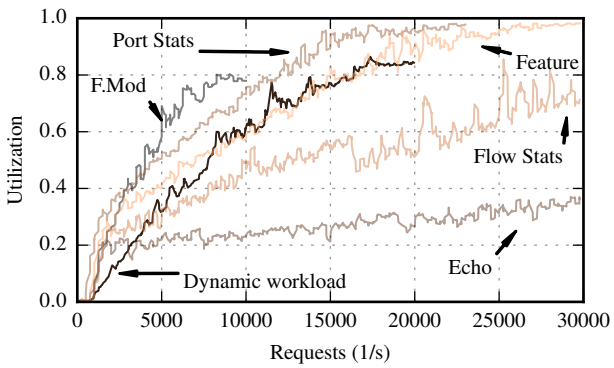

Figure 3. CPU utilization for FlowVisor for increasing request rates for different request types for one NOS. The figure illustrates the varying computational complexity of request types. Each line represents one experiment run.

In the first experiment we measure the CPU load of FlowVisor for increasing request rates up to 30,000 requests per second for different request types generated by one NOS. The average request rate is increased by 250 requests/s every five seconds. For flow modifications and dynamic workload, the current limits of *hvbench* for one NOS are 10,000 and 20,000 requests per second, respectively. The results are shown with a rolling mean with a window size of 5. Figure 3 illustrates one experiment run for each request type. Four observations can be made from the figure. First, there is a sudden rise to 20 % CPU load for all request types at around 1,000 requests per second. For lower request rates, the CPU statistics of the Linux kernel report a utilization of close to 0 %. Second, the slope of the increase in CPU load for higher request rates is different for each request type. For example echo requests are not computational expensive for that hypervisor and therefor

the increase for higher message rates is low. Flow modification requests on the other hand are computational expensive and therefor the CPU load increases fast for higher rates. Third, the behavior for some request type shows irregularities. Most notable here is the trend of the flow statistic requests. While port statistic requests exhibit a sub-linear behavior with no irregularities, flow statistics exhibit a drop of CPU load for about 16,000 requests per second. The drop indicates that the hypervisor handles high flow statistics request rates differently than lower ones. Forth, the requests generated by the Poisson-based generator are computational expensive compared to echo and port statistic requests. Furthermore, the trend for higher request rates is about linear up to 10,000 requests per second.

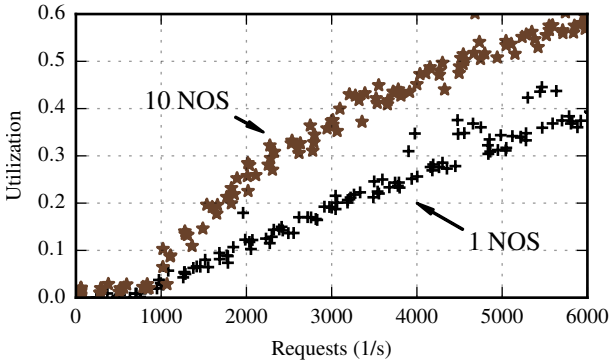

Figure 4. CPU utilization for up to 6,000 requests per second for one NOS and for ten NOS. The increase for one NOS is approximately linear, while the increase for ten NOS is super linear.

Next, we investigate the question whether there is a difference in CPU load between one and multiple NOS for the same request rate. Figure 4 shows the CPU utilization for up to 6,000 requests for one NOS and for ten NOS. The request rate for each of the ten NOS is set to $\frac{1}{10}$ of the request rate of the one NOS in the single NOS configuration. The figure shows the result of two experiment runs, one for one NOS and one for ten NOS, where the request rate was increased by 25 requests per second every five seconds. Repeated measurements show the same behavior. From the figure we conclude that there is a difference between the two configurations in terms of CPU load for a specific request rate. While the CPU utilization for one NOS behaves approximately linear for increasing request rates, the utilization for ten NOS increases super linear. However, starting from a message rate of 2,500, both configurations increase linear. For 6,000 requests per second, there is a difference of about 20 % CPU utilization. As also observed in Figure 3, for up to 1,000 requests per second the CPU utilization stays close to 0 % for both configurations.

To summarize, the results show that there is a difference in computational complexity of the different request types and request type distribution. Furthermore, the processing of the requests becomes less efficient, i.e. results in higher CPU load, if the messages are generated by more than one NOS. *hvbench* is a first step in gaining better insight in the performance characteristics of SDN hypervisors for dynamic workloads.

V. Conclusion & Future Work

SDN hypervisor allow multiple network operation systems (NOS) to operate slices of a network simultaneously. They slice the network using the flow space of OpenFlow and translate all requests from and to the NOS. This makes them a critical part of infrastructure. Therefore, it is critical to understand the performance characteristics of SDN hypervisor for varying workloads. In this work we first highlight the shortcomings of traditional SDN benchmarking tools to adequately mimic realistic deployment scenarios. We present *hvbench*, a flexible and horizontally scalable benchmarking framework for SDN hypervisors. *hvbench* generates request arrivals based on statistical arrival processes and run-time configurable to emulate different deployment scenarios. Preliminary measurements with *hvbench* motivate the need for sophisticated benchmarking tools as the results show that there is a significant differences between static and dynamic benchmarking scenarios. In the future, we extend the performance evaluation with a larger parameter space, e.g. more hypervisors and different hardware configurations. The source code and documentation of *hvbench* is available to the community under the MIT license.

References

[1] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. Mckeown, and G. Parulkar, "FlowVisor: A network virtualization layer," OpenFlow Consortium, Tech. Rep., 2009.

[2] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, and G. Parulkar, "OpenVirteX: a network hypervisor," in *Proc. Open Networking Summit (ONS)*, Santa Clara, CA, Mar. 2014.

[3] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on Network Virtualization Hypervisors for Software Defined Networking," *IEEE Communications Surveys & Tutorials*, pp. 1–32, 2015.

[4] "Apache Kafka," http://kafka.apache.org/.

[5] "etcd," https://coreos.com/etcd/.

[6] "hvbench," https://github.com/csieber/hvbench.

[7] A. Blenk, A. Basta, and W. Kellerer, "HyperFlex: An SDN virtualization architecture with flexible hypervisor function allocation," in *IFIP/IEEE Int. Symp. on Integrated Network Management (IM)*, 2015.

[8] A. Basta, A. Blenk, Y.-T. Lai, and W. Kellerer, "Hyperflex: Demonstrating control-plane isolation for virtual software-defined networks," in *IFIP/IEEE Int. Symp. on Integrated Network Management (IM)*, 2015.

[9] "cbench," https://github.com/andi-bigswitch/oflops/tree/master/cbench.

[10] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries, "A flexible openflow-controller benchmark," in *2012 European Workshop on Software Defined Networking (EWSDN)*, Oct 2012, pp. 48–53.

[11] N. Laurent, S. Vissicchio, and M. Canini, "SDLoad: An extensible framework for sdn workload generation," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014.

[12] "PktBlaster," http://www.veryxtech.com/products/product-families/pktblaster-sdn-software-defined-network-test.

[13] J. Teixeira, G. Antichi, D. Adami, A. Del Chiaro, S. Giordano, and A. Santos, "Datacenter in a box: Test your sdn cloud-datacenter controller at home," in *2013 Second European Workshop on Software Defined Networks (EWSDN)*, Oct 2013, pp. 99–104.

[14] C. Rotsos, G. Antichi *et al.*, "Oflops-turbo: Testing the next-generation openflow switch," in *2015 IEEE International Conference on Communications (ICC)*, June 2015, pp. 5571–5576.

[15] D. Kreutz, F. M. Ramos *et al.*, "Software-defined networking: A comprehensive survey," *proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[16] T. Sato, S. Ata, I. Oka, and Y. Sato, "Abstract model of sdn architectures enabling comprehensive performance comparisons," in *Network and Service Management (CNSM), 2015 11th International Conference on*, Nov 2015, pp. 99–107.

[17] C. R. A. Vidal, E. Fernandes and M. Salvador, "libfluid," http://opennetworkingfoundation.github.io/libfluid/, 2015.