IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 978-1-6654-0926-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/INFOCOMWKSHPS54753.2022.9798351

HVNet: Hardware-Assisted Virtual Networking on a Single Physical Host

Florian Wiedner, Max Helm, Sebastian Gallenmüller, Georg Carle
Department of Informatics, *Technical University of Munich*, Garching near Munich, Munich
{ wiedner | helm | gallenmu | carle }@net.in.tum.de

***Abstract*—Network experiments using real hardware are typically expensive and time-consuming. Multiple solutions exist to reduce costs, in particular network emulation, or simulation of performance metrics. However, each solution impacts the quality of experimental results, in particular concerning realism and precision. We propose HVNet, a novel approach to create virtualized topologies on a single host utilizing real networking hardware. Relying on real hardware, our approach offers realistic network behavior and high-precision measurements. HVNet enables measurements on flexible network topologies avoiding the drawbacks of the alternative solutions. We observed repeatable results with a small error margin and a low impact of the measurement setup on experimental results. Additionally, we compare latency and jitter distributions of HVNet and Mininet setups, observing an improvement factor of up to three orders of magnitude.**

***Index Terms*—Virtual machines, single-root io-virtualization, testbed, emulation.**

I. Introduction

Network experiments using real hardware carry significant costs and restrictions for changing or creating arbitrary topologies. These costs can be mitigated by alternative approaches, including network emulation, e.g., Mininet [1] or simulation such as OMNet++ [2]. However, these alternatives have certain limitations in terms of result quality, abstracting from certain hardware details. For certain experiment types such as latency measurements of software stacks, approaches are desirable that come with few hardware requirements, while providing realistic latency results representative for non-virtualized environments.

In this paper, we propose a Hardware-Assisted Virtual Networking (HVNet) framework running on commercial off-the-shelf hardware to analyze and verify topologies and network services based on a single physical host including two network interface cards (NICs). We propose a way to analyze hardware-level network behavior with minimal overhead through virtualization for low-latency measurements and performance benchmarking. Experiments on different network topologies demonstrate the potential benefits of HVNet. The goals of our work are threefold: we (i) demonstrate the impact of HVNet and its virtualization features on the actual measurements, (ii) investigate the repeatability of HVNet results, and (iii) compare HVNet and Mininet measurements with a focus on latency measurements.

The rest of the paper is structured as follows: Section II presents background information and related work. Details of the proposed HVNet framework are shown in Section III. HVNet and its limitations are analyzed and evaluated in Section IV. Information on accessing measurement data is presented in Section V with a conclusion in Section VI.

II. Background and Related Work

This section provides background information as well as related work on network topology emulation, simulation, and virtualization for stable and reliable latency analysis.

A. Network Simulation and Emulation

Network simulation and emulation software offer flexibility, scalability, and re-usability, thereby facilitating a wide range of experiments. However, none of the currently popular frameworks for simulation and emulation, e.g., OMNet++, or Mininet [2]–[5], are designed for realism and precision concerning packet processing times of the investigated network elements.

OMNet++ is a discrete event simulation framework [2] that allows, among other things, to simulate networks and derive metrics from the simulation, such as a prediction of latency or flow behavior. It does not support the execution of Linux internal network functions directly in real-time on the different nodes but allows the integration of external applications into the simulation using discrete events to simulate time and steps [6]. This approach allows the computation and execution of simulations with complex calculations without optimizing the system's performance.

Mininet is a network emulator suitable for addressing various performance issues. It uses Linux on-board features and can use standard network applications that send real traffic over emulated networks [1]. It uses container-based emulation with process spaces and network namespaces to improve network measurements [1], [7], [8]. Another attempt to improve network emulation is Containernet [9], a Mininet extension for experiments that include Mininet and Docker containers.

B. Network Virtualization

To create results that are highly similar to bare-metal networks, the impact of virtualization on measurement results needs to be avoided or reduced as much as possible. Several measures are available for isolating different virtual machines (VMs) to create a more predictable and stable behavior, such as pinning VMs to specific cores or partitioning the

978-1-6654-0926-1/22/$31.00 ©2022 IEEE

shared CPU cache across different VMs. Several tuning guides and publications are available that document and evaluate such optimizations and provide a set of measures to isolate the different VMs, to improve network and system latency properties [10]–[13].

We aim to create realistic network experiments with multiple emulated nodes. Therefore, we need to share network resources across VMs as the number of NICs and PCIe-slots is typically lower than the number of possible VMs. Traditional techniques such as a Linux bridge, or transition techniques such as network address translation, require the hypervisor to handle packet processing and the distribution of packets to VMs, thereby affecting performance [14]. Other options to interconnect VMs include emulated connections such as virtual Ethernet pairs in Mininet [1], which requires the host OS kernel to handle packet processing, affecting performance.

Huang et al. [15] suggest using features of widely-deployed NICs and CPUs such as single-root IO-Virtualization (SR-IOV) to scale the use of NICs to various VMs without using the host OS or hypervisor to handle packet processing tasks. As Dong et al. [16] describe, SR-IOV in combination with an I/O Memory Management Unit (IOMMU) is able to share an IO device between multiple VMs using hardware features. Hardware and driver are used to split an IO-device into multiple virtual functions (VFs) and a physical function (PF), directly available as individual PCIe devices. The IOMMU manages the address space for direct memory access between system and main memory. The Memory Management Unit (MMU) is still used to map physical addresses to virtual addresses. Not all parts are shared, such that, e.g., device-wide settings, are handled inside the PF. Using SR-IOV, networking in the VMs can be handled independently of the VM host system [16].

Several works analyzed the performance of SR-IOV in comparison to native and sharing techniques without SR-IOV, such as [15], [17], [18]. SR-IOV was investigated in the context of Ethernet and InfiniBand [18]–[20]. Latency analyses by Bauer et al. [21] and Lockwood et al. [18] show that SR-IOV introduces latency overhead between native and conventional virtual setups. Using SR-IOV in combination with low-latency optimization techniques significantly improves the network latency, as Gallenmüller et al. [22] have shown focusing on worst-case analysis of the latencies towards the 5G ultra-reliable and low-latency communication (URLLC) profile. They measured a 1 µs increase for tail latencies in an SR-IOV-driven setup.

Most analyses focus on a single VM rather than the impact on others, as in [12], [12], [15], [17]. This work aims to use a combination of network virtualization, multiple topologies, and low-latency optimizations to measure flexible topologies.

III. Hardware-Assisted Virtual Networking

This section explains details of HVNet, our hardware-assisted virtual networking framework for latency measurements with high precision. It uses VMs and SR-IOV on a single physical host and provides stable and reliable end-to-end latency measurements, supported by hardware acceleration and software optimization settings.

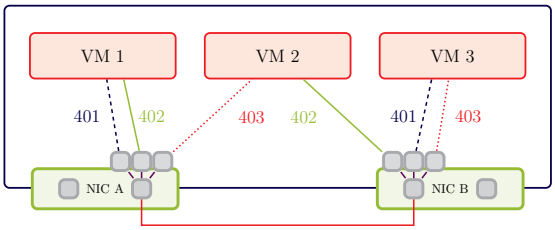


Fig. 1: Example HVNet setup including VLAN-IDs

A. Hardware Requirements and Architecture

Two NICs are interconnected using a wired connection. With SR-IOV, the physical link is logically separated into multiple virtual links [16]. As HVNet supports networks on Layer 2 and higher, we use virtual LAN (VLAN)-IDs to determine the target of a packet, handled by the respective VF. We use NICs, such as the Intel X700 series, that support unicast full promiscuous mode for the VFs. Figure 1 shows an example setup of HVNet using VMs 1–3 and two physically wired NICs A and B. VM 1 and VM 3 are connected to each other via VLAN-ID 401, with VM 1 attached to a VF on NIC A and VM 3 to a VF on NIC B. With the respective VFs being distributed to different NICs, the packets have to use the physical link. This distribution further ensures that the packets are not forwarded directly on the NICs without serialization, thereby skipping processing steps that happen on non-virtualized or emulated networks. The links between the other VMs are distributed in a similar way.

HVNet uses an IOMMU and an SR-IOV-capable CPU to enable direct passthrough from VFs to VM. We choose Kernel Virtual Machine (KVM) as hypervisor and Debian Buster as OS on both VMs and VM host. The overall system is managed using the testbed controller pos [23], which allows repeatable and reproducible measurements, to integrate the optimizations outlined in the following section.

B. Latency Optimization Techniques

We want to measure the behavior of HVNet in a challenging environment. Therefore, we opted for a measurement investigating tail-latency behavior. In such an environment, tail-latency differences between a bare-metal setup and a naively virtualized setup can reach up to several hundred microseconds [24].

To create measurements that behave more similar to the bare-metal setup, we extend the optimizations described in previous work [22]. Our novel setup involves several VMs and replaces DPDK-based with Kernel-based networking in comparison to Gallenmüller et al. [22]. On the VMs and VM-host, different optimization techniques are used. Using the Linux network stack requires the Kernel to run on all VM-cores.

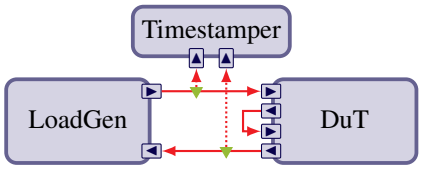


Fig. 2: Measurement setup

1) Virtual Machine Host: To reduce wake-up times for idling cores, energy-saving mechanisms are disabled. Using *nosmt*, simultaneous-multi-threading is disabled, preventing unwanted CPU resource sharing between different VMs. Furthermore, *nohz_full* and *isolcpus* are used, avoiding rescheduling and timer interrupts, thereby redirecting tasks and read-copy-update (RCU) callbacks to Core 0 used for the VM-host Linux Kernel and hypervisor processing. This feature avoids interrupt processing of the host OS on the CPU cores dedicated to VMs. Using these techniques, the expensive VM exits are reduced as much as possible as described in [22].

2) Virtual Machine: To use the in-built forwarding mechanism of the Linux Kernel, interrupts are necessary on the VMs such that we use a Debian Buster with *rt_preempt_full* for real-time communication. Additionally, power saving mechanisms are disabled where possible to avoid latency caused by waking up CPU cores.

IV. Evaluation

To demonstrate the potential of HVNet, we performed measurements using HVNet on several randomly generated topologies. We evaluated HVNet in comparison to Mininet on one example network.

A. Setup

To compare the measurements precisely and accurately, we use a measurement setup relying on hardware timestamping. The Device under Test (DuT) running HVNet is measured using the load generator (LoadGen), and the timestamping machine (Timestamper), both running the packet generator MoonGen [25]. Figure 2 presents the measurement setup.

Each VM is attached to the ingress and the egress port of the LoadGen via one VF, respectively. This configuration is required to allow individual source and destination hosts in the topology on a per-flow basis. Using MAC addresses, the source node is identified. The setup is performed using a configuration file, describing topology and flows.

The LoadGen is the sending/receiving node for traffic of all flows. The sending order of packets between flows is random to reduce the influence of packet order in the experiment. In addition, the Timestamper is connected to the outgoing/incoming physical link of the LoadGen with passive optical terminal access points (TAPs) and monitors the traffic while timestamping each packet with a resolution of 12.5 ns [26]. The timestamped packets measure the latency by calculating the difference between sending and receiving timestamp of the same packet equipped with a unique identifier ID in combination with the destination port and source IP address of the packet.

The DuT is equipped with an AMD EPYC 7551P 32-Core Processor with 128 GB RAM as well as 2 × Intel X710 10GbE SFP+ NICs, with a loop connected between both cards and the other ports connected to the LoadGen (cf. Figure 2). 15 VMs and 64 links can be used, as a result of the available memory and NIC hardware resources [27]. The LoadGen is equipped with an Intel Xeon Silver 4116 CPU, 192 GB RAM and a dual-port Intel Corporation 82599ES 10-Gigabit SFP+ NIC. The Timestamper features an Intel Xeon CPU D-1537, 32 GB RAM and a dual-port Intel X552 10 GbE SFP+ NICs capable of timestamping all received packets.

All topology measurements were performed with a packet size of 363 B without Ethernet framing for 3 min. Each measurement was repeated three times. UDP traffic was used to avoid the impact of congestion and flow control on measurement results. Using the default VF settings, the maximum throughput is the NIC's line rate divided by the number of generated links on the NIC. The maximum number of links corresponds to the maximum supported VFs on the X710 NIC. Artificially adding loss or latency to experiments is not considered in this work.

The routing policy database management tool `ip-rule` is used to enable multipath routing based on the UDP destination port used to identify certain flows, matching them to the correct routing table. To forward the packet correctly, we define a routing table for each interface on the node using the default gateway towards the interface of the connected VM.

B. HVNet in Isolation

We analyzed a total of 100 networks with three measurements each. To evaluate the results derived using HVNet, a randomly-selected example topology (`nw`) is used as shown in Figure 3. Figure 4 shows the 5000 worst-case latency events over time for all three repetitions.

All of these worst-case latency events are in a range between 0.22 ms and 0.41 ms. For performance reasons, we have generated a random packet-order table with a limited number of entries for each repetition. The worst-case events follow a stable distribution over time, showing repeated patterns in the interval needed to traverse the packet-order table once, which happens every 2.94 s with the table size used in `nw`. As shown, the behavior of the measurements does not differ significantly, such that a single measurement run is used for the detailed flow-level analysis.

In `nw`, all flows have per-flow fixed rates between 1 Mbit/s and 294 Mbit/s. Furthermore, they all have flow lengths between two and five hops. To analyze the worst-case events in detail, high dynamic range (HDR)-histograms are used, highlighting tail latencies [28].

In Figure 5a, the 2-hop flows with a sending rate of 1 Mbit/s are shown. These flows behave similarly at worst-case latencies. The flows traversing the same logical link, such as `f1` and `f21`, only differ slightly. The other flows are sent

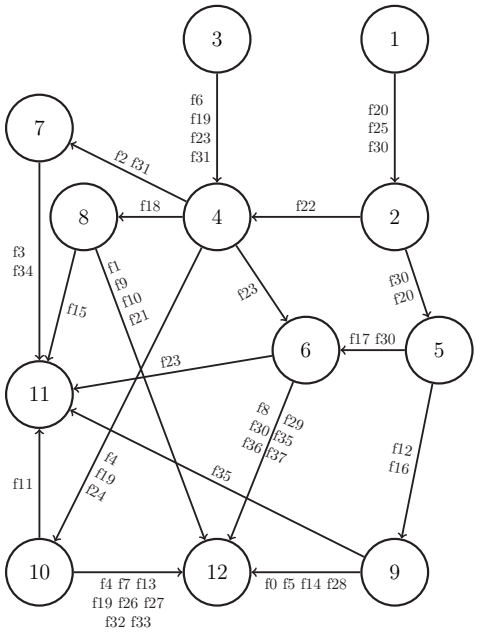


Fig. 3: Network topology of `nw` including flow routing

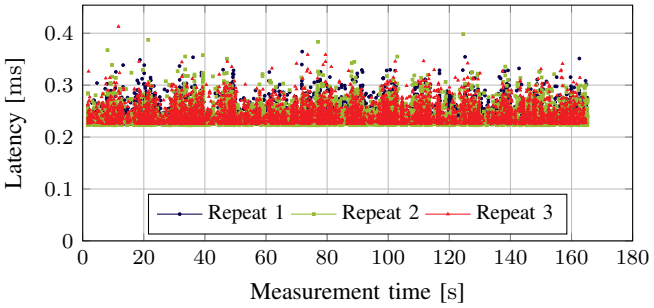
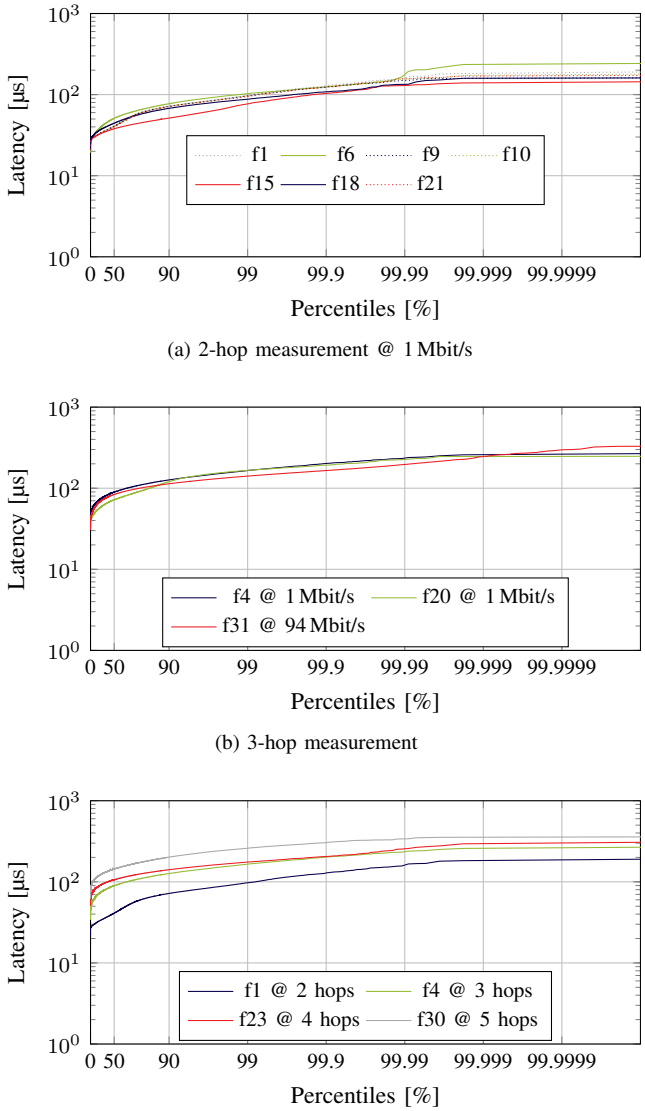


Fig. 4: 5000 worst-case events over time for `nw` using HVNet



(a) 2-hop measurement @ 1 Mbit/s

(b) 3-hop measurement

(c) 2 to 5-hop measurement @ 1 Mbit/s

Fig. 5: HDR-histogram of flows based on hops of `nw`

over different logical links, with `f15` and `f18` using similarly loaded links.

The results for 3-hop flows are shown in Figure 5b. A difference to note is that two of the flows are sent with an identical rate, while the third flow has a significantly higher rate. Flow `f31` has a rate of 94 Mbit/s compared to 1 Mbit/s each for the other two flows. The higher rate results in an increase of the latency in the higher percentiles. Except for this slightly higher worst-case latency, the behavior is in a similar range as for the other flows. The shown results represent consistent behavior based on topology and flow parameters.

In total, four different flow lengths are used in `nw`. We selected one for each flow length based on a rate of 1 Mbit/s. As Figure 5c shows, the tail latency is increasing for flows with a higher flow length. The latencies for `f23` and `f4` diverge from each other only in the higher percentile ranges. Flow `f23` traverses connections with few other flows, whereas `f4` traverses connections used by most other flows in `nw`.

Based on this data and the evaluations for the selected flows, we conclude that the measurements show the expected behavior. Specifically, influences of flow lengths, flow rates, link utilizations, and number of flows on a link have the expected impact.

C. HVNet vs. Mininet

To evaluate the improvement of HVNet, we compare it to an existing network emulation tool, Mininet [1]. To keep the measurements comparable, the same measurement setup was used with Mininet instead of HVNet running on the DuT. The evaluation uses the same example network, `nw`.

We chose to compare our results with Mininet, a network emulation framework, omitting OMNet++ and Containernet. OMNet++ does not allow the use of the same scripts for setting the network behavior on the nodes, and it is not possible to integrate it directly into the presented measurement setup to compare the results directly. For these reasons, we chose to

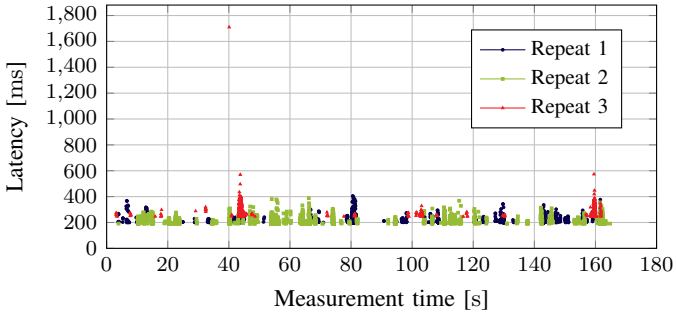


Fig. 6: 5000 worst-case events over time for `nw` using Mininet

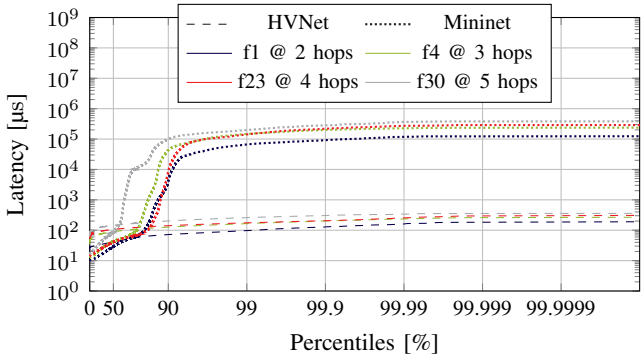


Fig. 7: HDR-histogram of flows based on hops of `nw`

compare it to Mininet as a network emulation tool rather than a discrete event emulation like OMNet++. In addition, the scope of the simulations is different since it is not directly possible to perform precise measurements of actual network behavior with the original software without modifying the software itself. Moreover, we decided to focus on Mininet beacause of its status as a commonly used tool. Comparison to Containernet are left for future work.

Figure 6 shows the 5000 worst-case events for `nw` with Mininet. For the Mininet measurements, most worst-case latencies are between 190 ms and 600 ms, rare outliers reach 1800 ms, showing a heterogeneous worst-case behavior.

Furthermore, we used selected flows to show the different delay behavior in comparison between Mininet and HVNet in Figure 7. Up to the 50^{th} percentile, the latency measured in the Mininet scenario is lower than HVNet, which was expected due to the lightweight virtualization used. Up to the 90^{th} percentile, the Mininet delays increase significantly compared to the measurements for HVNet, showing that the Mininet measurements are unstable regarding tail latencies. These results show significant precision improvements of HVNet.

Table I compares the maximum delay and the jitter between Mininet and HVNet. We analyzed the latency jitter for packets of the same flow. The tail latency improves by up to 1063 x; the tail jitter decreases by 1732 x. The improvement in terms of jitter and delay is increasing with an increasing flow length. As a result of the comparison, the quality of the results of HVNet show major improvements in stability and absolute values.

	Delay [µs]			Jitter [µs]		
Hops	Mininet	HVNet	Improv.	Mininet	HVNet	Improv.
2	125 536	203	618 x	124 154	164	757 x
3	241 471	282	856 x	232 714	207	1124 x
4	290 610	330	881 x	274 090	211	1299 x
5	388 123	365	1063 x	388 068	224	1732 x

TABLE I: Comparison of worst-case delay and jitter of Mininet and HVNet

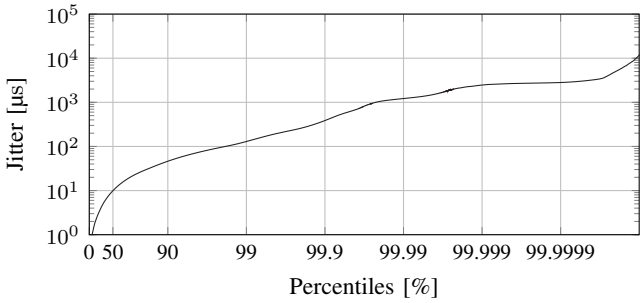


Fig. 8: HDR-histogram over all jitter values of all 100 topologies

D. Further Topologies

In addition to the example topology of `nw`, we analyzed other randomly generated networks. To further analyze the topologies, we show in Figure 8 the HDR-histogram of jitter of all topologies, showing jitter below 1 ms roughly up to the 99.9 percentile. We looked at the overall jitter in the measurements to ensure that the presented improvements for `nw` can be further shown in multiple random topologies analyzed in HVNet. This result shows that HVNet is capable of flexible measurements and experiments using a hardware-based virtual network emulation. We investigated topologies with up to 15 hosts and a maximum of 14 hops per flow.

E. Limitations

HVNet and the used measurement setup have certain limitations.

1) HVNet: Compared to Mininet, the per-node resource requirements of HVNet are higher. Topology sizes are limited based on the available hardware resources. The minimum requirement is 1 core and 8 GB memory per VM and an additional core for the Linux Kernel and hypervisor. The number of connections is limited to the maximum VFs supported by the NICs. Measurements rarely show a high-latency spike for all flows at exactly the same time. These spikes can be correlated to a Translation Lookaside Buffer (TLB) shootdown happening rarely causing VM exits on all cores simultaneously [29].

2) Measurement Setup: Since only one source of flows is used, it is impossible to analyze flows that send packets simultaneously on different hosts using the presented setup.

The measurement setup incurs additional overhead outside of the DuT, since it has an additional connection to and from the host for the start and end host requiring initial packet processing not included in the topology.

V. Measurement Data

We provide the latency data, configuration files, detailed figures, and scripts to recreate the diagrams to allow for further analysis. The data can be found at [30].

VI. Conclusion

Low-level network topology measurements with fewer hardware nodes and flexibility are important to scale measurements and test capabilities. In this paper, we have demonstrated an architecture consisting of a single host featuring two interconnected NICs. This setup allows analyzing different topologies on a single device without the need for rewiring, while leveraging hardware capabilities and sending packets over a real physical link. Compared to Mininet [1], a network emulator, our solution, HVNet, demonstrated stable, reliable latencies closely resembling the behavior of a real network.

Several randomly generated topologies were analyzed and evaluated. HVNet represents a robust solution for analyzing the latency of network topologies in a flexible manner that provides stable, reliable, and low latencies on VMs. Future enhancements are possible with the additional capabilities of modern 100 Gbit/s NICs, such as improved hardware timestamps of the NIC, or with extending HVNet towards supporting multiple physical hosts as underlying basis.

Acknowledgment

This work was funded by the European Union's Horizon 2020 programme (SLICES-SC, grant 101008468), the Bavarian Ministry of Economic Affairs, Regional Development and Energy as part of 6G Future Lab Bavaria, and the German Federal Ministry of Education and Research (BMBF) as part of 6G-life (grant 16KISK001K).

References

[1] B. Lantz, N. Handigol, B. Heller, and V. Jeyakumar. (2021) Introduction to Mininet. Last accessed: 2022-01-15. [Online]. Available: https://github.com/mininet/mininet/wiki/Introduction-to-mininet

[2] A. Varga, "OMNeT++," in *Modeling and tools for network simulation*. Springer, 2010, pp. 35–59.

[3] M. A. K. Saluja and M. S. A. Darg, "A Detailed Analogy of Network Simulators NS1, NS2, NS3 and NS4," *International Journal on Future Revolution in Computer Science & Communication Engineering*, vol. 3, no. 12, pp. 291–295, 2017.

[4] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International Conference on Communication, Computing & Systems (ICCCS)*, 2014, pp. 139–42.

[5] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu *et al.*, "Advances in network simulation," *Computer*, vol. 33, no. 5, pp. 59–67, 2000.

[6] C. P. Mayer and T. Gamer, "Integrating real world applications into omnet++," *Institute of Telematics, University of Karlsruhe, Karlsruhe, Germany, Tech. Rep. TM-2008-2*, 2008.

[7] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Mininet performance fidelity benchmarks," *Tech. Rep.*, 2012.

[8] ——, "Reproducible Network Experiments Using Container-Based Emulation," in *International Conference on Emerging Networking Experiments and Technologies*. New York, NY, USA: ACM, 2012, p. 253264.

[9] M. Peuster, H. Karl, and S. van Rossem, "MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016, pp. 148–153.

[10] J. Mario and J. Eder, "Low Latency Performance Tuning for Red Hat Enterprise Linux 7," Nov. 2017, last accessed: 2022-01-15. [Online]. Available: https://access.redhat.com/sites/default/files/attachments/201501-perf-brief-low-latency-tuning-rhel7-v2.1.pdf

[11] AMD, "Performance Tuning Guidelines for Low Latency Response on AMD EPYC-Based Servers Application Note," Jun. 2018, last accessed: 2022-01-15. [Online]. Available: http://developer.amd.com/wp-content/resources/56263-Performance-Tuning-Guidelines-PUB.pdf

[12] S. Gallenmüller, J. Naab, I. Adam, and G. Carle, "5G URLLC: A Case Study on Low-Latency Intrusion Prevention," *IEEE Communications Magazine*, vol. 58, no. 10, pp. 35–41, Oct. 2020.

[13] B. Zuo, K. Chen, A. Liang, H. Guan, J. Zhang, R. Ma, and H. Yang, "Performance tuning towards a KVM-based low latency virtualization system," in *2nd International Conference on Information Engineering and Computer Science*. IEEE, 2010, pp. 1–4.

[14] L. B. Yu, "A Research on KVM I/O Performance," in *Applied Scientific Research and Engineering Developments for Industry*, ser. Applied Mechanics and Materials, vol. 385. Trans Tech Publications Ltd, 11 2013, pp. 1790–1796.

[15] Z. Huang, R. Ma, J. Li, Z. Chang, and H. Guan, "Adaptive and Scalable Optimizations for High Performance SR-IOV," in *2012 IEEE International Conference on Cluster Computing*, 2012, pp. 459–467.

[16] Y. Dong, X. Yang, J. Li, G. Liao, K. Tian, and H. Guan, "High Performance Network Virtualization with SR-IOV," *Journal of Parallel and Distributed Computing*, vol. 72, no. 11, pp. 1471–1480, Nov. 2012.

[17] J. Liu, "Evaluating standard-based self-virtualizing devices: A performance study on 10 GbE NICs with SR-IOV support," in *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010, pp. 1–12.

[18] G. K. Lockwood, M. Tatineni, and R. Wagner, "SR-IOV: Performance Benefits for Virtualized Interconnects," ser. XSEDE '14. New York, NY, USA: ACM, 2014.

[19] J. Zhang, X. Lu, and D. K. Panda, "Performance Characterization of Hypervisor-and Container-Based Virtualization for HPC on SR-IOV Enabled InfiniBand Clusters," in *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2016, pp. 1777–1784.

[20] J. Jose, M. Li, X. Lu, K. C. Kandalla, M. D. Arnold, and D. K. Panda, "SR-IOV Support for Virtualization on InfiniBand Clusters: Early Experience," in *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, 2013, pp. 385–392.

[21] S. Bauer, D. Raumer, P. Emmerich, and G. Carle, "Intra-node Resource Isolation for SFC with SR-IOV," in *IEEE 7th International Conference on Cloud Networking (CloudNet'18)*, Tokyo, Japan, Oct. 2018.

[22] S. Gallenmüller, F. Wiedner, J. Naab, and G. Carle, "Ducked Tails: Trimming the Tail Latency of(f) Packet Processing Systems," in *3rd International Workshop on High-Precision, Predictable, and Low-Latency Networking (HiPNet 2021)*, Izmir, Turkey, Oct. 2021.

[23] S. Gallenmüller, D. Scholz, H. Stubbe, and G. Carle, "The pos Framework: A Methodology and Toolchain for Reproducible Network Experiments," in *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21)*, Munich, Germany (Virtual Event), Dec. 2021.

[24] P. Emmerich, D. Raumer, S. Gallenmüller, F. Wohlfart, and G. Carle, "Throughput and Latency of Virtual Switching with Open vSwitch: A Quantitative Analysis," Jul. 2017.

[25] P. Emmerich, S. Gallenmüller, D. Raumer, F. Wohlfart, and G. Carle, "MoonGen: A Scriptable High-Speed Packet Generator," in *Internet Measurement Conference (IMC)*, Tokyo, Japan, Oct. 2015.

[26] *Intel Ethernet Controller X550 - Datasheet*, Intel, 11 2018, rev. 2.3.

[27] *Intel Ethernet Controller X710/XXV710/XL710*, Intel, 8 2019, rev. 3.65.

[28] G. Tene, "HdrHistogram: A High Dynamic Range Histogram," last accessed: 2022-01-15. [Online]. Available: http://hdrhistogram.org/

[29] W. Li, "Torwards a more Scalable KVM Hypervisor," in *KVM Forum*, 2018, last accessed: 2022-01-15. [Online]. Available: https://events19.linuxfoundation.org/wp-content/uploads/2017/12/Update_Wanpeng-LI_Torwards-a-more-Scalable-KVM-Hypervisor.pdf

[30] F. Wiedner, M. Helm, S. Gallenmüller, and G. Carle, "HVNet: Hardware-Assisted Virtual Networking on a Single Physical Host." [Online]. Available: https://mediatum.ub.tum.de/1638129