Performance of Container Networking Technologies

Yang Zhao
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China 210023
yangzhao@smail.nju.edu.cn

Nai Xia
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China 210023
xianai@nju.edu.cn

Chen Tian
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China 210023
tianchen@nju.edu.cn

Bo Li
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China 210023
btlcmr0702@hotmail.com

Yizhou Tang
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing, China 210023
tangyeechou@gmail.com

Yi Wang
Future Network Theory Lab
Huawei
Hongkong, China 999077
wangyi18@huawei.com

Gong Zhang
Future Network Theory Lab
Huawei
Hongkong, China 999077
nicholas.zhang@huawei.com

Rui Li
College of Computer Science and Network Security
Dongguan University of Technology
Dongguan, China 523808
ruili@dgut.edu.cn

Alex X. Liu
Department of Computer Science and Engineering
Michigan State University
East Lansing, USA 48824
alexliu@cse.msu.edu

ABSTRACT

Container networking is now an important part of cloud virtualization architectures. It provides network access for containers by connecting both virtual and physical network interfaces. The performance of container networking has multiple dependencies, and each factor may significantly affect the performance. In this paper, we perform systematic experiments to study the performance of container networking technologies. For every measurement result, we try our best to qualify influencing factors.

CCS CONCEPTS

• **Networks** → **Network performance analysis**;

KEYWORDS

Container, Networking, Measurement

ACM Reference format:
Yang Zhao, Nai Xia, Chen Tian, Bo Li, Yizhou Tang, Yi Wang, Gong Zhang, Rui Li, and Alex X. Liu. 2017. Performance of Container Networking Technologies. In *Proceedings of HotConNet'17, Los Angeles, CA, USA, August 25, 2017,* 6 pages.
http://dx.doi.org/10.1145/3094405.3094406

HotConNet'17, August 25, 2017, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5058-7/17/08…$15.00
http://dx.doi.org/10.1145/3094405.3094406

1 INTRODUCTION

Container solutions, such as Docker [6], are now an integral part of cloud computing. Compared with virtual machine (VM) and hypervisor, container technology provides a lightweight alternative for virtualization. Container networking provides network access for containers by connecting both virtual and physical network interfaces. Besides providing advanced management and virtualization features, the networking solutions should provide good performance such as high throughput and low latency.

The performance of container networking has multiple dependencies, and each factor may significantly affect the performance. There are different NIC options, *e.g.*, veth and MACVLAN. There are different bridge options, *e.g.*, Linux bridge and OVS. There exists various communication topologies, *e.g.*, inside the same host, inside the same VM, inside the same host but one resides in a VM, and in different hosts *etc.* There are various communication patterns, *e.g.*, TCP v.s. UDP, single flow v.s. multiple flows *etc.* Further, the container networking in Windows is quite different from that of in Linux.

We perform the first systematic measurement to study the performance of different container networking technologies. We try our best to qualify influencing factors and explain the cause of overhead. There are still some measured results that we cannot fully understand or have enough confidence of the causes. To avoid misleading the readers, we use *italic* sentences for speculative explanations.

In Section 2, we present related work and evaluated container networking technologies. In Section 3, we first gives the experiment setup, then present our measurement results. Section 4 summarizes the paper.

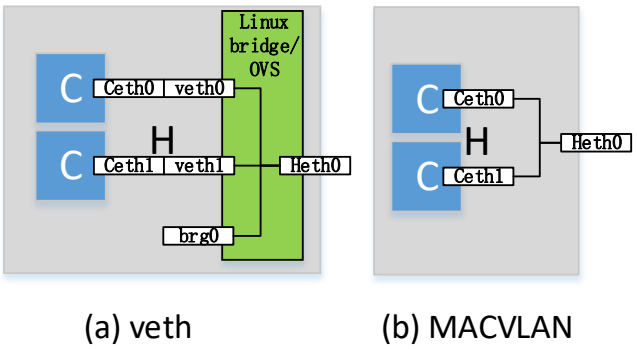

Figure 1: veth and MACVALN (bridge mode)

2 BACKGROUND

Related Work: Morabito *et al.* compare various system performance metrics between hypervisors and containers [7]. Regards networking, they use *Netperf* [5] to evaluate both unidirectional and request/response data transfer with TCP and UDP protocols. It demonstrates that Docker [6] can achieve almost equal TCP throughput compared to the native service without virtualization. However, the UDP throughput of Docker is 42.97% lower than that of the native service. For request/response transactions, Docker is 19.36%/12.13% lower than the native service with TCP/UDP respectively. In this paper, we demonstrate that the performance gaps among different container networking options are significant. More importantly, we try to explain the cause of the performance difference. We also confirm that UDP is inferior in performance compared with TCP.

Felter *et al.* compare MySQL [9] performance on Docker with two different networking settings: *-net=host* and *NAT* [2]. Apparently, NAT introduces additional overhead. In this paper, we show that MACVLAN has very little overhead, while veth mode adds significant overhead. For Network Function Virtualization [3], Bonafiglia *et al.* present measurement of chains of Docker containers [1]. With Linux bridge connecting containers, the throughput can be as low as 3 Mbps with 8 chained containers. The reason is that a single kernel thread executes all the operations associated with a packet without parallelization at all. In this paper, we also demonstrate that network throughput can be throttled by CPU capacity.

Evaluated Technologies: We measure two NIC modes: veth (virtual ethernet) and MACVLAN (mac-address based virtual lan tagging), as illustrated in Figure 1. The veth mode creates a pair of devices for each container, one attached to the container and the other attached to the host. Network communication in this mode is achieved by copying MAC frames between the paired devices. We can manage the veth devices like any other devices on the host. The container can communicate with host and other containers (on the same host or not) through a Linux bridge or an OVS bridge.

MACVLAN mode allows the host to configure sub-interfaces, each with its own unique (randomly generated) MAC address, of a parent physical Ethernet interface. A VM or container can then bind to a specific sub-interface to connect directly to the physical network, using its own MAC and IP addresses. MACVLAN has four modes: priviate, VEPA, bridge, and passtru mode. Among them, two modes (VEPA and bridge) allow containers on the same host to communicate with each other. However, the VEPA mode needs

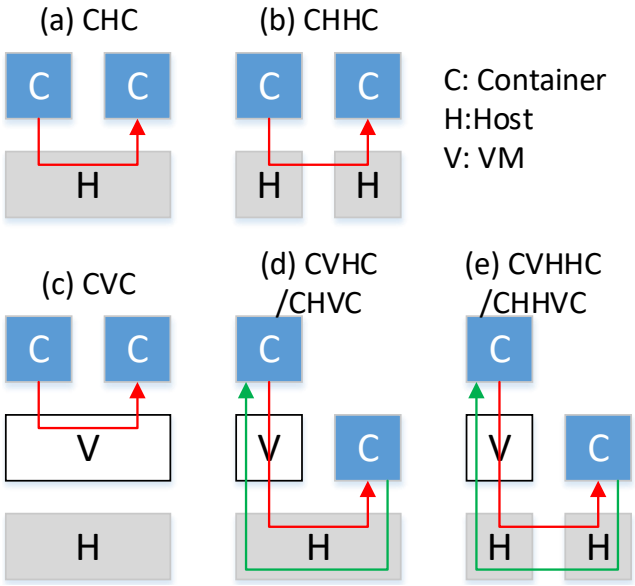

Figure 2: Topologies in our performance comparison.

feature support from switches. In this project, we choose bridge mode of MACVLAN to do the evaluation.

The bridge mode in MACVLAN is much simpler than standard Linux bridge or OVS and it does not have the overhead of device pairs. Generally speaking, MACVLAN performance is better than veth and is a good option for providing egress connection to the physical network. While, with the help of OVS (which is feature rich and has wide deployment), veth device pairs can be easily embedded in complex virtual network topologies. So most virtualization environments (e.g. docker, KVM, XEN) take veth network mode and OVS as their default choice.

3 EVALUATION

3.1 Setup

Testbed: Our testbed consists of 4 servers connected to a 32-port 100 Gbps Mellanox SN2700 switch. Each server is a DELL PowerEdge R730, equipped with two Intel Xeon E2650 CPU each with 8 cores (*i.e.*, totally 16 cores), 128 GB RAM, a 2 TB disk, and a Mellanox CX4 dual-port 100 Gbps Ethernet NIC. Two servers run Ubuntu Server 14.04 with Linux 3.19.0 kernel, and the other two run Windows Server 2016 Standard Edition.

Container and VM Setup: In Linux, networking mechanisms for both LXC and Docker containers are supported by the underlying kernel network stack and isolation mechanisms (*i.e.*, namespaces and cgroups). Docker official networking user space utilities support veth mode with Linux bridge (can be easily extended to OVS) and MACVLAN [4]. As a result, we choose LXC as our test container for simplicity and believe that our results also apply to Docker containers. The rootfs running in LXC is also Ubuntu Server 14.04. We use KVM as the virtual machine hypervisor. Each VM is installed with the same version of OS and kernel network setup as the host server. In Windows, we use Docker (version 17.03.0-ee-1) as our test container running Microsoft NanoServer.

Topology Setup: In our experiment, we have 5 topologies. As shown in Figure 2, we name each topology after its transmission

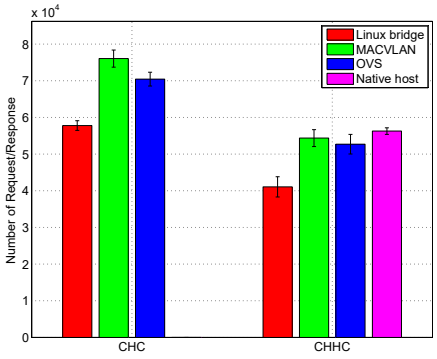

Figure 3: Latency dominates request/response speed.

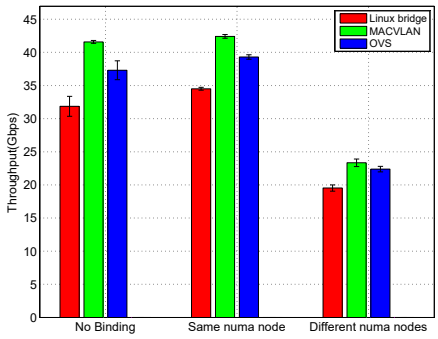

Figure 4: CHC single flow throughput with core-binding settings.

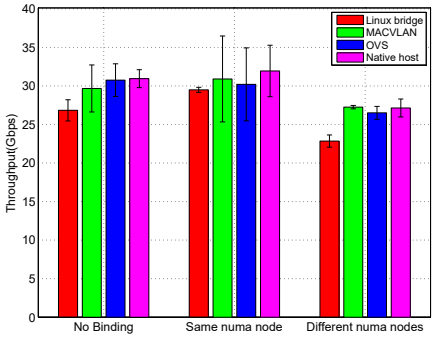

Figure 5: CHHC single flow throughput with core-binding settings.

direction. Topology a and b are bare metal installation without VM. Topology c, d and e are used to evaluate the performance of containers inside VMs. Topology a is *CHC*, where two containers in the same host communicate with each other. Topology b is *CHHC*, where two containers are installed in two hosts separately. Topology c is *CVC*, where two containers are installed in the same VM of a host. Note that all above mentioned topologies are symmetric. Topology d is *CVHC/CHVC*, where two containers are installed in the same host, but only one is inside a VM. As a result, we need to distinguish two communication directions. Topology e is *CVHHC/CHHVC*, where two containers are installed in two hosts separately, and one of them is inside a VM.

Evaluation Setup: In Linux, we use *netperf* to test the latency for TCP with packet size 64B. We use *iperf3* to test the throughput for TCP and *pktgen* kernel module for UDP. Unless otherwise specified, the throughput results in this paper are measured with packet size 1500B. Iperf3 tool and pktgen module both can control the message size and flow number.

To evaluate the impact of NUMA affinity, we collect the results from 3 cases when an iperf3 receiver is: a) not binding to a specific CPU node; b) binding to the same NUMA node (as the sender when in the same host or as the network card when across hosts); c) binding to a different NUMA node. It worth to note that some of the packet processing work is done at Linux kernel softirq context whose scheduling is much more harder to be tuned manually (*i.e.*, may depend on the different NIC drivers). In order to guarantee that our results are meaningful to most Linux users, we let Linux kernel and NIC driver to do the auto scheduling of interrupts and softirqs. In Windows, we use *NTttcp* tool to test the throughput of TCP. Each data line/bar in following figures is the average result of 10 experiment samples. Each experiment lasts 20 seconds.

3.2 Installation on host

Latency: In order to measure the performance of packet latency, we use netperf with TCP packet size 64B to log the number of request/response interactions per second between two containers. As shown in Figure 3, for networking options, MACVLAN is the best mode, followed by veth over OVS and veth over Linux bridge. As mentioned above, veth mode used by OVS or Linux bridge is composed of two paired devices. More processing is needed when packets pass through this device pair. The OVS mode is better than Linux bridge because OVS optimizes the packet forwarding process with its "fast path" [8]. On average, MACVLAN outperforms OVS/Linux bridge by 7.93%/31.63% on the same host respectively, and by 3.14%/32.39% across hosts respectively. Besides, MACVLAN is only less than bare machine by 3.41%.

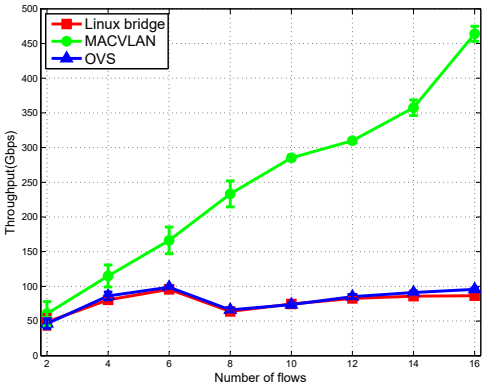

Figure 6: CHC multi flows throughput with networking modes.

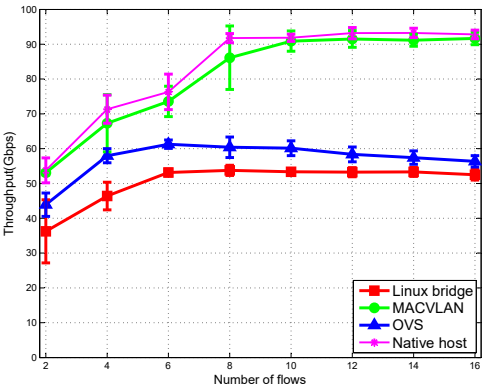

Figure 7: CHHC multi flows throughput with networking modes.

Single-flow Throughput: We use iperf3 tool to measure the single flow TCP throughput of two containers. Demonstrated in Figure 4 and Figure 5, MACVLAN in general achieves the best performance.

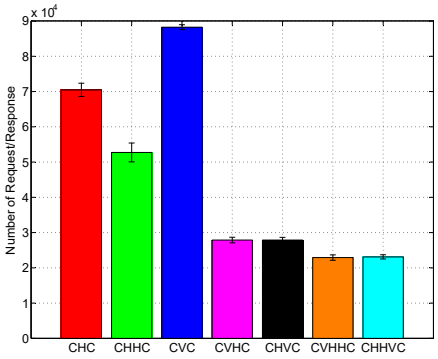

Figure 8: Latency performance across all topologies.

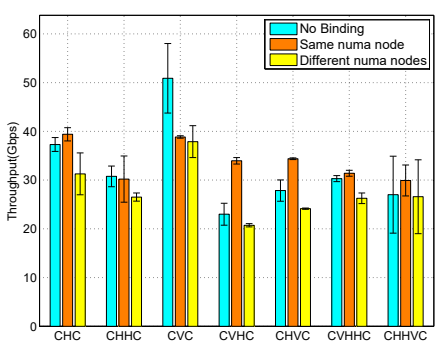

Figure 9: Single flow throughput with topologies.

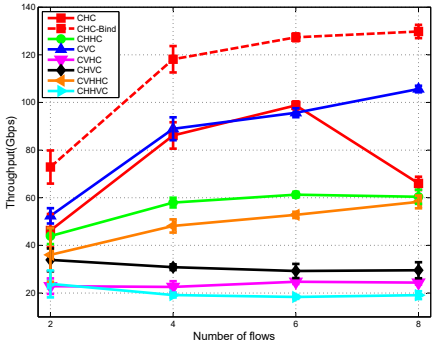

Figure 10: Multi flows throughput with topologies.

While with the CHHC topology, MACVLAN and OVS are close even to bare metal performance without containers (Figure 5).

We also measure the impact to TCP throughput caused by CPU scheduling and NUMA node affinity. Overall, two iperf3 applications in the same NUMA node can get better throughput than that in different NUMA nodes. On the same host, binding to different NUMA nodes drops the throughput by 77.99% on average compared with binding to the same NUMA node. While across different hosts, binding the iperf process to the same NUMA node with the network card can also bring a better performance. Without binding, the performance is close to manual binding. It is clear that Linux scheduling does a good job of automatical collocating sender and receiver processes. In addition, when binding the cores on the same NUMA node, the jitter is slightly bigger.

Multi-flow Throughput: Figure 6 and Figure 7 shows the throughput of multiple TCP flows between two containers. We set the maximum of flows to the CPU core numbers, *e.g.*, the bare metal has 16 CPU cores and the virtual machine has 8 cores. We do not consider NUMA affinity in this experiment because we notice that for flow number bigger than 2, we always get better result if we let Linux to do auto scheduling.

On the same host, the throughput for MACVLAN is nearly proportional to the flow numbers. The OVS mode outperforms the Linux bridge by 3.43%. We observed that when the flow number is above 6, the throughput for OVS and Linux bridge tend to stabilize at around 100 Gbps, far lower than the memory speed. We blame this to the intrinsic overhead since the underlying device for OVS and Linux bridge mode is veth. Veth needs additional kernel threads compared to MACVLAN. This increases the average per-packet processing time due to lock contention in network stack and also the complexity of CPU/NUMA scheduling. We observed that in veth mode, even with 16 flows, only about 70% of the total CPU resources can be used which we consider as an indication of un-balanced CPU contention. *We leave the kernel source code level analysis for this phenomenon as our future work.*

Across two hosts, MACVLAN outperforms OVS and Linux bridge and is very close to the bare metal throughput. The throughput for MACVLAN get its maximum value when the flow number arrives 10, then remains unchanged since it is already close to NIC capacity. When the flow number is 6, OVS and Linux bridge again achieve the maximum speed. With the increasing number of flows, the throughput for Linux bridge and OVS gradually decline.

3.3 Installation on VM

Note that despite the superior performance of MACVLAN, OVS is more prevalent in real deployment due to its rich features. For the rest of evaluations, we use veth over OVS as the main networking fabric.

Latency: We focus on the network delay of different topologies. As depicted in Figure 8, counter-intuitively, the CVC topology gets the best performance and even outperforms CHC by 25.24%. We observed that with the CVC topology, the VM can always auto-schedule the iperf3 sender and receiver to the same NUMA node (we will discuss this later in the multi-flow part). Therefore, CVC's performance is superior than CHC's. But from results in Figure 10, we can see that if we bind iperf3 sender and receiver in the same NUMA node, the CHC performance is superior than CVC's.

The CVHC/CHVC and CVHHC/CHHVC topologies are much lower in performance. CVHC/CHVC reduce the performance by 60.45% compared with the CHC topology. CVHHC/CHHVC reduce the performance by 56.37% compared with the CHHC topology. It is clear that the virtual machine layer introduces significant overhead for per-packet processing. The forward and reverse directions are almost the same because the interaction manner of netperf is TCP ping-pong manner, and the packet will go through both the forward and the reverse software stacks.

Single-flow Throughput: The single flow bandwidth of different topologies are demonstrated in Figure 9. The CVC has the best performance because of the VM CPU scheduling. The CVHC and CHVC have different throughput because TCP send and receive kernel paths have different overhead (the same are CVHHC v.s. CHHVC). The communications in the same host generally have better performance than those across the hosts. If we do not bind the process to NUMA node, the throughput with CVHC is lower than that with CVHHC. *We think that, due to the design in Linux network stack, with CVHHC more CPU cores can take part in packet processing.* We observed about 40% more CPU consumption (counting both sender and receiver) with CVHHC than that with CVHC.

Multi-flow Throughput: Figure 10 demonstrates the throughput of multiple TCP flows of different topologies. It is clear that inside a VM, the CPU core scheduling scales. As we expected, the

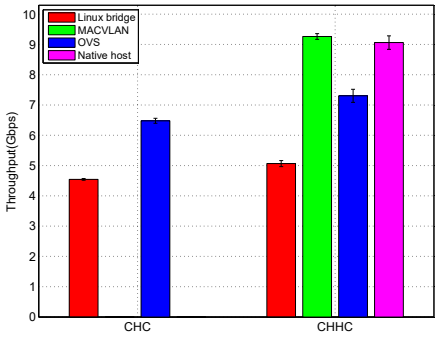

Figure 11: Throughput of single flow (UDP).

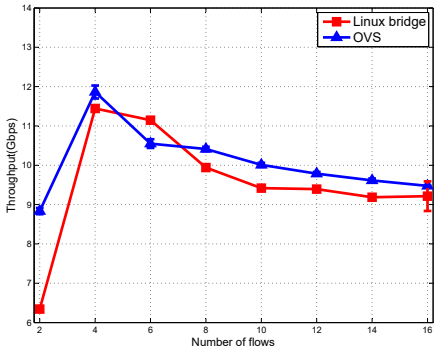

Figure 12: Throughput of multiple flows (UDP) with CHC topology.

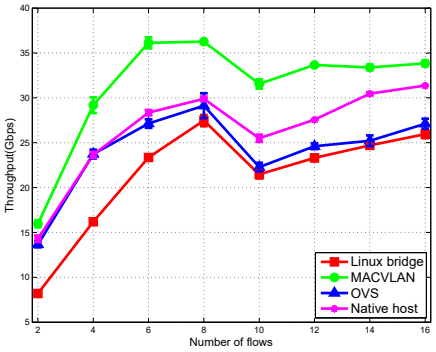

Figure 13: Throughput of multiple flows (UDP) with CHHC topology.

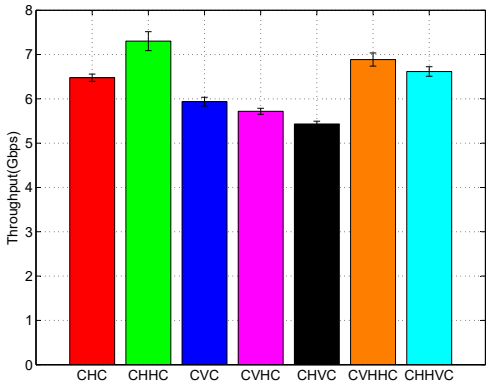

Figure 14: Throughput of single flows with different topologies (UDP).

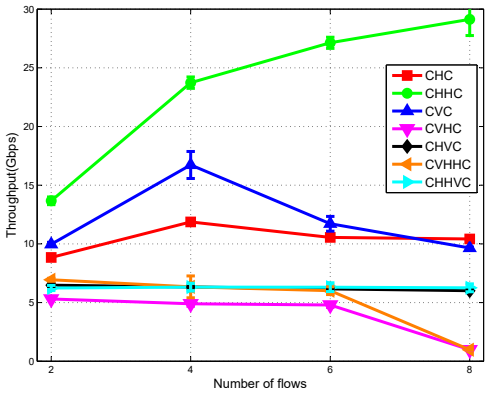

Figure 15: Throughput of multiple flows with different topologies (UDP).

throughput with CVC topology is superior than that with CHC because of the auto NUMA scheduling of KVM. We notice that with CHC topology, when flow number increase to 8, if all processes are free to be scheduled to different NUMA nodes, the throughput deteriorates significantly. However, if we bind all sender receiver pairs to the same NUMA node, the throughput with CHC topology is significantly higher than that with other topologies.

To our surprise, we found that with topolgies of CHVC, CVHC and CHHVC the throughput does not scale with the number of the flows, while with CVHHC topology the performance is much better. We believe there is some kind of performance bottleneck underneath. We left the analysis of this as our future work.

3.4 UDP

We use Linux kernel module pktgen to do the experiments for UDP. We control the send rate to ensure the loss rate is less than 3%. We can not get the result for CHC in MACVLAN mode because pktgen directly feeds the packets to the device driver API so is above the MACVLAN bridge layer. Since pktgen sends packets directly to the discard port, no user space program (on both send/receive sides) is involved during these tests. So the results in this section can be considered as pure Linux network stack performance values. The results are shown in Figure 11, 12, 13, 14 and 15.

There are four major observations. Firstly, all major results from TCP still holds. For example, MACVLAN has the best performance, throughput with CHVC, CVHC and CHHVC does not scale. Secondly, the throughput of UDP is significantly lower than that of TCP. It is expected because Linux network stack have receive offloading (i.e. GRO [10]) support for TCP which can significantly reduces the per-packet processing time. If we disable GRO in Linux kernel, the throughput of TCP drops significantly to the same level as UDP. Thirdly, due to the overall low performance, the impact of VM is negligible, which is shown in Figure 14 and 15,. Fourthly, for multiple flows, the MACVLAN mode significantly outperforms native host with CHHC topology. This is out of our expectation. *We leave the code level analysis for this phenomenon as our future work.*

3.5 Packet size

We perform some experiments to find out how packet size influences the throughput. Figure 16 and 17 demonstrates the throughput performance of single flow of TCP and UDP with different packet size. To our surprise, we find that Mellanox NIC has fixed maximum single flow Packet-Per-Second (PPS) rate for all packet sizes. So for communications across the hosts, their throughput is proportional to the packet size. However, TCP on the same machine's throughput is stable when the packet size is increasing, that's because the GRO for TCP on the same host can merge small packets into large ones with zero memory copy.

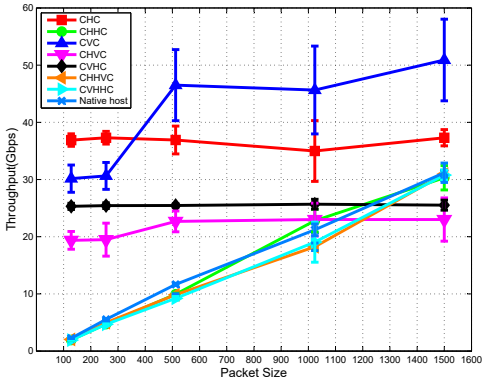

Figure 16: TCP single flow throughput versus packet sizes.

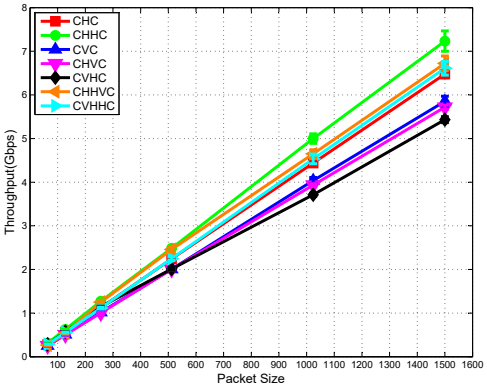

Figure 17: UDP single flow throughput versus packet sizes.

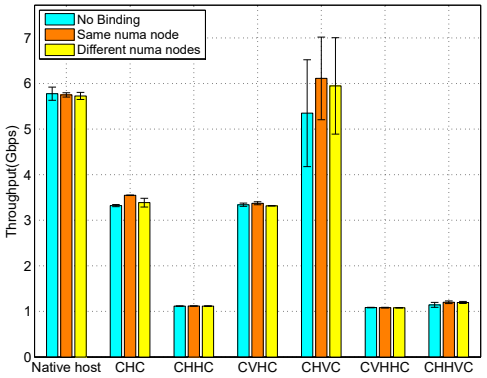

Figure 18: TCP single flow throughput (Windows).

3.6 Windows

We measure some topology on the same host to verify Windows's container networking performance. As depicted in Figure 18, the highest throughput of CHVC is only 6.1 Gbps. The jitter of the CHVC is huge. The communication in the same NUMA node is slightly better than that across different NUMA nodes. Windows's throughput is much lower than that of Linux. *It is hard, if not completely impossible, to tune Windows parameters as it is not open-sourced. It is hard to explain why the communication with CHVC topology significantly outperforms that with other topologies without digging deep into the windows network stack implementation. So we left it as our future work.*

4 CONCLUSIONS

In summary, our main results are listed as follows:

- Although dominate the real deployment, veth based networking (used with OVS or Linux bridge) is inferior in performance compared with MACVLAN. The overhead of veth limits the overall throughput of a system to under 130 Gbps even with 16 CPU cores with the CHC topology.
- Generally speaking, core binding can improve the performance when flow number is small (no bigger than 1/4 of the CPU core number of a single NUMA node).
- Install containers inside VMs introduces in around 50% performance loss and may cause severe scalability problem for multiple flows.
- Containers in the same VM might have even better performance than in the same bare metal host because of the auto NUMA scheduling.
- TCP throughput between the containers in the same host is almost irrelevant to packet size.
- UDP throughput is significantly lower than that of TCP.
- Windows performance is significantly lower than that of Linux.

ACKNOWLEDGMENT

The authors would like to thank anonymous reviewers for their valuable comments and Ben Pfaff for his instructive shepherding. This work is supported by the National Science and Technology Major Project of China under Grant Number 2017ZX03001013-003, the Fundamental Research Funds for the Central Universities under Grant Number 0202-14380037, the National Natural Science Foundation of China under Grant Numbers 61602194, 61373130, 61370226, 61672156, 61402198 and 61321491, the National Science Foundation under Grant Numbers CNS-1318563, CNS-1524698, CNS-1421407, and IIP-1632051, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

REFERENCES

[1] Roberto Bonafiglia, Ivano Cerrato, Francesco Ciaccia, Mario Nemirovsky, and Fulvio Risso. 2015. Assessing the performance of virtualization technologies for nfv: a preliminary benchmarking. In *Software Defined Networks (EWSDN), 2015 Fourth European Workshop on*. IEEE, 67–72.

[2] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. 2015. An updated performance comparison of virtual machines and linux containers. In *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium on*. IEEE, 171–172.

[3] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. 2015. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine* 53, 2 (2015), 90–97.

[4] Docker Inc. 2017. Docker container networking. https://docs.docker.com/engine/userguide/networking/. (2017).

[5] Rick Jones. 2007. Netperf homepage. *http://www. netperf. org/netperf/Netperf-Page. html* (2007).

[6] Dirk Merkel. 2014. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal* 2014, 239 (2014), 2.

[7] Roberto Morabito, Jimmy Kjällman, and Miika Komu. 2015. Hypervisors vs. lightweight virtualization: a performance comparison. In *Cloud Engineering (IC2E), 2015 IEEE International Conference on*. IEEE, 386–393.

[8] J Pettit. 2014. Accelerating open vswitch to ludicrous speed. (2014).

[9] Luke Welling and Laura Thomson. 2003. *PHP and MySQL Web development*. Sams Publishing.

[10] Herbert Xu. 2009. Generic Receive Offload. In *Japan Linux Symposium*.