Comparative experimental analysis of Docker container networking drivers

Lucas Litter Mentz, Wilton Jaciel Loch, Guilherme Piêgas Koslovski
Graduate Program in Applied Computing (PPGCA) – Santa Catarina State University (UDESC) – Joinville, Brazil
{lucas.mentz, wilton.loch}@edu.udesc.br, guilherme.koslovski@udesc.br

***Abstract*—Virtualization allows for more efficient hardware usage by allowing several instances of Operating Systems (OSs) or Virtual Machines (VMs) to run on a physical server. Containers are a subset of lightweight virtualization and reduce the overhead of virtualizing an entire OS by sharing the server's OS with the virtualized instances. Moreover, containers work closer to hardware than VMs and are similar to Linux processes, however, this limits connectivity freedom since processes do not have access to network addressing. To resemble container's communication to that of conventional networks we use network drivers. Studies show that in processing- or memory-bound scenarios, containers perform better than VMs, but in network-bound scenarios they achieve less performance. This work analyzes performance of networking implementations for Docker in different container allocations and workload scenarios.**

***Index Terms*—Cloud networking, Docker, containers, experimental, drivers**

I. Introduction

The search for greater hardware utilization on data centers has advanced with the use of technologies to execute several guest environments on a single server. Hardware virtualization better leverages physical resources by allowing multiple OSs to run logically isolated over an hypervisor – minimal OS destined to provide the needed VM hosting functionalities [1]. However, virtualization also implies on additional processing overhead and memory usage, since each VM runs a complete OS [2]. Lightweight virtualization deals with this problem by substituting the guest OS by an isolated environment that runs as a process on the host machine [3].

Containers are a form of lightweight virtualization that works by partitioning the host OS on logically isolated environments. Since the containers share the OS with the host machine instead of running individual complete instances, there is a reduction on processing overhead and both memory and disk usage when compared with VMs. Containers are utilized by tech giants like Google [4] and pave Container as a Service (CaaS) services like Amazon AWS Elastic Container Service. Among popular container implementations Docker [5] was chosen for this study for being a widely used platform.

One of the main differences between VMs and containers relates to the communication network. The hypervisor provides to each VM a virtual interface, which to the guest system works as a real network interface, while container communication needs more ingenious solutions since a container is analogous to a Linux process and therefore does not receive an individual address on the network, although it can create communication sockets. To approximate the functioning of container networks to regular ones, solutions named network drivers have been proposed, which abstract container networks to provide different forms of connectivity. Docker has as default five network drivers: *none*, *host*, *bridge*, *macvlan* and *overlay* [6]. With the extra functionality added to the container network comes processing overhead and as each Docker driver has a different behaviour related to the involved processes, the expected overhead is proportional to the process's complexity.

Related work presents performance reduction on container networks over executing the same tests on the host itself [1], [2]. Due to the large utilization of containers and the observed communication performance degradation, this work aims at experimentally identifying the additional load required by Docker container networks and its impacts. The tests also indicate the differences that allow selecting one network driver that objectively stands out on performance according to the studied metrics, independent from the utilization scenario. Specifically, the goal of this work is the empirical analysis of communication network's performance on Docker containers. We provide experimental results to guide the selection of an appropriate Docker network driver given a predefined allocation of containers. In addition to performing analyses based on synthetic flows (as used by specialized literature, Section II-C), we applied flow distributions based on DC reports (Section II-B).

Our experimental methodology comprises four representative workload scenarios (reference, interference, congestion and Data Center (DC) simulation) executed atop multiple containers allocations. Besides network performance indicators (throughput, latency, and Flow Completion Time (FCT)), the CPU metric is used to observe the impact of a CPU-intensive load on network metrics as well as the processing consumption of each driver. Results indicate that, in general, drivers with simpler implementation but more complex usage show better results than complex drivers with easier usage. In real-world traffic simulations we observe run times up to 20% longer using Docker *overlay* than with simpler drivers, and synthetic workloads show a staggering 95% slower achievable bandwidth with the *overlay* solution on some cases.

This paper is organized as follows. Section II presents the

This work was supported by FAPESC, LabP2D/UDESC, CAPES, and CloudLab.

literature review, motivation, and related work. The experimental methodology is described in Section III, while results are discussed in Section IV. Section V concludes the work.

II. Literature Review and Motivation

The Docker container networking is revised (Section II-A), as well as the key observations from multitenant DCs (Section II-B). Finally, the related work (Section II-C) summarizes the discussions from experimental analysis performed with Docker-based containers.

A. Docker Container Networking

As containers are analogous to processes, whose network capabilities are limited to sockets, there is the need to implement different forms of communication so that containers can communicate on a network as if they were regular hosts. These implementations are called network drivers and provide from basic connectivity, by associating the container to a virtual bridge, to scalable solutions of overlay networks comprising service discovery, load balancing and name resolution service.

The network drivers included on the Docker default installation are [6]: *None*, which does not provide network communication to containers. *Host*, which represents the communication through sockets, where each container holds one or more server ports directed to itself. *Bridge*, which creates a virtual bridge to connect the containers to valid IP addresses on a private subnet with local scope. *Macvlan*, which connects the containers to the external network and provides valid MAC (Media Access Control) addresses to them, similarly to how hardware virtualization handles networking. The containers' IP address is chosen by Docker in a subnet defined by the user upon network creation. Any device reachable in the defined VLAN can communicate with the containers on it. *Overlay*, which creates an overlay network that allows containers to communicate even if located on different hosts. Optionally, all the traffic of the overlay can be encrypted.

A representation of a possible container communication is presented on Figure 1. Container #1 has connectivity using the drivers *macvlan* and *bridge*, being accessible both externally, through VLAN 10 on the external network, and internally by Container #2 with which it shares the *bridge* connection. Container #2 is also connected to a Docker overlay network, which allows it to access containers hosted in other servers, such as Container #4. Container #3 uses only the driver *host*, exposing its port 5000 on the IP address of Server #2.

For the analysis we selected only the Docker network drivers that provide networking: *host*, *bridge*, *macvlan* and *overlay*. The four drivers can be used to create a service spanning one or more Docker nodes, however applying a driver that is not recommended for a certain allocation scenario generates performance loss, isolation reduction, additional configuration or greater usage complexity.

B. Cloud DC Network Load

In order to perform experiments with reasonable representation of a DC scenario, it is necessary to understand the behavior of a DC network, which according to [7] present workloads

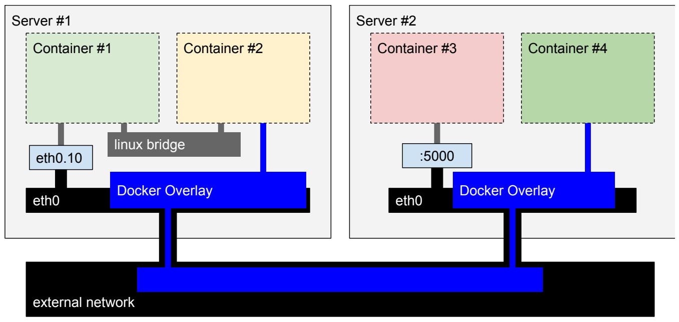


Fig. 1: Docker network configurations example.

that are directly dependent on the executed applications. The analysis of communication networks is conventionally done by observing and identifying the transferred packets, which provides information granularity proportional to the sampling rate and allow abstractions to more high level semantic structures like *flows*, for example. A flow is a TCP/IP structure that comprises a set of packets containing the same values for origin IP, origin port, destination IP and destination port, being represented by a tuple with these values [8].

Data Centers have much higher internal than external traffic – entering or leaving the DC – load due to the distributed services that are executed on them. For instance, upon a search on a search engine the external traffic to and from the DC is composed only by the request and the reply with the final results page, however, internally several services work to aggregate the indexed content and select the proper data to build the page. [9] estimates that the traffic inside a DC represents more than 3 times the traffic entering or leaving it.

Specifically, [8] analyzed the traffic of a 1500 server DC network and highlighted the following traffic characteristics: *(i)* 50% of the flows last less than 100 milliseconds; *(ii)* 80% of the flows last 10 seconds or less; *(iii)* virtually no flow surpasses 100 seconds; and *(iv)* about 50% of all the traffic is composed of flows of 25 seconds or less. The work of [10] analyses 10 DC spanning commercial clouds, private companies' DCs and educational institutions. In summary the study shows that 40% of the flows have less than 1 KB, 80% have less than 10 KB and almost all flows fall below 10 MB; the flow tail duration was 200 seconds with median varying between 0.5 milliseconds and 1 second [7], [10].

Finally, [11] analyses the traffic of Hadoop, Caches and Web Servers of a Facebook DC regarding flow duration, traffic volume, among other measures. This work shows median flow durations varying from 1 millisecond for Hadoop intra-rack traffic to 300 seconds for Web Servers inter-rack traffic, and median flow volume from 500 bytes to 100 KB. The articles converge in the characterization of the traffic as being mostly composed of low volume flows, generally with only tens of KBs of transferred size, while the occasional transfers of high volume flows are the exception. Moreover, services that tend to be more outward-facing like web servers have higher flow duration than more processing-oriented traffic such as reported by [8]. This information is fundamental to compose the traffic workloads used in Section III-A.

C. Related Work

The work of [1] compares the performance between VMs and containers, addressing CPU, memory, disk and connectivity indicators. Specifically for TCP-based tests, Docker containers achieved results virtually identical to the native configuration for unidirectional flows, while it presents 19% overhead for an incremental service request scenario. [12] evaluates network bandwidth between VM with KVM and containers with Docker and LXD on intra-server, intra-cloud and inter-cloud scenarios. For the intra-server scenario Docker containers outperformed KVM and LXD one hundred-fold, while on intra- and inter-cloud configurations Docker achieved a less pronounced advantage of around 30%. In turn, [13] quantifies the performance of Docker-based microservices. In addition to the standard Docker network drivers, the authors apply Software-Defined Networking (SDN) and encryption to route and isolate, respectively, TCP flows between the microservices highlighting that the degradation of network performance with containers is not negligible.

Docker overlay network drivers were investigated in [14]. For long-distance networks, the authors observed a reduction in performance when the CPU is loaded, and according to the packet size. Also, for multi-hop configurations, a predictable growth in latency was observed. The work of [2] discussed experiments with container networks running on VMs. Results vary from close to the native VM traffic rate to a reduction up to $1/4$ of the total bandwidth, with latency depending more on the use case of containers (internal load).

The related work covers a collection of Docker network drivers and scenarios with different performance measures. We observe interest in using overlay networks. Despite imposing a reduction in performance, overlays deliver flexible connectivity in scenarios with multiple hosts. This present work deepens the discussion on Docker network performance by composing different container allocation scenarios combined with reproduction of DCs traffic patterns.

III. Experimental Methodology

The experimental methodology comprises four TCP-based workload configurations and four container allocation scenarios (Section III-A). The experiments were performed on two private experimental clouds (Section III-B).

A. Workload and Allocation Scenarios

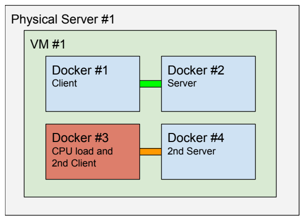
The experimental workload is organized in 4 configurations termed reference, interference, congestion and DC simulation. The reference workload identifies the best-case bandwidth (accounted with iperf3 [15]) and latency (measured with SockPerf [16]) for 2 communicating containers. Interference measures the effect of a stressed CPU on previous scenario: a CPU-intensive container (based on stress-ng [17]) is introduced to stress the environment pointing out the impact on reference bandwidth and latency values. The congestion workload uses 2 pairs of containers to represent background and foreground traffics. The background traffic stresses the network with iperf3 while the foreground assesses the drivers' ability to promote fairness in congested networks.

DC simulation workload is based on Section II-B. The Empirical Traffic Generator [18] is used to simulate traffic to one container (client) from three containers (servers) executed on 3 allocations: same VM, same server, and same rack. Empirical Traffic Generator (ETG) can reproduce traffic following a specified flow size distribution. For DC simulation we replicated data from distribution $PRV2_1$ from [10]: 20% of flows are under 100 Bytes; 40% under 450 Bytes; 50% under 900 Bytes; 60% fall below 1400 Bytes; 70% under 2200 Bytes; 80% are lower than 4500 Bytes; 90% under 20 KBytes; 95% under 100 KB; 99% are below 1.5 MB with the largest flow reaching 2.2 MB. To deepen the analysis, two workload configurations were used for DC simulation, termed standard (1 Gbps load and $250,000$ flows) and heavy (3 Gbps load with $500,000$ flows) configurations.

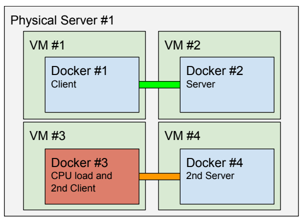
The four workloads are executed on distinct allocations of physical servers to host containers. The rationale behind this approach is to investigate the performance of Docker network drivers when communicating through switches or internally on servers. We argue that the results can assist users on choosing the driver based on the intended container placement. For Allocations 1, 2, and 3, the reference, interference and congestion workloads are executed, while DC simulation is applied for Allocation 4. Allocation 1's performance is limited by the loopback network interface of the VM that hosts the containers, while on Allocation 2 the limit is defined by the configuration and performance of the hypervisor. The third allocation represents the communication between servers in the same rack through a gigabit switch. The Allocation 4 denotes a union of the other three allocations to simulate a DC in which the distribution of containers can lead to situations where a container communicates with co-hosted containers and also distant containers. The experimental allocation scenarios are summarized by Figure 2. The studied traffic is shown as the green lines between containers while orange lines denote background traffic for congestion workload and the orange container runs the CPU load on interference workload. Each experiment was repeated 20 times, and the results are presented as throughput, latency, CPU load, and FCT metrics.

B. Experimental Clouds

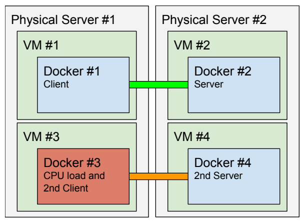
The experiments were executed on two private clouds, Tche cloud and CloudLab [19], managed by OpenStack [20]. For Tche cloud, the two compute nodes are equipped each with two Intel® Xeon® $E5-2620$ v2, 12 cores per server, 192 GB and 160 GB RAM, 500 GB HDD storage, while CloudLab offers HPE Proliant XL170r servers composed of Intel® Xeon® $E5-2640$ v4 with 10 cores, 64 GB RAM, and 480 GB SSD storage each. For both infrastructures, the physical network between servers was configured to 1 Gbps. Specifically for CloudLab, *macvlan* driver cannot be used because CloudLab provisions networking for experiments allowing



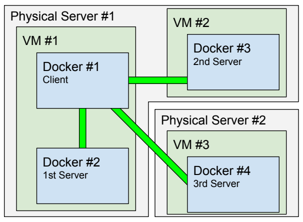
(a) Allocation 1.



(b) Allocation 2.

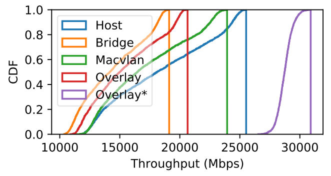


(c) Allocation 3.

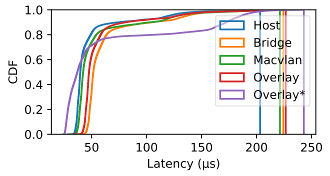


(d) Allocation 4.

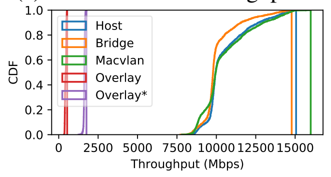
Fig. 2: Experimental allocation scenarios.



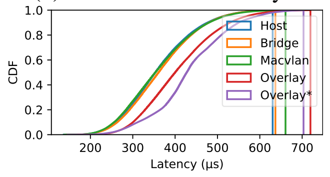
(a) Allocation 1 - throughput.



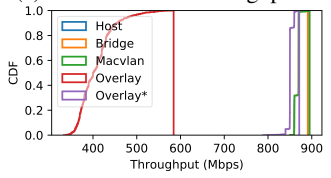
(b) Allocation 1 - latency.



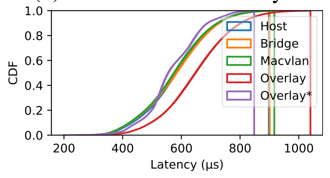
(c) Allocation 2 - throughput.



(d) Allocation 2 - latency.



(e) Allocation 3 - throughput.



(f) Allocation 3 - latency.

Fig. 3: Results for reference workload.

only some specific Virtual Local Area Network (VLAN) tags to flow through. In both physical clouds, VMs had 2 GB RAM, 2 vCPUs and 40 GB storage, executing Ubuntu Server 18.04.2 LTS. All synthetic workloads (reference, interference, and congestion) were executed atop Tche cloud, while DC simulation was performed on CloudLab.

IV. Results and Discussion

The results discussions follow the workload order: reference, interference, congestion, and DC simulation, presented in Sections IV-A, IV-B, IV-C, and IV-D, respectively.

A. *Reference Workload*

Figure 3 presents the plots with the results of the reference workload tests with all allocations. On Allocation 1 we observe a highly divergent performance with *host* and *macvlan* having better results than *overlay* and *bridge*. Interestingly, *overlay* has a higher throughput than bridge, even though it uses a virtual bridge for local communication and therefore expected a similar result to Docker bridge. Figure 3(b) presents the distribution of latency results. These results show repetition of the *bridge* driver on the last position, with the highest observed latency, and again *host* with the best performance. These results corroborate with the expectation that drivers with simpler implementation would have better performance, with the exception being *bridge* having the worst result. With respect to the use of processing (Figure 6(a)), we notice that the biggest portion of work for all the drivers is dedicated to system processes, suggesting that the drivers benefit from the performance assurances promoted by the kernel space instead of the user space.

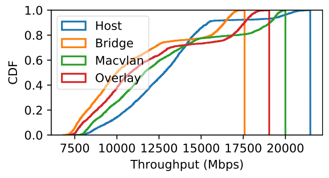
For Allocation 2 (Figures 3(c) and 3(d)), the first observation is the low performance of *overlay*, reaching a mean throughput of only 437 Mbps while the other drivers reached around 10 Gbps. For obtaining such a distant result from the other drivers, the tests of this scenario with the *overlay* driver were repeated on the CloudLab infrastructure and labeled *overlay**. We highlight the discrepancy between the results of this driver executed on Tche cloud and on CloudLab, with the latter reaching numbers up to four times higher, however these results are similar in the aspect that they are still a lot smaller than the results of other drivers on the same test, and this difference could be attributed to the servers on CloudLab having a more modern architecture, with faster processors and memory than the ones used on Tche cloud. The repetition of the behavior of massive throughput reduction on Allocation 2 with *overlay** leads us to believe that it is because of the limitations imposed by the Docker *overlay* driver implementation, however we have not identified any specifics that explain the performance loss. As the CPU utilization of *overlay* has also diminished close to the numbers observed on Allocation 3, we suggest the source of throughput reduction might be programmed instead of being processing capacity limitations.

Another observation is the slight reduction on the performance of the *bridge* driver, accompanied by an unusually large portion of the CPU usage dedicated to software interrupts, compared to other drivers. On the latency test, presented by the plot on Figure 3(d), it is interesting to notice that *overlay* does not present the same level of performance degradation as was seen on the throughput test, although it was an average of 13% higher. This slightly larger latency is expected, considering the additional processing required by the encapsulation and encryption provided by the driver. Still on this test we observe that *bridge* driver had better results, close to the *host* and *macvlan*, agreeing with the initial hypothesis that its less complex implementation would imply on better performance.

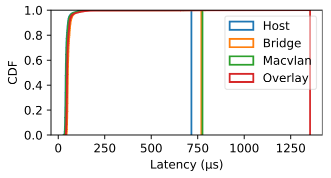
Figures 3(e) and 3(f) present the results from Allocation 3. In the bandwidth test we can once again observe that the results of *overlay* are far below the other drivers. The comparison between these numbers and the ones from Allocation 2 show that the results are quite similar, reinforcing the suspicion that the implementation is responsible for the lower performance. In the latency test it is possible to see the same image of Allocation 2, with *host*, *bridge* and *macvlan* drivers achieving similar values and *overlay* reaching numbers from 10% to 13% worse. For all drivers on this Allocation there was little CPU usage, with the biggest part being dedicated to software interrupts and system operations.

B. *Interference Workload*

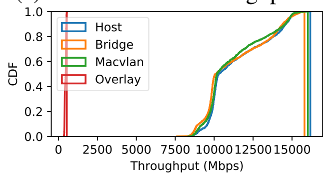
The second scenario, interference, adds a container in the same host of the client container on the tests. Figure 4 highlights the results for all allocations. For Allocation 1, the first observation is the reduction in the mean throughput on all the drivers compared to the reference scenario. The *host* and *macvlan* drivers lead in performance, followed by an 8% slower *overlay* and *bridge* with another 8% lower numbers. Regarding latency, the *host* and *macvlan* drivers have once again close results and about 15% to 25% smaller latency than the others. The processing utilization (Figure 6(b)) on this allocation presents a significant portion dedicated to user processes due to the execution of *stress-ng* for loading the processor on another container on the same VM. Besides *usr*, it is possible to identify that the *soft* and *sys* fractions follow a similar relation, but lower overall, to that observed on the reference scenario with Allocation 1.



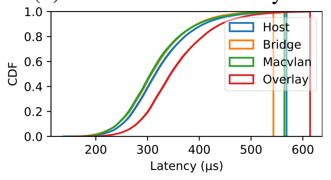
(a) Allocation 1 - throughput.



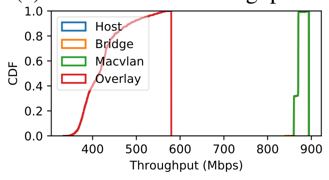
(b) Allocation 1 - latency.



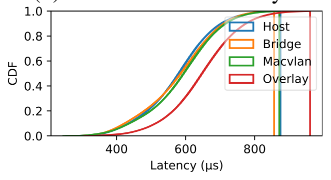
(c) Allocation 2 - throughput.



(d) Allocation 2 - latency.



(e) Allocation 3 - throughput.



(f) Allocation 3 - latency.

Fig. 4: Results for interference workload.

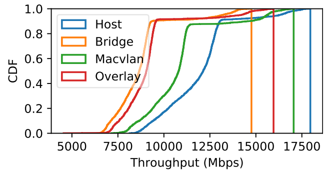
From the observed on Allocation 1 it was expected a slight reduction on the performance with the distribution of the containers on distinct VMs (Allocation 2), although Figure 4(c) and 4(d) presents the opposite. It is possible to see a higher throughput concentration than the ones from the reference workload, and the drivers' performance is roughly equivalent except by *overlay*, which again shows results closer to the expected from Allocation 3. On latency the results are once again surprising, with all the drivers reaching latencies about 10% lower than the reference scenario. The CPU usage (Figure 6) is similar to the one seen on the reference workload mostly because in this allocation the CPU stress container is on a different VM than the ones used for the bandwidth tests, thus its effect is not observed on the plot of processor usage.

For the Allocation 3 we find results slightly better than the observed for the same allocation on the reference scenario. The bandwidth and latency tests' results are presented on Figures 4(e) and 4(f). On the bandwidth test, the results are identical to the ones from the reference workload for the *host*, *bridge* and *macvlan* drivers, with a minor improvement on the *overlay* mean throughput, indicating that for the assessed network connection (1 Gbps) the applied CPU workload does not affect negatively the drivers' performance. These observations can be partially explained by the fact that, even being on the same physical machine, the load in two processing cores of one VM is not enough to impact the overall capacity of the underlying 12 core physical host to the point of suffocating the processing capabilities of other VMs.

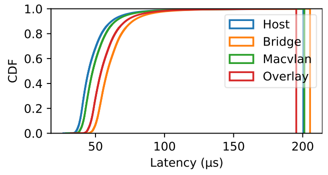
C. *Congestion Workload*

The third synthetic test scenario is congestion, whose execution is composed of two pairs of containers where one pair is responsible for saturating the network, while the other executes the bandwidth and latency tests. On the first allocation (Figures 5(a) and 5(b)) it is expected for the observed throughput to be close to half of the link's capacity for the foreground flow – the evaluated one, which competes with an already existing background flow – but the actual results show a reduction of 35% to 40% for all the drivers when compared to the reference workload. On latency we see values similar to the reference workload, despite the competition for bandwidth with the background flow. It can be observed that the best latency results are again with *host* and *macvlan*, while *overlay* and *bridge* reach results up to 25% slower.

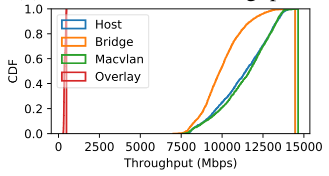
On Allocation 2 (Figures 5(c) and 5(d)), instigating results are observed. For the first time there is a higher achieved bandwidth on Allocation 2 than on Allocation 1, being the *bridge* driver accountable for such result, with 14% higher throughput. The other drivers maintain similar results to the Allocation 1 on this workload, except the *overlay* which gets slowed even further to an average of 377 Mbps. The processing usage indicated on Figure 6(c) presents a noticeable difference with increases of usage between 30% and 50% in respect to the observed on Allocation 2 of reference scenario for the *host*, *bridge* and *macvlan* drivers. One can conclude that between distinct VM pairs, even being on the same physical host, there is no significant competition. A possible explanation could be the underutilization of the available resources on the physical machines, since the server is equipped with 12 processing cores and only 8 cores are utilized (4 VMs with 2 cores each), allowing distinct and isolated work between the VMs. The latency tests shows close values between *host*, *bridge* and



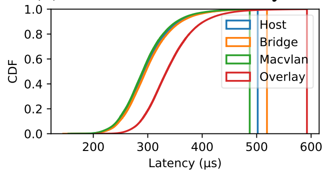
(a) Allocation 1 - throughput.



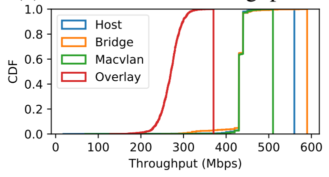
(b) Allocation 1 - latency.



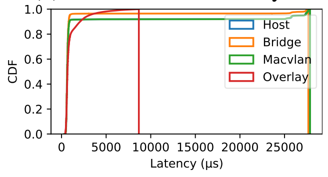
(c) Allocation 2 - throughput.



(d) Allocation 2 - latency.

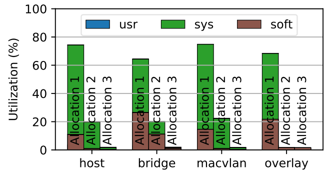


(e) Allocation 3 - throughput.

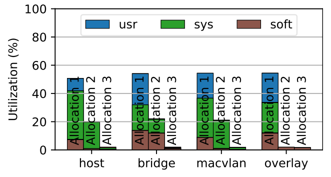


(f) Allocation 3 - latency.

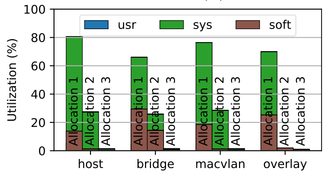
Fig. 5: Results for congestion workload.



(a) Reference workload.



(b) Interference workload.



(c) Congestion workload.

Fig. 6: CPU usage for reference, interference and congestion workloads.

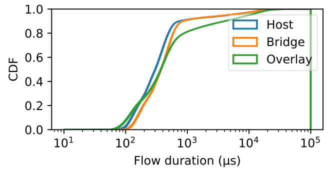
macvlan with *overlay* having 15% more latency. In comparison with the reference workload with the same allocation scenario we see a reduction in latency of about 18%.

The last allocation of the synthetic tests puts two flows to compete for the 1 Gbps bandwidth between VMs hosted on different servers. Figures 5(e) and 5(f) show the results. It is noticeable that for the *host*, *bridge* and *macvlan* drivers the obtained throughput is half of what they reached on the same configuration of the reference scenario and half of the allocation's available bandwidth. With *overlay*, the measured bandwidth is 66% of the achieved on the reference workload, indicating that the total throughput was larger with two flows than with only one on the reference scenario, although still a worse performance than the other drivers. There were no perceptible differences on the CPU usage of this specific test with relation to the reference or interference workloads. The latency test presents no similarity to the other workloads, with a surprise being *overlay* having the best results. The *host*, *bridge*, *macvlan* and *overlay* drivers had, respectively, 389%, 187%, 397% and 90% higher latencies compared to the reference workload with Allocation 3.

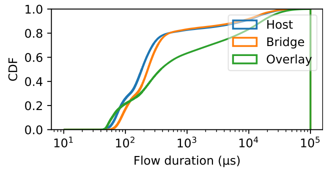
D. DC Simulation Workload

This section presents the results and analysis of the Data Center traffic simulation tests executed with the ETG tool. Figure 7(a) presents the Flow Completion Times. The *overlay* presents better FCTs on the 20% smaller flows when compared to the other *drivers*. Since these flows have very short durations, it is suspected that they represent traffic on the same VM or on different VMs on the same physical machine, contrasting with the results obtained on the synthetic tests, where the *overlay* obtained the worst results in both bandwidth and latency. On the execution times, *host* has the better performance followed by *bridge* with 5% and *overlay* with 6.8% higher durations. These numbers agree with the observations of the reference workload to which this one more closely relates.

The heavy test (Figure 7(b)) exacerbates the discrepancy between *overlay* and the other drivers. The higher requested bandwidth results in *overlay*'s maximum bandwidth on different VMs and on different physical machines being achieved more regularly. For this reason the *overlay* has a 13.8% higher test duration than *bridge* and 20.1% higher than *host*. *Bridge* has a 6.2% slower execution than host.



(a) Standard workload.



(b) Heavy workload.

Fig. 7: Results for on DC simulation workload.

V. Conclusion

The containers' communication limitations, arisen from the Linux process alike functioning, impelled the development of network drivers that add more traditional communication capabilities, which also implies overhead proportional to their complexity. This work executed an evaluation of Docker network drivers *host*, *bridge*, *macvlan* and *overlay* on synthetic and simulation experiments. The quantitative focused metrics involved both common evaluation dimensions like bandwidth, latency and CPU usage and also a rather unused one, the Flow Completion Time. Our findings suggest that applications that require higher bandwidth and lesser latencies steer away from the most simple-to-use Docker *overlay* driver and consider using *macvlan*, a good compromise between *host*'s performance and *overlay*'s ease-of-use.

REFERENCES

[1] R. Morabito, J. Kjällman, and M. Komu, "Hypervisors vs. lightweight virtualization: a performance comparison," in *2015 IEEE International Conference on Cloud Engineering*. IEEE, 2015, pp. 386–393.

[2] K. Suo, Y. Zhao, W. Chen, and J. Rao, "An analysis and empirical study of container networks," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 189–197.

[3] S. J. Vaughan-Nichols, "New approach to virtualization is a lightweight," *Computer*, vol. 39, no. 11, pp. 12–14, 2006.

[4] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 2015, p. 18.

[5] DOCKER, INC., "Enterprise container platform — docker," aug 2019. [Online]. Available: https://docker.com/

[6] ——, "Overview — docker documentation," 2019. [Online]. Available: https://docs.docker.com/network/

[7] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1492–1525, 2017.

[8] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*. ACM, 2009, pp. 202–208.

[9] CISCO, "Cisco global cloud index: Forecast and methodology, 2015-2020 white paper," *Retrieved 1st June*, p. 15, 2016.

[10] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[11] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.

[12] I. M. A. Jawarneh, P. Bellavista, L. Foschini, G. Martuscelli, R. Montanari, A. Palopoli, and F. Bosi, "Qos and performance metrics for container-based virtualization in cloud environments," in *Proceedings of the 20th International Conference on Distributed Computing and Networking*. ACM, 2019, pp. 178–182.

[13] N. Kratzke, "About microservices, containers and their underestimated impact on network performance," *CLOUD COMPUTING 2015*, pp. 165–169, 2015.

[14] A. Zismer, "Performance of docker overlay networks," *University of Amsterdam*, 2016.

[15] iPerf, "iperf - the tcp, udp and sctp network bandwidth measurement tool," 2019. [Online]. Available: https://iperf.fr/

[16] Mellanox, "Mellanox/sockperf: Network benchmarking utility," 2019. [Online]. Available: https://github.com/Mellanox/sockperf

[17] C. I. King, "stress-ng," 2019. [Online]. Available: https://kernel.ubuntu.com/~cking/stress-ng/

[18] CISCO, "datacenter/empirical-traffic-gen: Simple client-server application for generating user-defined traffic patterns," 2019. [Online]. Available: https://github.com/datacenter/empirical-traffic-gen

[19] D. Duplyakin, R. Ricci, A. Maricq, G. Wong, J. Duerig, E. Eide, L. Stoller, M. Hibler, D. Johnson, K. Webb, A. Akella, K. Wang, G. Ricart, L. Landweber, C. Elliott, M. Zink, E. Cecchet, S. Kar, and P. Mishra, "The design and operation of CloudLab," in *Proceedings of the USENIX Annual Technical Conference (ATC)*, Jul. 2019, pp. 1–14. [Online]. Available: https://www.flux.utah.edu/paper/duplyakin-atc19

[20] OpenStack, "Build the future of open infrastructure," 2019. [Online]. Available: https://www.openstack.org/