Exploring Network Contention in Containers

Conrado Boeira
PUCRS
conrado.boeira@edu.pucrs.br

Tiago Ferreto
PUCRS
tiago.ferreto@pucrs.br

***Abstract*—The popularity of newer architecture models such as microservices has given rise to the use of containers in Cloud environments. To that end, multiple studies on network optimization in containers have been conducted, with the end goal of allowing more technologies that depend on high communication performance to migrate to this type of architecture. In this work, we focus on exploring the contention for network resource in these environments. Considering that a server may host tens to hundreds of containers, each one with its own virtual network interface, we aim to study how containers compete for network resources when communicating concurrently. Besides, due to the availability of different virtual interfaces that can be used in containers, we also investigate how the chosen interface can affect the containers' performance regarding communication. In this exploratory work, we present the results of multiple scenarios of contention using containers and demonstrate the differences in performance between different network stack virtualization techniques, as well as numerically showing the contention for network resources in this context.**

I. Introduction

Cloud computing has become part of the technological landscape we see today. As cloud providers are able to offer Infrastructure, Platform and Software as a Service (IaaS, PaaS and SaaS), small capital initiatives can more easily deploy their solutions and larger enterprises can scale to the size needed, does not matering its proportions [1]. The on-demand, pay-as-you-go model that allows users to quickly spin-up their applications and pay accordingly, resulted in a massively popular solution.

However, as different technologies such as the Cloud Native [2] architecture and Edge computing [3] rise, the need for more easily manageable virtualization techniques rise too. Cloud Native and microservices architecture both advocate for breaking monolithic systems into multiple single function components [2]. These components are normally deployed using containers due to the smaller size and faster boot time they offer in comparison to larger Virtual Machines (VMs). Edge Computing defines a new paradigm where part of the computation in the cloud environment is migrated to smaller nodes closer to the end-users, called *cloudlets* [4], with lesser computational power but with lower delays in communication due to its proximity to end-users. In this case, containers can also be a good fit to not overuse the already small amount of *cloudlets* resources.

Many of these solutions revolve around using multiple containers to conduct distinct computations that together compose a single service. Replicas can be replicated or deleted quickly and communication between all parts of the service is key to achieving a good performance. Many solutions have been proposed for network stack virtualization, which is a crucial part of efficiently reaching satisfactory communication speed between the containers. Virtual network interfaces, such as Linux Bridge [5], Macvlan [6] and Open vSwitch [7], offer different functionalities and are build upon diverse strategies.

Having this in mind, it is important to guarantee that services can achieve a similar performance, with low latency and high throughput, with containers as can be seen when using specific hardware and virtual machines. Hence, it is important to explore how replicas compete for network resources when they are deployed on the same host, as this can be a important point of contention for the service. If specific instances can just utilize a huge portion of the bandwidth offered, it can lead to critical containers that require fast responses not being able to execute inside its time limits and creating a bottleneck for the whole system. Furthermore, with the different network virtualization techniques available, it is crucial to understand their main differences and determine if there are better or worse options when it comes to coping with the large number of concurrent flows created in microservices architectures.

Therefore, we propose a study on contention for the virtualized network stack in containerized environments, using standard tools such as Docker [8]. We focus on exploring the competition for available bandwidth between containers communicating between themselves. Our aim is to define how much the increase of containers using a network can influence in the total throughput. We also study the different virtual interfaces with the goal of finding out if there is one option that performs better and is able to more effectively handle the increase in the number of flows created. Moreover, we also aim to outline how much replicas with larger network workloads can influence others using the same stack, in order to understand how fairly the link between containers is shared when using standard solutions.

The rest of this work is organized as follows: Section II provides background information in the topics explored and the motivation for this work. Section III describes how the experiments were conducted, which scenarios were created and how the measurements were made. Section IV addresses the results of the tests and discuss our main findings, respectively. In Section V we overview some other relevant works in the area. Finally, Section VI presents our final conclusion and future works.

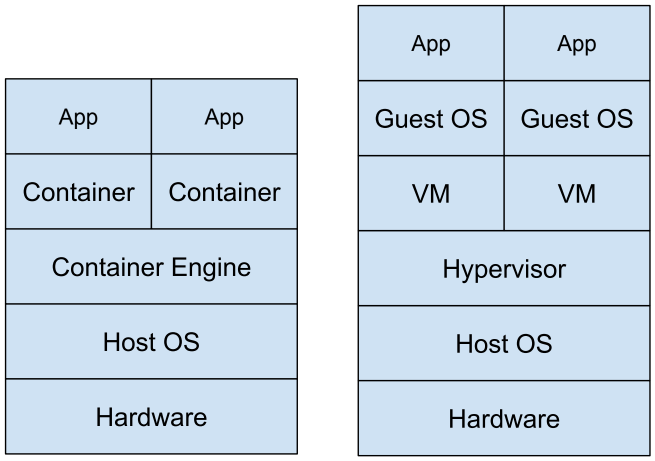


Fig. 1: Comparison between containers and VMs.

II. Background

In this section we define some of the main concepts used in this work, going through virtualization techniques, the virtual interfaces studied and the microservices architecture .

A. Virtualization and Cloud Computing

Cloud Computing is a model for providing on-demand access to resources that can be quickly provisioned and released to users [9]. Providers offer to users the possibility to easily deploy their applications and allow for paying only the amount of resources used. Therefore, in order to allow for multi-tenancy, i.e. multiple users running their software in the same physical host, cloud providers make use of virtualization techniques such as Virtual Machines and containers.

Virtual Machines (VMs) [10] consist on running a guest Operational System (OS) on top of a hypervisor, a software responsible for isolating the guest OS and connecting it to the underlying system. There are two types of hypervisors, Type 1 or bare metal, and Type 2 or hosted hypervisor. Type 1 is installed and runs directly on top of the physical host, and Type 2 runs on top of an already installed OS. Both of them act like a middle-man, among other functions, translating system calls and interrupts from the OS running on top of it to instructions that system running below it can execute upon.

Another approach used is hosting applications on containers. They leverage an engine to use features of the underlying OS in order to provide isolation for the software running on it. Due to that, containers are lightweight and faster to create and deploy then VMs. Figure 1 provides a scheme to present the difference between both virtualization techniques.

B. Virtual Network Interfaces

While making use of VMs or containers, it is necessary to create abstractions for the physical resources available, such as the NIC. Therefore, as multiple approaches were proposed, a diverse number of virtual network interfaces were created in order to connect the virtualized environment to the network.

One of the most common ways to do this is through Linux Bridges [5]. A bridge can work like a switch, connecting all VMs or containers using it and allowing communication between the virtual network and the host network. The Bridge consists of four main components [11]: a set of network interfaces that can be either physical or virtual; a control plane that prevents loops and crashes in the network; a forwarding plane, that forwards packets based on their MAC; and a MAC learning database the stores the values for machines in the network.

Another option is to use a technique called Macvlan [6]. It allows for one NIC (the parent interface) to have multiple MAC and IP addresses assigned to it. This way, each VM or container can have a Macvlan interface attached to it and interact directly with the host network using its own MAC and IP addresses. One of the drawbacks of this interface is that the containers created using it are not able to directly communicate with the parent interface. When connecting multiple containers, Macvlan is normally used in bridge mode. It creates a small bridge to allow for communication between containers. However, as the MAC addresses are known, neither the MAC learning database nor the algorithms to prevent loops in the control plane are needed. This results in a lighter bridge. Nonetheless, differently than traditional bridges, a Macvlan bridge depends on the parent interface and will go down if any crashes occur.

Open vSwitch or OVS [7] is another solution used, consisting of a multilayer software that acts as a switch able to connect virtualized environments even on different physical hosts. OVS has seen a lot of use in the paradigm of Software-Defined Networking (SDN) as it leverages OpenFlow and can operate as the control stack for switching. OVS also supports many commonly used technologies such as NetFlow, sFlows and VLANs.

C. Microservices

Cloud computing has recently been going through impactful changes to the architecture of services deployed. Originally, most applications were deployed as a single monolithic service, which could be hard to manage, replicate and be quickly deployed. These questions gave rise to the microservices architecture, which revolves around breaking an application into multiple single-function services that, connected together, constitute a full system.

This new structure has its key advantages centred around the smaller size of the microservices. With this, an operator can easily manage and spawn new replicas of the specific service needed, not being forced to scale a whole application. Moreover, this architecture allows for services components to be deployed on different physical hosts, leading to a better usage of the resources available.

Microservices are normally implemented using containers as they can be easily deployed and managed. Each service is deployed on a container and the graph containing all of them and their connections constitute the full service. This lead to applications being comprised of a large number of containers. A common type of application, such as a social network system, can have more than 40 containers for a single instance [12], [13].

With the large number of containers that can be involved into instantiating a single application, the contention for network resource they incur can be an important factor when trying to maximize their performance. As many different services in the cloud environment can share a physical host, it is important that this contentiousness be studied for virtual interfaces. Having this in mind and the multitude of options for interfaces exposed in Section II-B, our work focus on defining how much containers contend for virtual network resources and how the different interfaces behave under these circumstances.

III. Methodology

In order to accurately measure the contention for network resources, we prepared multiple scenarios, namely *Throughput*, *FCT* and *Elephant/Mouse* scenarios. With each one, we aim to explore a different facet of this issue. Each scenario can vary the amount of client/server pairs from 10 up to 150, as we tried to explore to the maximum the contention for network resources. We exhibits the difference in performance from the 3 studied virtual interface solutions: Bridge, Macvlan and OVS.

In the *Throughput Scenario*, all clients, concurrently, establish TCP connections to a matching server and start sending packets at the maximum rate possible. We then collect metrics on the average bandwidth utilized by each client and the total throughput achieved in the system during the communication. We then vary the amount of client/server pairs and present data for the total throughput of the system, individual throughput of the clients and number of packets retransmissions in the clients.

Another metric used for network evaluation is the Flow-Completion Time (FCT), which consists on the time for completing a communication flow. A flow can be defined as a set of packets send from one address to another using the same ports and protocol of communication. This measurement can be of great interest in the microservice context, as users are more interested in the time it takes to complete a desired operation, such as downloading a file, than the actual throughput of the network [14]. Therefore, in the *FCT Scenario*, we measure the completion time by having all clients sending a single flow consisting of 5MB of data to their matching server. Finally, we collect the time for each individual client to complete this transmission.

Moreover, an aspect also studied is the changes in performance when the clients do not send flows with the same size. In the *Elephant/Mouse Scenario*, 20% of the clients send a much larger flow (Elephant Flow) consisting of 50MB of data, and the rest of clients send a smaller flow (Mouse flow) consisting of 5MB of data. This would mimic real life behaviour were some applications send larger amounts of data while others only need to send smaller packets. Having this in mind, it is important to study this kind of scenario in order to evaluate if the available bandwidth is fairly divided between services and how much the smaller flows are affected by the larger ones.

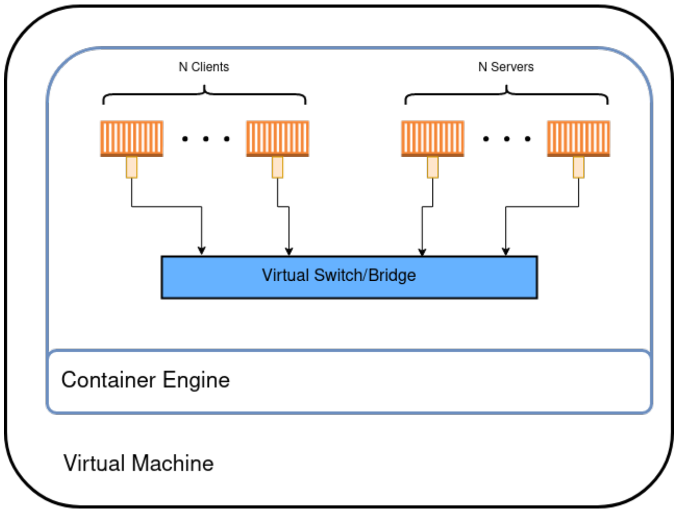


Fig. 2: Diagram of the testbed used for testing.

IV. Results

The described scenarios were executed on top of a virtual machine running Ubuntu Server 18.04.3 LTS with 8 CPUs and 8GB of RAM. This was done as it is common practice for containers to be deployed on top of VMs in CLoud scenarios. The container engine used was Docker 20.10.6, and the clients are running *Iperf* 2.0.10 [15] for the *Throughput Scenario* and Netcat in the other scenarios. No CPU pinning was conducted and the containers were run with default resources attributed by Docker. A diagram of the testbed used for the tests can be seen in Figure 2.

A. Throughput Scenario

The first scenario to be executed was the *Throughput* one. We collected individual throughput measurements of each client and present the complete results in Table I. As it can be seen, there is a steadily decrease in the average bandwidth as the number of pairs increase for all of the interfaces. Comparing them, one can observe that in most of the cases OVS outperforms the two other options, with Macvlan coming in a close second in the majority of times. This is indicated on Figure 3, where the boxplot shows the throughput for 20, 40, 60, 80 and 100 pairs. The total throughput each interface achieves tend to fluctuate around an average, which implies that we are using the total bandwidth that the interface can offer. However, Macvlan seems to slightly increase its performance with higher loads.

Both OVS as Macvlan offer a better average and overall throughput compared to Bridge due to the difference in the underlying implementation. OVS offers a fast-path mechanism that accelerates the communication between containers. It stores a table of known flows and when a matching packet arrives, it is automatically directed to the destination. This whole process occurs in kernel mode, which makes it much faster. In the beginning of the tests the switch will enter user mode initially to address the new packets arriving, but once it receives packets from all flows, it can solely use the fast-path system and achieve the results observed.

Macvlan outperforms the default Docker Bridge due to the fact that it uses a lighter bridge system. As replicas already have their MAC assigned, there is no need to perform MAC learning as is done in the default bridge. Moreover, the Spanning Tree Protocol (STP) used by default in bridges to avoid loops is not present in Macvlan as all replicas already have direct access to the MAC addresses of neighbour containers.

These results match what can be found in other works such as the one proposed by Anderson et al. [16]. In this paper, the authors tested with Virtual Network Function chains with variable sizes and found out that, when running container in the same host, Macvlan and OVS produced similar delay values and achieved lower values than the traditional Linux Bridge. The claim that Macvlan is able to perform better than Bridge in container environments is supported by other works as well, such as [17], [18].

Another detail to be noted is the disparity between the individual throughput of containers while executing the workload. Figure 4 presents a Cumulative Distribution Function (CDF) graph for the throughput of each of 50 clients in the experiments. As can be seen, values can range from less than 1 Gbps up to 2 Gbps. Normally, it would be assumed that the link bandwidth would be shared fairly between all containers, but this result shows otherwise.

In order to better understand this interval of results found when executing the tests, we investigated the number of retransmitted packets for each client. Our first hypothesis was that instances that achieved higher throughput values would have less packets lost and, therefore, less retransmitted packets. However, as can be seen in Figure 5, the presented behaviour seems to be the opposite, with higher retransmission rates in average happening in instances that were able to send more data during the experiment.

In Figure 5 we can see the number of retransmissions per container throughput in a 2 minute test. We have extended the time duration of this specific test due to replicas not having enough time to acknowledge the loss of packets and perform the necessary communication when executing for a single minute. With this test, it is possible to see the correlation between the number of retransmissions and throughput achieved. This happens because instances that would receive CPU and network resources first, could retransmit lost packets first and thus achieve higher performance values.

B. *FCT Scenario*

In the *FCT Scenario*, as expected, the maximum time a client takes to complete the communication rises with the increase of amount of pods. In Figure 6 we can see the Cumulative Distribution Function (CDF) for the FCT results of each client with 10, 50, 100, 150 pairs. It is possible to

Pairs	**Average Throughput (Gbps)**			**Total Throughput (Gbps)**		
	Bridge	Macvlan	OVS	Bridge	Macvlan	OVS
10	5.6389	6.5896	7.1469	56.3890	65.8960	71.4690
20	2.8602	3.0490	3.3197	57.2040	60.9800	66.3940
30	1.8700	2.1482	2.3952	56.1000	64.4460	71.8560
40	1.4409	1.5861	1.7484	57.6360	63.4440	69.9360
50	1.1365	1.3018	1.4112	56.8250	65.0900	70.5600
60	0.9844	1.1145	1.1720	59.0640	66.8700	70.3200
70	0.8675	0.9856	0.9988	60.7250	68.9920	69.9160
80	0.7535	0.8811	0.8818	60.2800	70.4880	70.5440
90	0.6614	0.7892	0.7768	59.5260	71.0280	69.9120
100	0.5884	0.7013	0.6824	58.8400	70.1300	68.2400

TABLE I: Results from the *Throughput Scenario* for the 3 virtual interfaces studied.

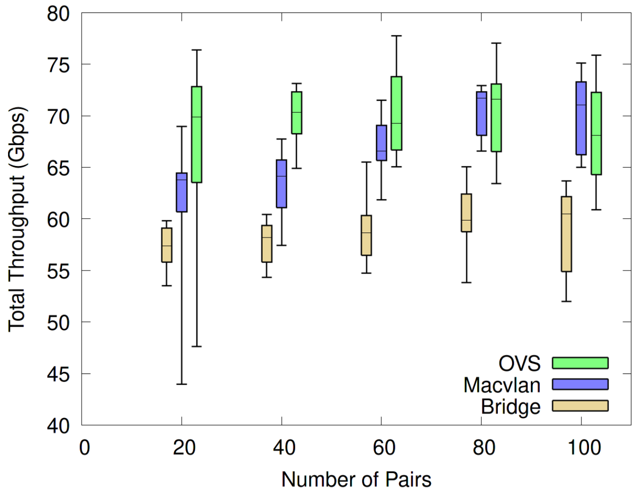


Fig. 3: Total Throughput for OVS, Macvlan and Bridge and 20, 40, 60, 80 and 100 pairs.

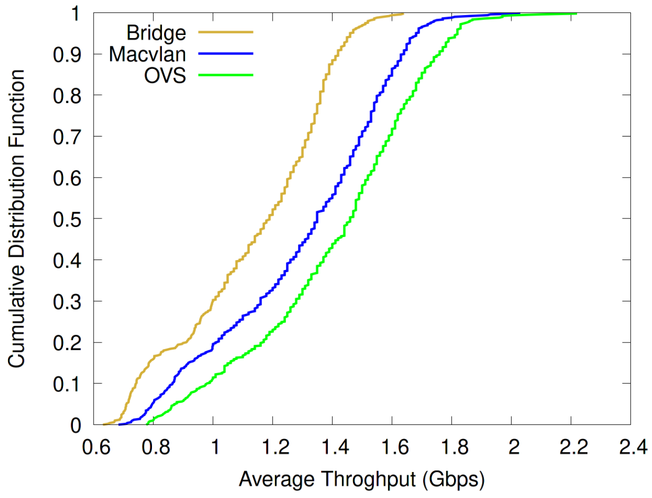


Fig. 4: Cumulative distribution function for the individual throughput of containers in a scenario with 50 pairs.

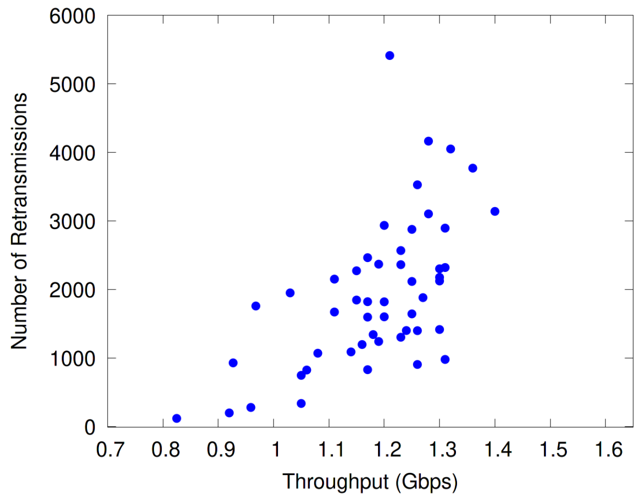


Fig. 5: Number of retransmissions for the throughput achieved in a test with 50 pairs using Bridge as the virtual interface and with test time duration of 2 minute.

analyze in the graphs that, in the four cases studied, OVS has the lowest FCT values for most flows, followed by Macvlan and finally Bridge. The latter present elevated tail values, as can be seen more specifically in Figure 6a, where the largest observed completion time is much larger in comparison to the ones from other interfaces.

Another interesting observation to be made is the interval in which the FCT measurements distribute themselves. In Figure 6a all values are in the range of 0.02 to 0.18 seconds. For 50 values in Figure 6b this range increases to around 0.05 to 0.25 seconds. This is not a large difference but, when the number of pairs goes to 100 and 150, Figures 6c and 6d respectively, the increase is more expressive. For 100 pairs, flows can take up to 3 full seconds and for 150 pairs, this value reach up to 7 seconds.

The difference in performance between the studied virtual network interfaces can be attributed to the same reasons seen in Section IV-A. The OVS fast-path mechanism and Macvlan lighter bridge structure allow the two interfaces to perform better than Bridge. The high difference in FCT for more containers can be explained by the CPU usage. When sending data, containers contend for the CPU cores available to write the data to the network buffer. Moreover, the interfaces themselves also participate in this, as, since all communication is being carried out in the same host, every packet sent heavily depends on the CPU. This could also explain the increase in the FCT upper-bound we can see between 50 and 100, as this can be thought as the point where the CPU contention reaches levels that affect more greatly the performance of most of the containers.

With the *FCT Scenario* it is also possible to see that the time containers can take to send a specific amount of data can greatly vary even between instances running on top of the same host and concurrently. Specific services can take up to 3 seconds to complete a flow while others can finish under a second. This shows that, although the available bandwidth is supposed to be fairly shared between all instances, the contention for the network and subsequently the CPU can lead to great divergence in completion time, which can be the source of problems when trying to guarantee a specific level of performance for services running on top of containers.

C. *Elephant/Mouse Scenario*

The last scenario tested in this work was the *Elephant/Mouse Scenario*. We again tested for the three virtual interfaces and the results for 10, 50, 100, 150 pairs can be seen in Figure 7. It is possible to see that now the difference between the network virtualization solutions are narrower. When looking at the first 80% of the flows, which consists of the mouse flows, the FCT achieved with all interfaces are really similar, although there is still a slightly better performance achieved by OVS, followed by Macvlan and finally Bridge. However, the difference is larger for the elephant flows, last 20%. The highest FCT values reached by Bridge are consistently higher than what can be seen in OVS and Macvlan.

Another observation that can be made from the graphs is how much the elephant flows affect the completion time of the smaller mouse flows. This can be seen when comparing Figure 6b and 7b for 50 pairs. In the *Elephant/Mouse Scenario* the mouse flows can take more than 0.5 seconds to complete, twice more than the upper limit for flows in the *FCT Scenario*.

The results presented once again corroborate, in the studied scenarios, Open vSwitch incurs less CPU usage when compared to the other options, which allows it to perform better in these contention scenarios. Specially in this case, CPU contention is even more of a issue as Elephant flows can use up more of the processor and reduce the efficiency for other flows.

Another take-way from this test is how containers that do not need to send larger amounts of data can be affected by other replicas. This can become an issue when we consider that these containers could have a critical function inside a microservice architecture. For example, containers sending small amount of control data could be drowned out by replicas from a different service that is running in the same host.

Tests with variable workload size have been done before in works such as [17], however, they have worked with only two pairs of containers competing for the resources. Moreover, the scenario here explored is different as it focus on containers on a single VM and on the three diferent virtual interfaces studied, differently than the previously cited work that shows results for this kind of workload only for multi VM scenarios with different interfaces.

V. Related Work

A. *Containers Network Performance*

Zhao et al. [18] performed studies on the varying network interfaces and deployment technologies for containers. The authors first compared multiple different virtual interfaces such as linux bridges and MACVLAN, and found out that, although

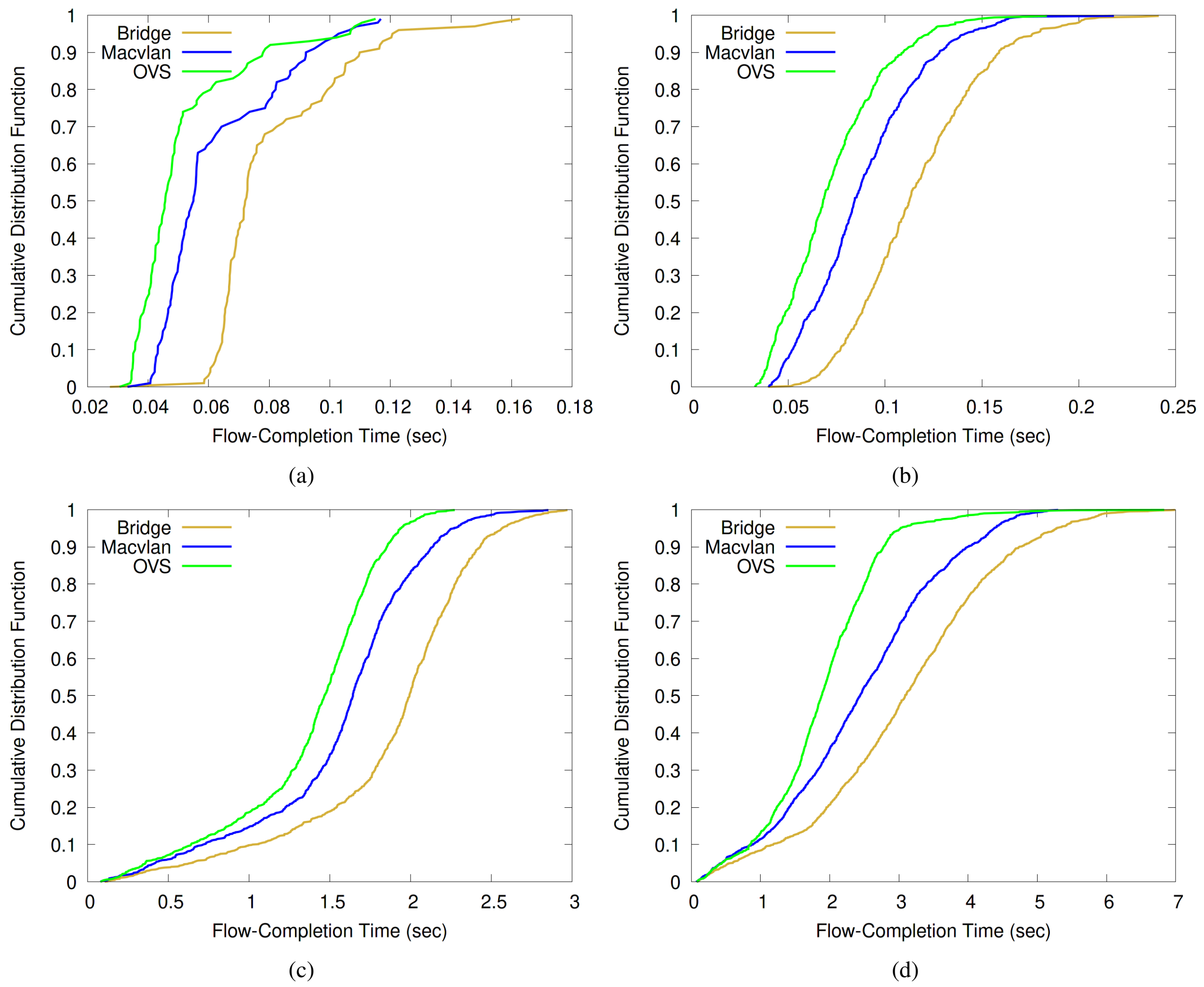


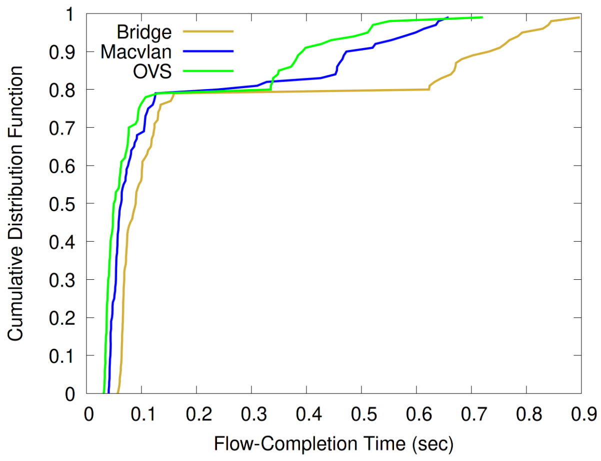
Fig. 6: (a) Cumulative Distribution Function results for the Flow-Completion Time for (a) 10 pairs, (b) 50 pairs, (c) 100 pairs, (d) 150 pairs.

veth based interfaces (as OVS [7] and Linux Bridge) are more commonly used, MACVLAN interfaces exhibited better results in the tests conducted. In the topology aspect, the authors experimented with containers on the same or different physical hosts and deployed on top of Virtual Machines or directly on the host. The authors found that performance drops when containers are placed on different VMS. However, due to the automatic NUMA schedulling, containers placed on the same VM can achieve even faster communication than when running directly on the same host.

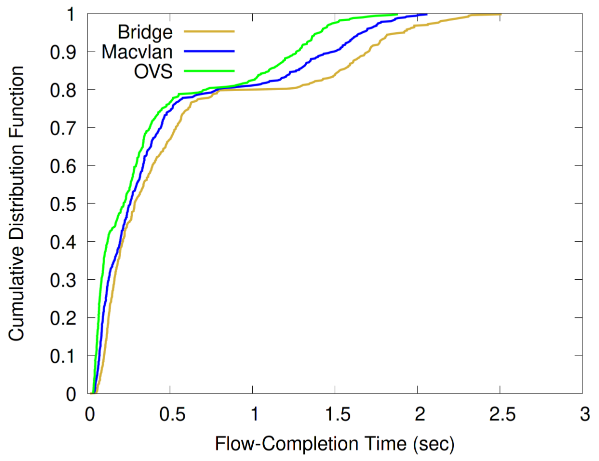
The paper presented by Anderson et al. [16] focus on evaluating the network usage efficiency in VNFs deployed over containers. The author investigates the performance for containers using different virtual network interface options, such as SR-IOV and MACVLAN, similarly to [18]. Two scenarios are explored: a single container running with no competition, and a group of VNFs running in the same hosts and constituting a Service Function Chain (SFC). The authors discovered that in the single VNF case, both MACVLAN and SR-IOV presented better results compared to OVS and Linux Bridge. However, in the SFC scenario, it is MACVLAN and OVS that end up performing the best, while SR-IOV has a drop in performance.

In the work proposed by Mentz et al. [17], a similar study to this work is proposed. The authors compare the standard Docker virtual interfaces (Bridge, Host, Macvlan, Overlay) with different scenarios. They explore the contention for resources with containers communicating while another concurrently stress the CPU of the host. They also studied the effects of multiple containers trying to use the network simultaneously. However, they simulated this using only two pairs of competing client/server container pairs, while we focus specifically on the effects of large scale deployments with multiple replicas.

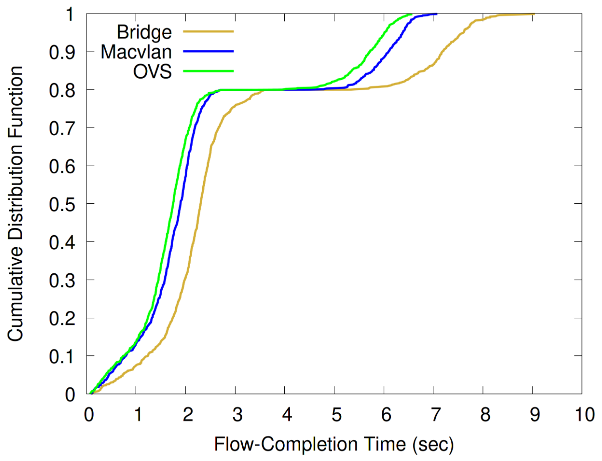
Suo et al. [19] explored and evaluated multiple network solutions for container communication in the same host, such as Bridge, Host mode and container mode, and in different hosts, overlay solutions like Weave [20], Flannel [21] and Calico [22]. They conducted experiments to evaluate network



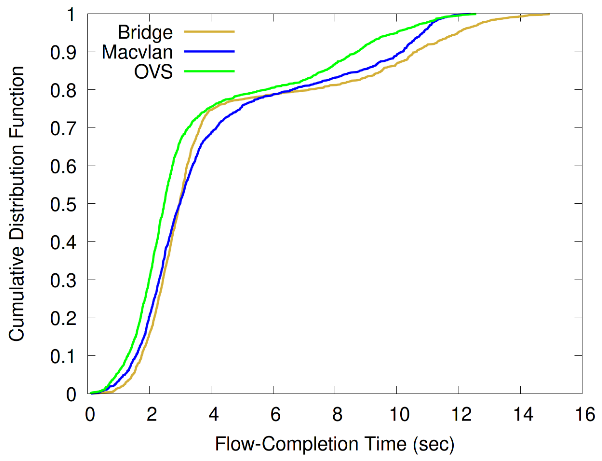
(a)



(b)



(c)



(d)

Fig. 7: CDF results for the Flow-Completion Time in the Elephant/Mouse scenario for (a) 10 pairs, (b) 50 pairs, (c) 100 pairs, (d) 150 pairs.

performance for intra and inter-host communication of containers. They also explored the influence of different packet sizes, the presence of CPU interference and contention for network resources. However they only evaluated the Bridge interface when studying the contention generated by multiple containers in the same host.

In another work, Qi et al. [23] analyzed the differences and techniques used by 5 of the most popular CNI Kubernetes [24] networking plugins: Flannel [21], Kube-Router [25], Calico [22], Cilium [26] and Weave [20]. The authors also compared the performance of these solutions and performed a breakdown of sources of overhead for each of the plugins.

In Table II we present a comparison between the proposals made by the papers discussed above and our proposal, as they are the most similar to our work.

B. Microservices Implementations in the Cloud

The authors of [27] proposed a paradigm called X-Container to execute cloud-native containers with isolation between them. The presented model does not depend on hardware virtualization support and creates an environment where containers are isolated between each other, at the cost of reducing the isolation between processes in the same container. This is done in order to allow for better performance for the services using the architecture. The developed implementation can process containers application and rewrite costly system calls for others that can be directly made to the operational system. The authors were able to achieve better performance results than some state-of-the-art approaches, although the architecture does present a larger memory usage and boot up time.

Working inside the context of microservices, Abranches et al. [28] proposes Shimmy, a different model for inter-containers communication. The authors advocate for the use of shared memory channels for passing information between services of the same application. They argument that this can lead to improvements in scenarios where information need to be processed by a chain of containers. The model created deploys agents together with the services and use a central controller to manage the creation of the shared memory channels between needed containers. In order to allow for applications to run scattered between more than one physical host, Shimmy uses RDMA technology to directly copy data from one server to another without involving the processor.

The authors of [29] focus on overlay networks that connect containers of the same application in multiple different physical hosts. They argue that the traditional overlay solutions lead to a large delay in CPU usage due to the need of making packets pass through 2 network stacks (in the sender and receiver) composed of the virtual interface, virtual switch and physical NIC. They propose Slim, a new solution that allows for connection based protocols, such as TCP, to perform at much higher levels in terms of latency and CPU usage. For this, they use a custom socket integrated with the application that together with a user-space router translate all connection calls to the correct IP used in the host network. The proposed solution also offers functionalities as QOS guarantees and rate limiting.

Hong et al. [30] proposes a model for deploying Virtual Network Functions (VNFs) in containers. VNFs consist of network functions such as Firewalls and Load Balancers that originally are deployed on specific hardware being used with virtualization techniques. The solution presented consist in using the BESS virtual interface [31] with DPDK [32], a development kit for accelerating packet processing, in order to increase the throughput and reduce the latency for packet processing. The tests are conducted, one where only the interface is tested, another using a Click VNF running in the proposed architecture and, finally, one where multiple VNFs are connect, creating a SFC. The authors were able to reach latency of less than 40 microseconds.

In [33], the authors focus on using VNFs and SFCs in the microservices architecture. The idea is to split the functions in the smallest units possible, as microservices, to avoid the need of the same functionality running in multiple different VNFs and allow for a more specific resource allocation for each component. The paper presents academic and enterprise attempts to perform these architectural changes.

C. Network QOS Guarantees with Containers

The authors of [34] focus on creating a system for allowing network QOS in communications between Kubernetes pods. The idea is to extend the already existing system for controlling the usage of other resources, such as CPU and memory, that consists of defining in the manifesto of the application the requirements and maximum limits for usage. The system proposed does the same thing for bandwidth, separating pods according to their priority and guaranteeing the intervals defined using Linux Traffic Control utility tool (TC) [35] to create multiple queues. However, the solution only works for inter-pods communication between pods on different Kubernetes nodes. Communication between containers in the same pod is not taken into consideration as they share all resources (shared memory and network interface). Moreover, communication between pods on the same node is not inside the scope of the work since it is done through shared-memory.

Dusia et al. [36] proposed a method for guaranteeing QOS for applications running on top of containers using the Docker Engine. The method employed was to create different queues for packets and allow for 3 different levels (low, medium and high) of priority between containers. Linux Traffic Control (TC) [35] was used to create these queues for arriving packets and uses a SFQ algorithm for each queue. The authors also set a throttle for the rate of package sent by the containers.

VI. Conclusion

In this work we have presented a exploration of multiple scenarios for container contention for network resources in a single host. We detailed the differences between three available network interface virtualization solutions, namely Open vSwitch, Macvlan and Linux Bridge. In our experiments, we investigated the effects of multiple levels of contention on the total and individual throughput of containers and the Flow-Completion Time for uniform and elephant/mouse workloads.

Our experiments showed that OVS presents better results then Macvlan and Bridge, the later exhibiting a considerably lower performance then the other two in the studied scenario of network contention. This can be explained by the faster forwarding mechanism implemented in OVS, which allowed it to use less of the CPU during testing and, therefore, allow for faster communication in the scenarios observed.

In the future, we want to expand these scenarios to encompass multi-VM scenarios. We have seen in this work that containers running on the same host can have a high contention for network and CPU resources, which leads to lower performance. However, when containers are separated in different VMs, we need to take into account the costs for encapsulating and sending packets from one host to another. Therefore, by expanding this work for environments with multiple hosts, we wish to explore how much this migration can affect network performance. We also want to encompass container orchestration solutions such as Kubernetes [24] and Docker Swarm [37].

Moreover, another observation from this work is that, achieving fair division of network resources can be difficult in container environments, as some may have a much higher throughput then others on the same host. Moreover, services that send larger flows can use up the available bandwidth and reduce the throughput for other replicas with smaller flows. This can be a problem when we have critical instances that specifically require a certain amount of bandwidth. Therefore, we also want to study methods for allowing network operators to easily create priorities for a microservice system, without losing the already established base performance.

Acknowledgements

I would like to specially thank Tiago Ferreto for guiding and mentoring me in this work and Miguel Neves and Israat Haque from Dalhousie University for counselling me during this project. I would also like to thank my family, my girlfriend and my close friends for supporting me in this whole journey.

Proposals	Virtualization Technique	Interfaces	Evaluated Parameters	Experiment Scale
Zhao et al. [18]	Containers	Bridge, Macvlan, OVS	Different allocation scenarios	Single pair of containers
Anderson et al. [16]	Containers	Bridge, Macvlan, OVS, SR-IOV	Size of VNF chain	Single chain of containers
Mentz et al. [17]	Containers	Bridge, Host, Macvlan, Overlay	Different allocations, workloads, congestion	Up to two pairs
Suo et al. [19]	Containers	Bridge, Container, Host and multiple Overlay solutions	Different packets size and network contention and CPU interference	Up to 8 pairs
Qi et al. [23]	Pods	CNI Plugins (Flannel, Kube-Router, Calico, Cilium, Weave)	Allocation and network contention	Up to 50 pairs in intra-host scenarios
Our Proposal	**Containers**	**Bridge, Macvlan, OVS**	**Network Contention and different flow sizes**	**Up to 150 pairs**

TABLE II: Comparison of our proposal and other papers exploring network performance of containers.

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] D. Gannon, R. Barga, and N. Sundaresan, "Cloud-native applications," *IEEE Cloud Computing*, vol. 4, no. 5, pp. 16–21, 2017.

[3] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[4] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[5] "Linux bridges," 2021. [Online]. Available: https://wiki.linuxfoundation.org/networking/bridge

[6] "Macvlan," 2021. [Online]. Available: https://docs.docker.com/network/macvlan/

[7] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vswitch," in *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, 2015, pp. 117–130.

[8] "Docker," 2021. [Online]. Available: https://www.docker.com/

[9] P. Mell, T. Grance *et al.*, "The nist definition of cloud computing," 2011.

[10] G. J. Popek and R. P. Goldberg, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM*, vol. 17, no. 7, pp. 412–421, 1974.

[11] N. Varis, "Anatomy of a linux bridge," in *Proceedings of Seminar on Network Protocols in Operating Systems*, vol. 58, 2012.

[12] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 19–33.

[13] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: practical and scalable ml-driven performance debugging in microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 135–151.

[14] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[15] "iperf," 2021. [Online]. Available: https://iperf.fr/

[16] J. Anderson, H. Hu, U. Agarwal, C. Lowery, H. Li, and A. Apon, "Performance considerations of network functions virtualization using containers," in *2016 International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2016, pp. 1–7.

[17] L. L. Mentz, W. J. Loch, and G. P. Koslovski, "Comparative experimental analysis of docker container networking drivers," in *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE, 2020, pp. 1–7.

[18] Y. Zhao, N. Xia, C. Tian, B. Li, Y. Tang, Y. Wang, G. Zhang, R. Li, and A. X. Liu, "Performance of container networking technologies," in *Proceedings of the Workshop on Hot Topics in Container Networking and Networked Systems*, 2017, pp. 1–6.

[19] 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.

[20] "Weave," 2021. [Online]. Available: https://github.com/weaveworks/weave

[21] "Flannel," 2021. [Online]. Available: https://github.com/coreos/flannel/

[22] "Calico," 2021. [Online]. Available: https://github.com/projectcalico/calico-containers

[23] S. Qi, S. G. Kulkarni, and K. Ramakrishnan, "Assessing container network interface plugins: Functionality, performance, and scalability," *IEEE Transactions on Network and Service Management*, 2020.

[24] "Kubernetes," 2021. [Online]. Available: https://kubernetes.io/

[25] "Kube-router," 2021. [Online]. Available: https://github.com/cloudnativelabs/kube-router

[26] "Kube-router," 2021. [Online]. Available: https://cilium.io/

[27] Z. Shen, Z. Sun, G.-E. Sela, E. Bagdasaryan, C. Delimitrou, R. Van Renesse, and H. Weatherspoon, "X-containers: Breaking down barriers to improve performance and isolation of cloud-native containers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 121–135.

[28] M. Abranches, S. Goodarzy, M. Nazari, S. Mishra, and E. Keller, "Shimmy: Shared memory channels for high performance inter-container communication," in *2nd {USENIX} Workshop on Hot Topics in Edge Computing (HotEdge 19)*, 2019.

[29] D. Zhuo, K. Zhang, Y. Zhu, H. H. Liu, M. Rockett, A. Krishnamurthy, and T. Anderson, "Slim:{OS} kernel support for a low-overhead container overlay network," in *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, 2019, pp. 331–344.

[30] D. Hong, J. Shin, S. Woo, and S. Moon, "Considerations on deploying high-performance container-based nfv," in *Proceedings of the 2nd Workshop on Cloud-Assisted Networking*, 2017, pp. 1–6.

[31] "Bess (berkeley extensible software switch)," 2021. [Online]. Available: https://github.com/NetSys/bess

[32] "Dpdk (data plane development kit)," 2021. [Online]. Available: https://www.dpdk.org/

[33] S. R. Chowdhury, M. A. Salahuddin, N. Limam, and R. Boutaba, "Rearchitecting nfv ecosystem with microservices: State of the art and research challenges," *IEEE Network*, vol. 33, no. 3, pp. 168–176, 2019.

[34] C. Xu, K. Rajamani, and W. Felter, "Nbwguard: Realizing network qos for kubernetes," in *Proceedings of the 19th International Middleware Conference Industry*, 2018, pp. 32–38.

[35] "Linux traffic control," 2021. [Online]. Available: http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html

[36] A. Dusia, Y. Yang, and M. Taufer, “Network quality of service in docker containers,” in *2015 IEEE International Conference on Cluster Computing*. IEEE, 2015, pp. 527–528.

[37] “Docker swarm,” 2021. [Online]. Available: https://docs.docker.com/engine/swarm/