2019 IEEE European Symposium on Security and Privacy Workshops

SeCoNetBench: A modular framework for Secure Container Networking Benchmarks

Amr Osman
Chair of privacy and security
TU Dresden
Email: amr.osman@tu-dresden.de

Simon Hanisch
Chair of privacy and security
TU Dresden
Email: simon.hanisch@tu-dresden.de

Thorsten Strufe
Chair of privacy and security
TU Dresden
Email: thorsten.strufe@tu-dresden.de

***Abstract*—Container security, especially in the quest for controlled access and secured communication between containers, has spawned a multitude of implementations, based on various concepts and design choices. They are characterized by very different performance properties, which so far have not comprehensively been benchmarked nor compared in a fair manner. The emerging paradigm of moving execution to edge clouds requires both: efficiency in the light of ephemeral containers and mobility, and security in the face of resource sharing between various tenants at hosts of various providers.**

In this paper we introduce SeCoNetBench, a modular benchmarking platform for container network security, and compare the most prominent frameworks for access control and network isolation in the container ecosystem. The results demonstrate that trade-offs have to be made during infrastructure deployment, and we provide guidelines for designing high-performance secure container networking platforms in adversarial settings.

***Index Terms*—Container networking, Network micro-segmentation, Microservices, VPN, Secure network tunnels, Packet filters**

I. Introduction

Conventional services where running atop bare-metal, physical hardware on premise or in co-location data-centers. They motivated perimeter defense, relying on firewalling internal network segments from the external Internet, from where classic adversaries were assumed to attack.

The advent of virtualization, with elastic cost-models for XaaS (Anything-as-a-service), has changed this scenario disruptively, as various tenants are now sharing the same physical hardware. The agile edge cloud scenario complicates this trust model further, as partial services may now run in hardware of entirely different providers, chosen by their proximity to the end-user.

Traditional VM-based virtualization has proven to be too inefficient and slow for agile edge clouds[1], [2], [3]. We hence observe a dramatic surge towards microservice virtualization, decomposing monolithic software into isolated components that run in their own compartments. Microservice virtualization is characterized by finer-graind isolation. By leveraging kernel name spaces, full OS virtualization is no longer necessary, and containers become much more lightweight, hence faster to bootstrap[4].

Considering mobile edge clouds, recent studies highlight the demand for very fast migration and cloning of microservices, to achieve QoS guarantees and minimal delay penalties in face of roaming end users [5], [6].

Implementing filtering, to achieve controlled access to specific services, and isolation, to achieve confidentiality and integrity of the data exchanged between microservices becomes highly challenging in these scenarios, and traditional perimeter defense becomes obsolete.

Early approaches to protect traffic between virtual machines commonly assume infrequent topology changes, and are designed thinking on the level of hosts, rather than the level of specific services of various tenants [7], [8]. Several efforts have been made to draft solutions for high-fidelity network access control between containers, recently [9]. They follow various design choices with respect to container connectivity, confidentiality of the network traffic, and consequently the filtering and isolation of network flows.

In this paper we are hence set out to discuss security properties, and to evaluate the performance of different secure container networking solutions. We dissect the underlying mechanisms and primitives that are embodied as different types of packet filters, secure networking protocols, and Linux interfaces.

We reason about both: their individual, as well as their cumulative impact on network performance for agile edge clouds, subject to various synthetic and realistic workloads.

This paper hence makes the following contributions:

1) We design the modular benchmarking framework SeCoNetBench to assess the performance of secure container networking solutions.
2) We implement SeCoNetBench on top of Kubernetes, the state of the art container orchestration platform today.
3) We perform a comprehensive performance analysis of the secure container networking design space with respect to the aforementioned underlying mechanisms, using synthetic and realistic microservice workloads.

The rest of this paper is structured as follows. In Section II, we give a structured overview of the related work. Section III describes our system and adversary model. Section IV lays down the architecture for our benchmarking framework. We later in Section V evaluate our framework with multiple workloads and present the results. We finalize with a discussion of

DOI 10.1109/EuroSPW.2019.00009

our results and then summarize our conclusions and existing open problems in Section VI.

II. Related Work

This section divides the related work into: container networking, packet filters, and secure container networking.

A. Container networking

Some of the underlying network primitives used to connect between containers such as soft switches or Linux bridges have been benchmarked in isolation. Yet, no comparison nor investigative study about the bottlenecks and trade-offs was made. Claassen et. al.[10] evaluated veth, Macvlan and IPvlan in two different settings: locally and remotely. IPvlan yielded the best local TCP throughput, whereas Macvlan had the best local UDP throughput. For remote communication, veth sustained a better TCP throughput, and Macvlan had a better UDP throughput. The authors recommended using Macvlan for both single- and multiple- node deployments as it performs well on average for both TCP and UDP flows. The authors confirm our hypothesis that further evaluation is necessary to measure how these interfaces perform in different container overlay networks.

Similar experiments have also been conducted under different settings and measured more technologies such as Open vSwitch (OVS), and SR-IOV. Veth still outperformed OVS and Linux bridges, and a 50% performance loss was also observed when containers were run on VMs[11]. Though deploying process ensembles in the form of nested containers[12] might yield certain performance benefits at the cost of less isolation. Moreover, Macvlan was concluded to have lower network latency and jitter compared to the aforementioned interfaces and Linux bridges[13].

Docker overlay networks were also benchmarked. Arne Zismer[14] analyzed the TCP and UDP throughput of three popular libraries: libnetwork, Flannel and Weave. Overall, Weave followed by Flannel performed better compared to Libnetwork, and the CPU was shown to be a limiting factor to the achievable throughput. In alignment with our hypothesis, future work regarding the impact of encryption on the overlays was suggested.

B. Packet filters

The concept of dynamic packet filters[15] and dynamic firewalls[16] have long existed under different terminologies but similar semantics. They both address the problem of changing or optimizing the packet-matching rules and policies at runtime without the need to restart the firewall, and optimally without disrupting existing connections. This however requires some knowledge about the ongoing traffic flows.

Chandramouli[17] provided a preliminary report on the different technologies which could be used to: isolate virtual networks of VMs from one another, and to provide network ACLs between them. However, no evaluation to support the concluded set of recommendations was made.

Bruijn[18] compared the performance of two packet filters: iptables and extended Berkley packet filter (eBPF). It was concluded that the filtering rule complexity had negligible impact on the performance of eBPF, which was the inverse observation to iptables' performance. Furthermore, eBPF performed better when the communicating containers were on the same host, and iptables was best during cross-host communication.

C. Secure container networking

Nane Kratzke[19] studied the impact of: containerization, software-defined overlays and encryption on network throughput. Docker was selected as a containerization solution, and Weave as an overlay with support for encryption. The impact on HTTP throughput was then evaluated on a single CPU-core system. Docker containerization impacted the performance by 10-20%. Weave further lowered the performance by 30% and up to 70% for larger message sizes of more than 200kB. Weave encryption was shown to have minor impact on performance and up to 10% lower throughput for large message sizes. It was alleged that Weave had such a huge performance impact as its router daemon was bottlenecked by the single CPU running the experiments. But, a validation of this claim is left for future work.

Other secure protocols have also been subjected to basic micro-benchmarks[20]. It was shown that the Wireguard protocol outperforms IPsec, and that OpenVPN came with the most significant overhead on ping time and network throughput.

D. Summary

To this end, none of the above approaches tackle the problem at hand in its wholeness. The performance and the security achieved when providing dynamic network-wide ACLs between containers is a product of the underlying bridge mechanisms, overlay technologies and the selected cryptographic protocols as well as the ACL enforcement technique itself. Additionally, the past benchmarks did not consider realistic, and various types of container workloads. We therefore proceed by shedding more light onto our fundamental assumptions and design decisions for SeCoNetBench.

III. SeCoNetBench

Communication between containers could take multiple forms. It could happen between containers on the same host machine as well as across multiple hosts and edge clouds. Therefore regardless of where containers are scheduled, the communication channels between them should be transparent to the underlying topology. This section highlights our assumptions and sheds light on the adversary model, communication model as well as the design space of technologies that could be leveraged to counter it.

A. Communication model

We consider a microservice deployment within a single network administrative domain such as an edge cloud offered by a single provider. This deployment consists of different services which run inside Linux containers, and are spread

across multiple hosts in the network according to a scheduling strategy by a logically-centralized-but-physically-decentralized container orchestrator. Each container has its own unique MAC and IP addresses, which are used to uniquely identify it in the network. These containers then communicate using TCP/IP in order to exchange data bidirectionally. This setting is aligned with major container orchestration platforms such as Kubernetes, Mesos and Docker, and offers more clarity, and scalability compared to other network addressing mechanisms which could be used such as port mappings.

B. Adversary model

Given the aforementioned deployment, we assume an active internal attacker who controls one or more containers, and seeks to have unauthorized network access to other containers in the same network, or are co-located on the same host. It is also capable of scanning the network to collect reconnaissance information. However, we subsume that it cannot break out of the confines of its container isolation which is provided by the Linux kernel through side channels or other local exploits.

In addition, we assume the presence of an external active attacker which is capable of manipulating, dropping, or crafting network traffic between containers that traverses multiple hosts (Dolev-Yao adversary). It is capable of inspecting the content of the traffic within its reach, and aims to have unauthorized access to containers, and the data exchanged between them. However, it cannot decrypt encrypted traffic assuming the choice of a secure cipher suite, and that the respective cryptographic keys are not compromised.

C. System model

Figure 1 highlights multiple variables in the microservice deployment under test which must be considered by SeCoNetBench.

1) Single-host networking: In order to protect the network against the internal attacker, strict fire-walling and packet filtering must be enforced between the communicating co-hosted containers. Optimally, all traffic must be blocked and a whitelisting approach should only allow specific pairs of containers to communicate as necessary. Therefore, two design decisions lie in question. First, the choice of network interfaces or bridges to allow containers to communicate (3). Second, the choice of the packet filter (1).

2) Multi-host networking: As containers communicate across multiple hosts, one must secure communication against both: the internal and the external attackers. This implies that the design space expands further to include the choice of routing protocols or network tunnels to realize transparent connectivity (2a), in addition to the choice of encryption for the dataplane (e.g. through a secure VPN) to ensure the confidentiality and integrity of the data (2b).

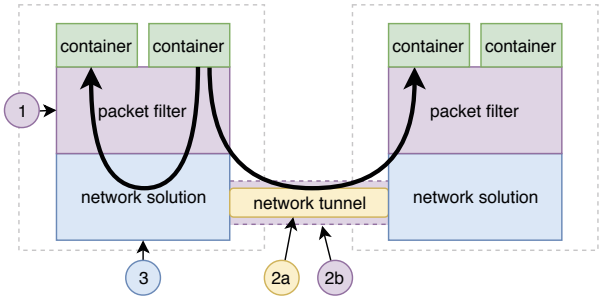


Figure 1. Secure container networking

IV. Framework and Methodology

Motivated by the models and constraints in the above section, we describe SeCoNetBench, which is a benchmarking framework and a holistic testbed to deploy container workloads and instrument the configuration of their underlying secure network communication building blocks under test.

A. Architecture

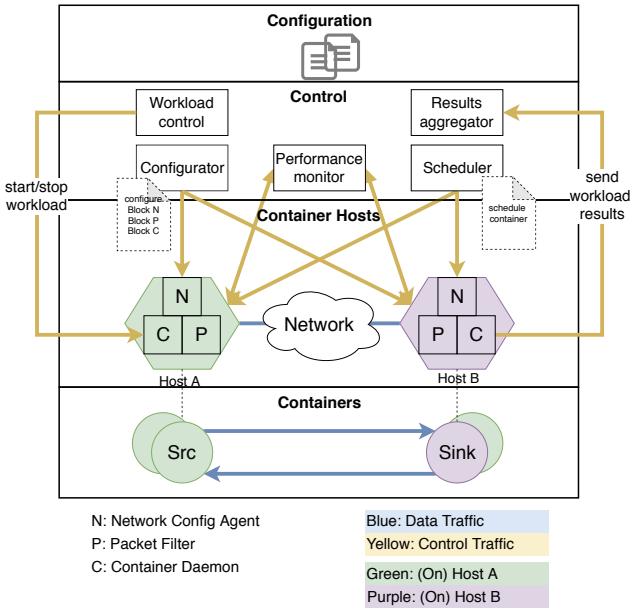


Figure 2. The SeCoNetBench architecture

Our architecture (see Figure 2) consists of four distinct layers. The first is the configuration layer that describes which building blocks to configure, the number of containers, the workloads to run, and the packet filter rules. The Control layer is responsible for configuring, monitoring and executing the workloads. The Configurator applies the network configuration by communicating with the Network Config Agent, which is located on the container hosts. The Scheduler performs the scheduling of the containers on the hosts and the Configurator conjunctively creates the packet filter rules for each container. An additional component of the Control layer is the

performance monitor which collects the performance metrics of the container hosts such as CPU, Memory and Network I/O utilization. The remaining components of this layer are the workload control and the result aggregator, who are responsible for starting the workloads on the containers and receiving the workload results. Layer three represents the container hosts and the intermediate network topology connecting them. The Containers run on top of the hosts in their own logical network that is abstracted from the physical network on which the hosts are connected. This can be seen in layer four of our software stack.

B. Benchmarking workflow

The benchmark workflow (see Figure 3) starts with the Configurator applying the configuration to the container hosts. In this step, the network solutions, packet filters, and network tunnels are set up. The Scheduler then spawns the required containers on the hosts. After the containers are running the workload control instructs the containers to run the first workload. The workload control waits for all containers to conclude and transmit their workload results to the results aggregator before starting the next workload. This process is repeated until all workloads have been executed. Then, the Scheduler checks if there are more container and packet filter rule configurations to run. When the Scheduler is finished, the Configurator configures the next setup, until all of them are complete.

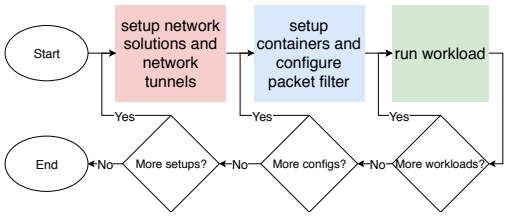


Figure 3. Benchmarking workflow

C. Single-host networking

Containers on the same host are usually connected via virtual interfaces (veth) which act as a data pipe, transporting packets from one endpoint to another. The container gets one endpoint and the other one is connected to the network solution. Network solutions are responsible for forwarding packets between multiple endpoints. We select bridge, Open vSwitch (OVS), Berkley packet filter (BPF) and routing as candidate network solutions. We additionally identify three popular concepts for packet filtering: iptables, Open vSwitch, and Berkley packet filter.

iptables[21] works on a rule-set that is organized as tables, which organize the rules in data structures called chains. Each table has multiple entry points that correspond to specific stages in the Linux network data flow. The entry points lead to specific chains that are executed when a packet enters through them. Inside a chain, the rules are evaluated in sequential order. Rules consist of a predicate and an action. When a packet matches the predicate the action of the rule is executed. Actions can be to jump to another chain, to accept, drop, or to modify the packet. If a packet reaches the end of a chain the default policy of the chain, which is either accept or deny, is executed.

Open vSwitch[22] uses OpenFlow rules to determine the next hop of a packet or filter it. The classification is realized by having a set of hash tables which match packets by a specific hash tuple, e.g., the source address plus the port number. A table stores all rules that match on the same hash tuple. Packets traverse all hash tables until a match is found. Due to how lengthy this approach is, OVS only classifies the first packet of a flow and then caches the decision in a hash table that matches consecutive packets to their flow.

Berkley packet filter[23] uses a directed acyclic flow graph (DAG) as a filter model. Each node of the graph is a match on a part of the packet which determines the next node in the graph. The evaluation is concluded when a leave node is reached. The benefit of this model is that the decision of each check is implicitly stored by the position of the packet in the graph, making redundant checks of the same protocol field unnecessary. The DAG is compiled to byte code by translating each node to a comparison plus a conditional jump to the next node. This code is then run in its special-purpose VM within the Linux kernel.

We expect that OVS will perform best as found in [24]. We also expect to see a spike in latency for the first packet forwarded by OVS due to its classification process. The results from [18] and [25] suggest that BPF will perform better than iptables when it comes to large rule sets on on the same host.

D. Multi-host networking

Multi-host networking expends on top of the single-host networking and adds the connections between multiple container hosts. We consider network tunnels and VPNs to separate the container network from the host topology.

Network tunnels encapsulate network packets inside a tunneling protocol. The tunnel decouples the encapsulated traffic from the underlying network. This allows bridging networks that do not support certain types of connectivity or protocols. A Further benefit is that operators can establish logical networks that hide the underlying topology to its endpoints. Del Picolo et. al. [26] gave an overview of the network tunnels which could be used for data center networks.

We pick IP-in-IP (IPIP), Generic Routing Encapsulation (GRE), and Virtual eXtensible Local Area Network (VXLAN) for the network tunnels. IPIP encapsulates IP packets by adding an additional IP header with adds 20 bytes. GRE has a variable size header that ranges from 4 to 16 bytes overhead and is inserted between the IP header and the payload. VXLAN transports Ethernet frames by encapsulating them with UDP (8 bytes) and its own header of 8 bytes. Because of the outer IP header (20 bytes) and Ethernet header (14 bytes), its total overhead is 50 bytes.

Virtual private networks (VPNs) are network tunnels that use cryptographic measures to ensure confidentiality and in-

tegrity of its encapsulated traffic. The confidentiality is preserved by encrypting the traffic and integrity by adding a Message authentication code (MAC) to the packets.

We pick IPsec, Wireguard (wg), OpenVPN (OVPN), and Tinc as suitable VPNs. IPsec uses the Encapsulating Security Payload (ESP) protocol for encapsulation. ESP adds a header (8 bytes) behind the IP header and a trailer (18 - 33 bytes) on top of the payload. Wireguard, OVPN, and Tinc all use UDP (8 bytes) as the outer carry-protocol and add their own header and trailer to the payload. Wireguard adds a total of 40 to 55 bytes, OVPN adds 48 to 63 bytes, and Tinc adds 28 to 43 bytes of overhead, depending on the length of the plaintext padding.

We expect the VPNs to perform worse than the network tunnels because they require additional time to process each packet and due to their header overhead. The results of Donnerfeld et. al. [20] suggest that IPsec and Wireguard will perform similar to each other while OpenVPN will be significantly slower.

V. Evaluation

This section discusses our evaluation. First, we describe our setup and testbed specifications. Then, we explain our workloads and lay down our results for single- and multi-host networking respectively.

A. Setup

We implemented SeCoNetBench using a cluster of virtual machines (VMs) as container hosts. Each VM had four cores and 10 GB of RAM running Debian 8 as OS with Kernel 4.9. We used a single hypervisor with 40 Intel Xeon E5-2630 v4 CPUs (2.2 GHz) and 500 GB of RAM. Kubernetes (1.7.5) orchestrates the Docker (18.06.1-ce) containers and allows the deployment of network solutions and packet filters via CNI plugins. We choose Canal (Calico v2.6.9 & Flannel v0.1), Cilium (v1.2.0-rc1), Weave (v2.3.0), and Contiv (v1.2.1) as plugins to have iptables, BPF, and OVS as packet filters. We choose OpenVPN (2.3.4), Tinc (1.0.24), Strongswan (5.2.1), Wireguard (0.0.20181018) as VPNs and iproute2 (4.9.0) to configure network tunnels. In order to protect the integrity and confidentiality of the network flows, we use AES-256-GCM-128 (IPsec & Tinc), AES-256-CBC-HMAC-SHA1 (OpenVPN), and ChaCha20-Poly1305 (Wireguard) as cipher suites. We generate our workloads using iputils-ping (s20161105), Uperf (1.0-dev), and D-ITG (2.8.1). Dstat (0.7.2-4) was also used to measure the performance of the VMs

B. Workloads

We used six different workloads to evaluate our benchmark and the technologies under test:

1) *Latency Basic*: A simple ping (ICMP) is used to measure the round-trip time between the two containers.
2) *Throughput Basic*: Using Uperf we saturate the link to measure the effective throughput between the containers. We used 64, 653, and 1370 bytes of payload to study the effect of the payload size on the throughput.
3) *Bursty Traffic*: We use D-ITG to emit small 1-second bursts of UDP traffic. Each burst had a throughput of 500 Kbps.
4) *100-TCP-Connections*: Here, D-ITG creates 100 simultaneous TCP connections between the container pairs with a throughput of about 500 Kbps per connection.
5) *Quake 3*: The video game Quake 3 requires low latency UDP traffic for its players to have an enjoyable experience. Lang et. al. [27] investigated the traffic characteristics of Quake 3 and reported random distributions that can be used to synthesize the traffic. We use D-ITG to generate 4 flows of Quake 3 traffic per container pair, simulating a four-player game.
6) *Video Streaming*: Video on demand platforms like YouTube and Netflix make up a large share of today's Internet traffic [28]. In order to synthesize video streaming traffic, we rely on the data reported in [29]. We draw 10 videos with various lengths and sizes from the distributions to create corresponding TCP streams with D-ITG. We calculate the throughput required to watch the videos in real-time with a payload size of 1370 bytes. Since the reported video data is from 2009 we scale the flows to transport 3 GB per hour on average, which is the estimate of Netflix for HD video streams.

Each workload is used to send network flows from one source to one corresponding sink container. This allows scaling the workloads by scheduling more sink-source pairs independent from the rest of the network. We ran our workloads with 2, 8, and 32 total containers. Each of the workloads was run multiple times until the result values converged. In order to avoid packet fragmentation, e.g., in the multi-host networking scenario, we selected 1370 bytes as the maximum payload size. The default payload size used was 653 bytes, unless stated otherwise.

The first two workloads are micro benchmarks designed to find the minimal latency and the maximum throughput of the system. The respective following workloads are synthetic and were designed to stress the packet filter and secure network tunnels by exhibiting unusual traffic patterns or a high number of connections. The last two workloads are derived from realistic traffic scenarios by using random distributions.

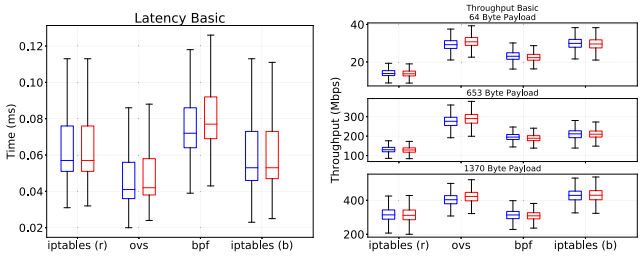
C. Single-host networking

We ran the aforementioned workloads to measure the network performance of containers communicating on the same host. We hereby analyze the impact of secure packet filtering on the overall performance. We inserted filtering rules that enforce network isolation between only the communicating container pairs and follow a whitelisting approach (default deny). Our results are shown below for a total of 32 containers.

1) Basic workload: In Figure 4a, the latency for the *Latency Basic* workload is shown. The red box plots depict the results with packet filtering and the blue box plots depict the results without any filtering. Also, we distinguish between iptables under two different settings: the first (iptables (r)) is when the traffic between the containers is routed by means of direct

layer-3 routes in the Linux routing table, and the second setting (iptables (b)) is when the containers are communicating directly using a classic Linux bridge. We can observe that packet filtering has a negligible impact on the latency. We observed that the first packet of each flow exhibits a latency that is magnitudes higher (up to 30 ms) than the following ones in OVS. We can also observe that BPF has relatively a higher latency than iptables and OVS.

For the *Throughput Basic* workload (Figure 4b), we observe that all network solutions appear to scale proportional to the payload size and that the packet filtering has a negligible impact on the throughput. We also find that BPF on average (653 payload size) has significantly less throughput (100 Mbps less) than iptables and OVS.



(a) Latency basic (b) Throughput basic

Figure 4. Singe-host networking - basic workload

2) Synthetic workloads: We find different results for the *Bursty Traffic* (fig. 5), where the packet filters increase the latency. During the *100-TCP-Connections* workload, we also observe that the latency slightly increases for iptables and OVS.

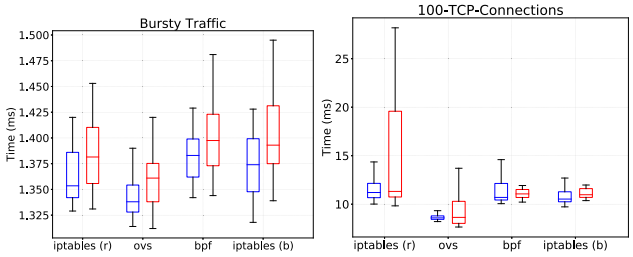


Figure 5. Single-host networking - Synthetic workloads

3) Realistic workloads: A significant increase in case of packet filtering is moreover observed in the *Quake 3* workload (fig. 6) for iptables and BPF. *Video Streaming* (fig. 6) exhibits a large latency variance for BPF and iptables, while OVS is unaffected. The relation between OVS, iptables, and BPF also correlate with the basic workloads.

We did not observe a significant impact of packet filtering on the throughput during both: the Synthetic and the realistic workloads. Therefore, we did not include the results here. However, one can realize that the Quake 3 benchmark closely shows similar latency result values compared to the *Latency Basic* benchmark. The result of the synthetic and realistic workloads exhibit an order of magnitude higher latency values compared to the *Latency Basic* workload.

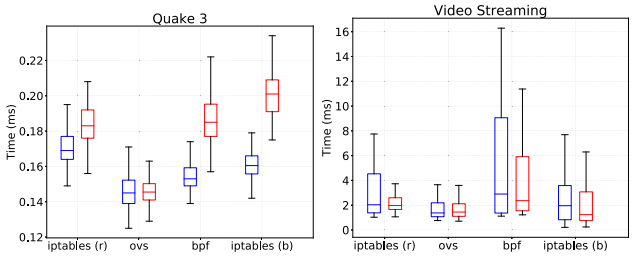
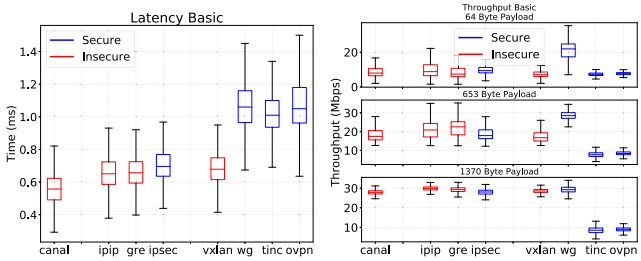


Figure 6. Single-host networking - Realistic workloads

D. Multi-host networking

We repeated the same workloads to test the impact of secure network tunneling on the network performance of container pairs that communicate across multiple hosts. Below, we show our results for a total of 32 containers. We distinguish between multi-host networking use routing (Canal), using layer-3 tunnels (IPIP, GRE, IPsec), and layer-4 tunnels (VXLAN, Wireguard, Tinc).

1) Basic workload: The *Latency Basic* workload is shown in Figure 7a. All tunnels add additional latency to the traffic. IPIP, GRE, VXLAN, and IPsec add around 20% more overhead, while Wireguard, Tinc, and OpenVPN nearly double the latency. This aligns with smaller preliminary studies [30], [31]. In the results of the *Throughput Basic* workload, we find that Wireguard shows the largest increase in throughput when compared to the other tunnels for 64 and 653 bytes of payload. We hypothesize that this is because Wireguard uses a bigger packet buffer for packet processing[20]. Hence, it favors throughput at the cost of per-packet latency. Most of the tunnels are able to saturate the links at 1370 bytes of payload and show the same throughput. However, Tinc and OpenVPN have the smallest throughput implying the most overhead due to encryption in user space as opposed to kernel space, in which the other secure tunnels apply encryption.



(a) Latency basic (b) Throughput basic

Figure 7. Multi-host networking - Basic workload

2) Synthetic workload: Wireguard again shows the highest effective throughput for the *Bursty Traffic* workload, seen in

Figure 8 due to its buffering mechanisms. The rest of the technologies fall relatively close to the bursty behavior, which is inherent to the workload itself. The *100-TCP-Connections* workload crashed often and its results were inconclusive. Therefore, they were not shown here. We leave a thorough investigation of the root cause for future work.

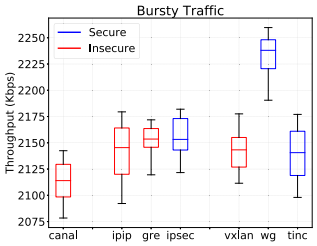


Figure 8. Multi-host networking - Synthetic workloads

3) Realistic workload: In the Quake 3 workload (fig. 9), all network tunnels performed close to each other, only Wireguard showed a slightly higher throughput. Thus, the impact of encryption was not significant. During the *Video Streaming* workload (fig. 9), IPIP and GRE performed the best with close to 30 Mbps throughput, which is close to what was observed in the *Throughput Basic* Workload. IPsec shows more than 8 Mbps less throughput and Tinc shows the least throughput for this workload. Hence, they are not suitable for video streams which entail large volumes of traffic under variable network bit rates.

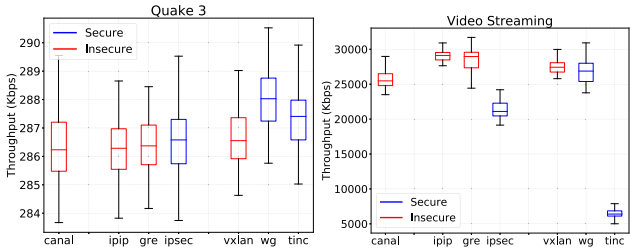


Figure 9. Multi-host networking - Realistic workloads

While we also benchmarked the packet filters we found that their effect on the results is negligibly small compared to the secure tunnels. Thus, we excluded them from the graphs above. Moreover, we exclude the latency measurements for both: the synthetic and the realistic workloads as the relative behavioral difference between the technologies under test was directly correlated with the baselines in the *Latency Basic* workload.

E. Discussion

The impact of network throughput and latency on real cloud-based applications have long been discussed, and it has been shown that the effective application performance is directly correlated[32]. Thus, platforms like SeCoNetBench are highly important to characterize the impact of the design decisions made when connecting containers across edge clouds on real-world performance.

Our current study still has some limitations, and this paper reports on work in progress. The measurements were performed in a virtualized environment. This biases the results, albeit in a fair manner for all compared technologies.

In future undertakings, we will perform the measurements on distributed, bare metal machines, to avoid virtualization artifacts within the benchmarking platform itself. We are also planning to extend the number of scenarios and parameter ranges. Increasing the number of connections and access control policies as well as the traffic, we will bring all components into heavy regimes. We will then increase the detail of our measurements, thus being able to report more fine-grained results with respect to the influence of CPU load, IO, memory, and network saturation.

On a more general level it will be interesting to identify the root cause of some of the performance differences: IPsec clearly benefits from hardware acceleration of AES, Tinc and OpenVPN clearly suffer from user-space rather than kernel-space encryption. However, the reasons for highly deviating throughput penalties of different technologies that wrap iptables, for example, currently remain largely unclear.

VI. Conclusion

In this paper, we surveyed the field of container networking security solutions, with a focus on isolation of both containers within a single host, as well as the network flows of their communication across hosts. We analyze the design space, as embodied in the latest applied technologies for inter-container networking, filtering, network tunnels, and VPNs.

We then designed SeCoNetBench, a modular benchmarking framework to measure their impact on network performance in edge cloud deployments. Using synthetic and realistic container workloads it can assess the performance impact in both single-host and multi-host container networking scenarios. It thus provides insight into the overhead of both filtering, as well as networking and tunneling technologies.

We instantiated SeCoNetBench on Kubernetes, and performed a benchmark of the frameworks that are currently discussed for container virtualization. With respect to the overhead, we confirm our expectations that (a) packet filters have a larger impact on the delay than on the throughput in single-host settings, and (b) that network isolation by tunnels or VPNs had a more pronounced impact on the throughput than filtering, in multi-host scenarios.

The results do not identify a single best solution for all settings. However, considering typical edge cloud setups with large numbers of connections, Berkeley Packet Filters (BPF) and Open vSwitch (OVS) outperformed iptables in filtering tasks. The results of tunneling were very encouraging: deploying both IPsec and WireGuard entailed only very limited overhead. In both cases the benchmarks achieved throughput similar to the insecure tunneling counterparts VXLAN and IPIP, and security in this case hence comes at almost negligible cost. WireGuard, however, exhibits increased latency penalties,

which we attribute to its internal buffering and queuing that is implemented to increase throughput.

ACKNOWLEGMENTS

This work in parts has been funded through the Cluster of Excellence EXC 2050 “CeTI” and BMBF FASTcloud.

REFERENCES

[1] M. Eder, “Hypervisor-vs. container-based virtualization,” *Future Internet (FI) and Innovative Internet Technologies and Mobile Communications (IITM)*, vol. 1, 2016.

[2] C. Pahl and B. Lee, “Containers and clusters for edge cloud architectures–a technology review,” in *FiCloud*. IEEE, 2015, pp. 379–386.

[3] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, “An updated performance comparison of virtual machines and linux containers,” in *ISPASS*. IEEE, 2015, pp. 171–172.

[4] A. Machen, S. Wang, K. K. Leung, B. Ko, and T. Salonidis, “Live service migration in mobile edge clouds,” *CoRR*, vol. abs/1706.04118, 2017. [Online]. Available: http://arxiv.org/abs/1706.04118

[5] L. M. Vaquero and L. Rodero-Merino, “Finding your way in the fog: Towards a comprehensive definition of fog computing,” *ACM CCR*, vol. 44, no. 5, pp. 27–32, 2014.

[6] H. Chang, A. Hari, S. Mukherjee, and T. Lakshman, “Bringing the cloud to the edge,” in *INFOCOM WKSHPS*. IEEE, 2014, pp. 346–351.

[7] F. Callegati and W. Cerroni, “Live migration of virtualized edge networks: Analytical modeling and performance evaluation,” in *SDN4FNS*. IEEE, 2013, pp. 1–6.

[8] A. Manzalini, R. Minerva, F. Callegati, W. Cerroni, and A. Campi, “Clouds of virtual machines in edge networks,” *Communication Magazine*, vol. 51, no. 7, pp. 63–70, 2013.

[9] “A view of fog computing from networking perspective,” *CoRR*, vol. abs/1602.01509, 2016, withdrawn. [Online]. Available: http://arxiv.org/abs/1602.01509

[10] J. Claassen, R. Koning, and P. Grosso, “Linux containers networking: Performance and scalability of kernel modules,” in *NOMS*. IEEE, 2016, pp. 713–717.

[11] 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*, ser. HotConNet ’17. New York, NY, USA: ACM, 2017, pp. 1–6.

[12] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder, “Performance evaluation of microservices architectures using containers,” in *NCA*. IEEE, 2015, pp. 27–34.

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

[14] A. Zismer, “Performance of docker overlay networks,” 2016.

[15] Z. Wu, M. Xie, and H. Wang, “Swift: A Fast Dynamic Packet Filter.” in *NSDI*, vol. 8, 2008, pp. 279–292.

[16] Q. Duan and E. Al-Shaer, “Traffic-aware dynamic firewall policy management: techniques and applications,” *IEEE Communication Magazine*, vol. 51, no. 7, pp. 73–79, 2013.

[17] R. Chandramouli and R. Chandramouli, “Secure virtual network configuration for virtual machine (vm) protection,” *NIST Special Publication*, vol. 800, p. 125B, 2016.

[18] N. de Bruijn, “eBPF Based Networking,” 2017.

[19] N. Kratzke, “About microservices, containers and their underestimated impact on network performance,” *CoRR*, vol. abs/1710.04049, 2017. [Online]. Available: http://arxiv.org/abs/1710.04049

[20] J. A. Donenfeld, “WireGuard: Next generation kernel network tunnel,” in *Proceedings 2017 Network and Distributed System Security Symposium*. Internet Society, 2017.

[21] N. team, “IPtables,” 1998. [Online]. Available: https://www.netfilter.org/projects/iptables/index.html

[22] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, A. Networks, and M. M. Casado, “The Design and Implementation of Open vSwitch,” *NSDI*, pp. 117–130, 2015. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/pfaff

[23] S. McCanne and V. Jacobson, “The BSD Packet Filter: A New Architecture for User-level Packet Capture.” in *USENIX winter*, vol. 93, 1993.

[24] P. Emmerich, D. Raumer, F. Wohlfart, and G. Carle, “Performance characteristics of virtual switching,” in *IEEE CloudNet*, 2014, pp. 120–125.

[25] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, “Performance implications of packet filtering with linux eBPF,” in *2018 30th International Teletraffic Congress (ITC 30)*. IEEE, sep 2018.

[26] V. D. Piccolo, A. Amamou, K. Haddadou, and G. Pujolle, “A survey of network isolation solutions for multi-tenant data centers,” *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2787–2821, 2016.

[27] T. Lang, P. Branch, and G. Armitage, “A synthetic traffic model for Quake3,” *ACM ACE ’04*, no. cycle 1, pp. 233–238, 2004. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1067343.1067373

[28] Sandvine, “Global Internet Phenomena Report 2016 North and Latin America,” 2017. [Online]. Available: https://www.sandvine.com/hubfs/downloads/archive/2016-global-internet-phenomena-report-latin-america-and-north-america.pdf

[29] A. Abhari and M. Soraya, “Workload generation for YouTube,” *Multimed. Tools Appl.*, vol. 46, no. 1, pp. 91–118, 2010.

[30] A. Buzachis, A. Galletta, and A. Celesti, “Towards Osmotic Computing: Analyzing Overlay Network Solutions to Optimize the Deployment of Container-Based Microservices in Fog, Edge and IoT Environments,” p. 10.

[31] R. Bankston and J. Guo, “Performance of container network technologies in cloud environments,” in *Electro/Information Technology (EIT)*. IEEE, may 2018.

[32] D. A. Popescu, N. Zilberman, and A. W. Moore, “Characterizing the impact of network latency on cloud-based applications’ performance,” p. 20.