Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2020) 28: 3368 – 3388
© TÜBİTAK
doi:10.3906/elk-2002-68

A new smart networking architecture for container network functions

Gülsüm ATICI[1,*], **Pınar BÖLÜK**[1,2]
[1]Department of Computer Engineering, Faculty of Engineering and Natural Sciences, Bahçeşehir University, İstanbul, Turkey
[2]Department of Software Engineering, Faculty of Engineering and Natural Sciences, Bahçeşehir University, İstanbul, Turkey

Received: 11.02.2020 • **Accepted/Published Online:** 10.09.2020 • **Final Version:** 30.11.2020

Abstract: 5G slices have challenging application demands from a wide variety of fields including high bandwidth, low latency and reliability. The requirements of the container network functions which are used in telecommunications are different from any other cloud native IT applications as they are used for data plane packet processing functions, together with control, signalling and media processing which have critical processing requirements. This study aims to discover high performing container networking solution by considering traffic loads and application types. The behaviour of several container cluster networking solutions – Flannel, Weave, Libnetwork, Open Virtual Networking for Open vSwitch and Calico – are explored with regard to the most commonly used container network functions in the form of MongoDB and web access. Evaluations show that none of the solutions provide a high throughput for all types of workload under optimum or heavy load situations. Hence, this research presents the view that traditional container networking implementation methods may not fulfill the container network functions' networking performance requirements. This is because container networking performance changes dynamically, depending on traffic load and application types. To overcome this problem, a new smart container networking architecture is proposed which allows containers to use several container networking solutions dynamically in conjunction with container monitoring tools. Eventually, the primary implementation of proposed architecture has been performed and evaluated. This research shows that proposed smart architecture delivers promising results compared to traditional implementation methods, in case appropriate decision is made during dynamic interface selection process.

Key words: CNF, container, performance, smart networking, traffic load, dynamic interface selection

1. Introduction

Network function virtualization (NVF) is of great importance in the development of 5G networks. NFV will eventually mature by overcoming automated deployment, virtual network functions (VNF) onboarding, scaling, configuration and updating [1]. Traditional VNFs (virtual machines) have some deficiencies including the excessive consumption of hardware resources in high traffic situations, and difficulty in operating such resources dynamically pursuant to traffic load by causing inadequate user experience or even service interruptions [2]. Building cloud-native VNFs (containers) which are referred to as container network functions (CNFs) overcome these limitations of traditional VNFs [3].

CNFs have APIs which facilitate automated scaling depending on dynamic traffic requirements, self-healing or fault tolerance, together with automated capacity, fault and performance management. They do

*Correspondence: gulsumatici@gmail.com

this with the help of container orchestrator platforms such as Docker Swarm, Kubernetes, Amazon EKS etc. [1]. Simplified management with the reduction of unnecessary allocated resources, reusability, together with the sharing of processes by reducing power consumption and hardware resources, are crucial benefits of CNFs in throttled situations [2]. Although providing several benefits, there are certain apprehensions with regard to using container technology in NFV, especially for network domains. The reasons why diversified 5G slices have challenging application demands with regard to a wide variety of fields are given in Figure 1; The CNFs requirements which are used in telecommunications are different from those relating to any cloud native IT applications.

CNFs in telecommunications are used for data plane packet processing functions, together with controlling, signalling and media processing with critical processing requirements such as less than 1 ms end-to-end over-the-air latency. CNFs' availability and coverage rise to 100 percent, and 99.99 percent respectively [4]. In addition, high traffic handling is expected from CNFs with a 1000 times larger throughput than that found in ordinary IT applications [4]. Due to the fact that any error or insufficiency impacts the number of subscribers using the network, containers networking performance becomes the biggest challenge facing the fulfillment of the 5G high performance targeted applications. In terms of 5G NFV implementation operational requirements with regard to every type of application and every level of traffic situation, a high performing and adaptive container networking architecture is required.

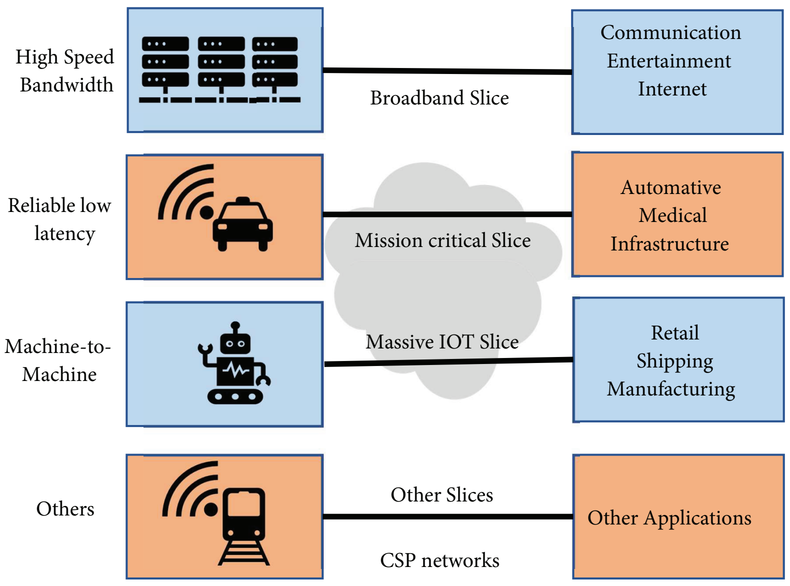


Figure 1. 5G Telecommunications applications challenging requirements.

Most studies have examined the performance of popular container cluster networking solutions in terms of different performance indicators such as throughput, latency, response times, bandwidth within different applications on local environments or public cloud infrastructures [2–5] However, few studies have focused on performance under load. The studies which have been carried out have generally been executed on high performance computing (HPC) platforms with applications which have different resource intensities [6–9]. Nevertheless, a few study have been actualized on public cloud environments within workloads which are similar to those found in common CNF applications [10].

Buzachis et al. investigated the performance of four network overlays – Open Virtual Networking for Open vSwitch (OVN) [11], Calico [12], Weave [13] and Flannel [14] – by using Kubernetes in a clustered environment designed for cloud, edge and fog [10]. They deployed distributed microservices based on FTP and constrained application protocol (CoAP) and executed the benchmarks by considering the transfer times of each microservice [10]. They compared the overhead of overlays according to container running in the host within five different configurations and increasing payloads. Their study presented that best solution is the OVN, with better time performances in all configurations, both for CoAP and FTP scenarios [10].

Ruan et al. researched the most suitable way to use containers in terms of different aspects, and performed a series of experiments to measure performance differences between application containers and system containers. They did this by observing the overheads of extra virtual machine layers between the bare metal and the containers [6]. They found that system containers are more suitable for sustaining I/O bound workloads than application containers. This was because the application containers suffered from high I/O latency because of the layered file system used [6].

Martin et al. investigated the interoperability of Rkt containers in high performance applications by running the HPL and Graph500 applications, then simulated the performance with commonly used container technologies such as Docker and LXC containers [7]. They analyzed the performance results of data intensive and compute intensive practical applications in several scenarios [7]. Casalicchio and Perciballi explored the measurement of the Docker performance from the perspective of the host operating system and the virtualization environment by outputting a characterization of the CPU and the disk I/O overhead generated by containers [8]. They studied this with two use cases – a CPU intensive workload and a disk I/O intensive workload – doing sequential or random reads and writes. However, they could not find a correlation between quotas and overheads, an aspect which requires further investigation [8].

Herbein et al. studied resource management problems in containerized HPC environments from three perspectives in the form of CPU allocation for CPU-intensive applications, disk I/O contention, and disk I/O load unbalance [9]. They considered network bandwidth throttling and prioritization by aiming to achieve better application performance and system utilization. They proposed a mechanism that enables bandwidth limits and privileged delivery service for critical applications in containers which control latency and delay, while providing a way to reduce data losses [9].

All the abovementioned former studies evaluated multihost container networking solutions in terms of specific application requirements. As far as we know, there have not been any studies that investigate traffic load effects for common CNF workloads on widely applicable container networking solutions. This study aims to determine the high performing container networking solutions in terms of traffic load, which is given in Figure 2. We do this by exploring the behaviour of several container cluster networking solutions over most commonly used CNF operations in the form of MongoDB and web access. Docker cluster networking solution's behaviours under high traffic conditions are simulated by using multithreading in order to imitate real world cases. Container cluster networking solutions such as Flannel, Weave, Libnetwork [15], OVN and Calico are deployed as separate use cases on top of bare-metal hosts. The high traffic situations' attitudes with regard to tested solutions are evaluated in terms of performance and reliability. The evaluations indicate that the tested container cluster networking solutions performances fluctuate, depending on traffic load and application types. In order to fulfill the CNF application requirements in terms of latency, bandwidth, etc. for dynamic traffic load situations, a high performing smart networking architecture is proposed and preliminary implementation has been accomplished.

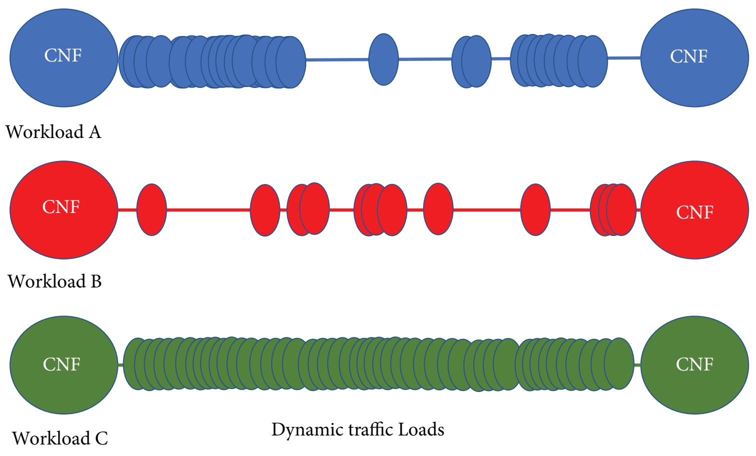


Figure 2. Dynamic traffic loads with different workloads on CNFs.

Our work is organized in 6 sections. Section 1 introduces the importance of containers usage and exigence for the analysis of CNF workloads under high traffic conditions. In addition, related works investigated in terms of load-based performance, and an outline of the research are given in this section. Section 2 presents the cluster networking solutions such as Libnetwork with Etcd, Weave, Flannel, OVN and Calico. Section 3 describes the test environment, the test scenarios, measurement tools, performance metrics and analysis techniques, together with the case studies which are implemented. Section 4 presents the results of the experimentation according to different traffic situations, by considering metrics such as throughput, retransmitted TCP segments, lost datagrams and jitter. The figures illustrating the results, together with a discussion of the results in terms of application type, traffic load and robustness, are presented. Additionally, our proposed architecture is introduced and the preliminary performance results of the architecture are given in Section 5. Eventually, Section 6 presents a conclusion, together with recommendations for future investigations.

2. Container cluster networking solutions

Containers are lightweight software constituents which are dependent on an image with the configuration running on traditional operating systems in isolated user environments [16]. Docker is the most commonly used container and makes up 83 percent of all containers in 2018, a number which is down from 99 percent in 2017 [17]. There are two common container networking models: container network model (CNM) and container network interface (CNI). The Docker default overlay network is called swarm services which provide the traffic flows between swarm managers and workers [15].

Kubernetes, which is created and open sourced by Google is one of the important Docker container orchestration system. It uses the labels and pods in order to manage applications easily by grouping them as logical units. However, Kubernetes does not manage the network directly by itself. It uses network plugins which are called CNI's which provide the networking of Kubernetes cluster's several components [18].

The Libnetwork overlay uses native virtual extensible local area network (VXLAN) features to build the overlay network together with iptables, Linux bridges and veth. The overlay datapath exists in the kernel space, and brings the benefits of less CPU usage, low latency, fewer context switches, and direct traffic between the network interface controller (NIC) and the application. Layer 2 Ethernet frames are encapsulated in Layer 3

UDP packets. By identifying each layer 2 subnets with a VXLAN header, the container traffic is traversed between hosts. This process requires VTEP (VXLAN tunnel endpoints) to encapsulate and decapsulate the packets that brings about significant performance overheads [5].

Flannel offers a layer 3 IPv4 network between the overlay host by controlling the traffic put across between hosts. Flannel has several backend modes such as UDP, VXLAN, AWS VPC and GCE [5]. Although Flannel's default backend is UDP, the VXLAN mode is commonly prefered in production, as VXLAN forwarding increases the performance of the overlay solution according to UDP. The VXLAN backend is similar to the Docker Libnetwork overlay solution. In the VXLAN backend, Linux kernel VXLAN tunnel devices are constituted, and user space process flanneld has been executed. Flannel assigns each Docker daemon an IP segment by solving the IP conflict in the Docker default configuration. However, it has the drawback that it is not possible to assign fixed IP addresses to containers and brings about a deficiency of multi subnet isolation [5]. Flannel stores the assigned subnets, routing table and hosts information in a distributed key-value store such as Etcd.

OVN, which is a system that provides virtual network partition, supplements the capabilities of Open Virtual Switch (OVS). OVS consists of three main components entitled vswitchd, ovsdb-server and kernel module (datapath). OVS utilizes the virtual bridges and forwards the packets through the flow rules between hosts, by connecting the tap interfaces and namespaces. Datapath forwards the packet by caching the flows and executing the actions on the received packets according to matched flows. This is referred to as a fast datapath. If the received packet does not match any flows, this packet is sent to ovs-vswitchd in userspace. Every packet which goes to the userspace decreases the performance of OVS and this is referred to as a slow datapath.

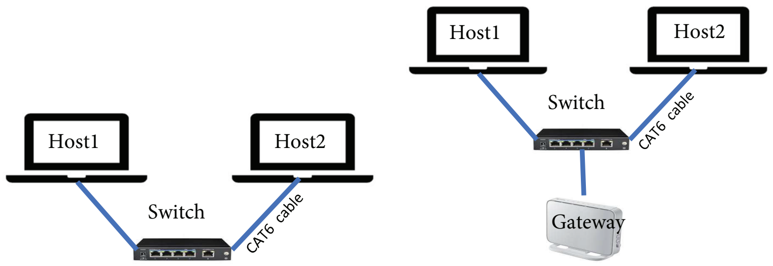
Weave is a widely deployed overlay networking solution developed by the Weaveworks company. It has two working modes entitled fastdp which is default mode, added to the Weave version 1.2 and sleeve (pcap). Weave starts with the fastdp mode except that it uses untrusted networks and encryption. Fastdp mode is quite similar to Libnetwork which uses encapsulation with VXLAN and encryption via IPSec. However, there is a prominent difference with Weave in that it uses an Open vSwitch datapath module in a Linux kernel to accelerate the traffic flows by reducing the number of context switches [13]. Running this module in a kernel space throughout the known flows boosts performance. In addition, Weave also deploys a Linux bridge for traffic broadcasting then learns and updates the flows. Weave comes with two different plugins: Weavemesh and Weave. Weavemesh, which is deployed by default, works without a key-value store by using the conflict-free replicated data type (CRDT) to store all the clusterwise information.

Calico is a solution which does not apply any encapsulation methods. Consequently, it is not an overlay solution. It is a pure layer 3 network by establishing a cluster container networking through the IP forwarding functionality in a Linux kernel. Calico uses the border gateway protocol (BGP) client entitled bird internet routing daemon (BIRD) together with a vRouter to manage the end to end layer three networking. Calico utilizes the distributed key-value store to keep information in terms of clusterwise hosts, subnet and IP addresses. Calico has a weakness in that it has very large routing tables, as every container has an IP address [5]. Although Calico has to perform a complex routing task, it is not affected by any overlay overheads such as NAT and tunnelling.

3. Experimental setup

3.1. Test environment

A multihost container networking environment consists of two bare-metal hosts, one Gigabit Ethernet switch, and one gateway as shown in Figure 3a. Computer Dell Latitude E7440 with a 4 CPUs Intel(R) Core(TM) i7-4600U CPU@2.10GHz, 16 GB memory, SATA 6 Gbps I/O interface and a 1Gbit/s Intel Corporation I218-LM network adaptor is used as host1. Host2 is a Dell Inspiron 15 7000 Gaming computer with 8 CPUs Intel(R) Core(TM) i7-7700HQ CPU@2.80GHz, 16 GB memory, SATA 6 Gbps I/O interface and a 1Gbit/s Realtek 8169 network adaptor. Each host uses the Ubuntu 16.04.5 LTS operating system with a kernel version 4.15.0-66-generic x8664. In performance investigation scenarios, the two hosts are connected through the CNet CGS-800 8 port Gigabit Ethernet switch by CAT6 cables. Except for experiments with an Apache jmeter, the gateway device is not utilized. During the external Web access benchmark tests executed with the Apache Jmeter, a 4 ports 100 Mbit/s ADSL2 VDSL2 Huawei HG658 V2 gateway device is added to the environment in order to improvise it to a real world scenario as is demonstrated in Figure 3b. In preliminary implementation of the proposed architecture, only host2 is used in order to constitute single node Kubernetes environment, which is shown in Figure 4.



(a) Iperf3 and Mongo-Perftest environ-ment (b) Jmeter tests environment

Figure 3. Experimental setup for performance investigation in Docker cluster.

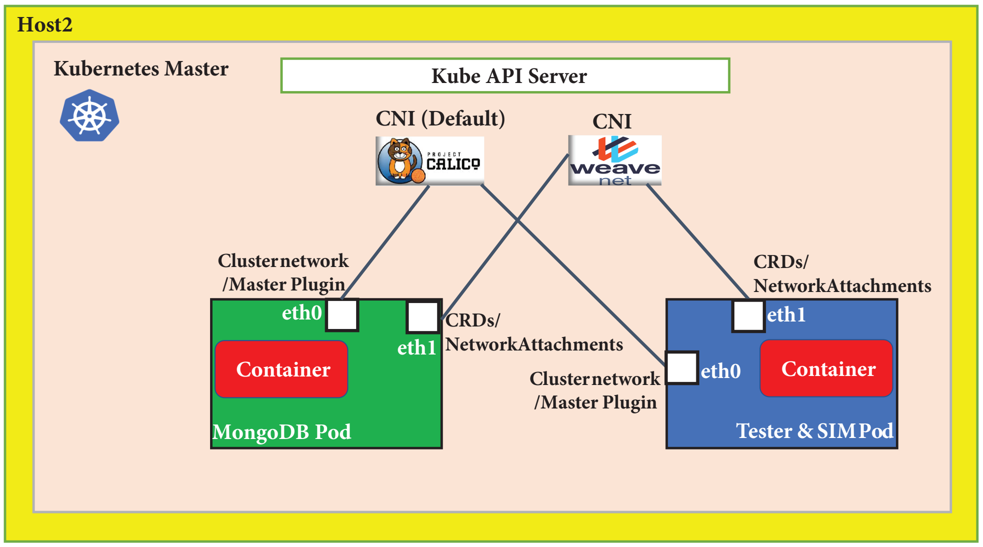


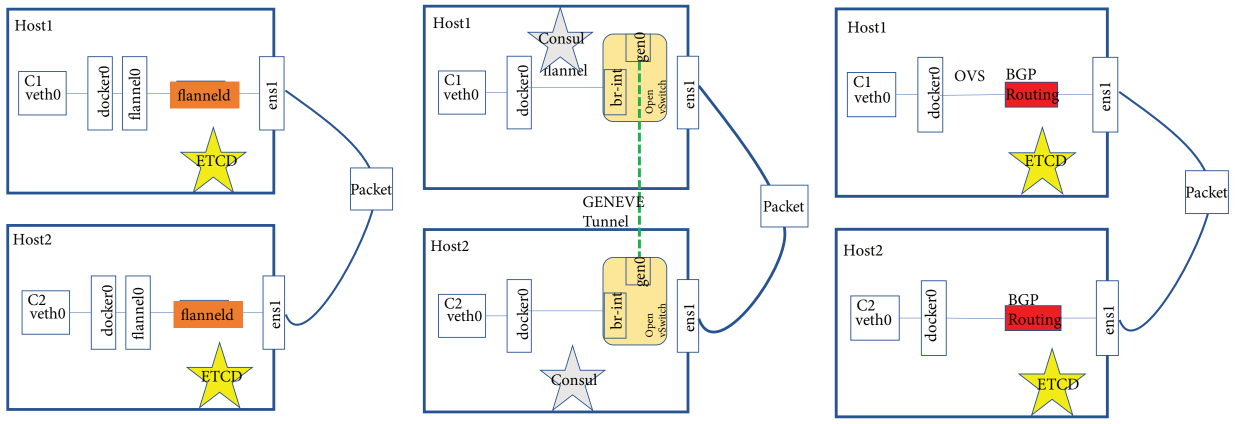
Figure 4. Experimental setup of the proposed architecture.

3.2. Use cases

Container multihost networking has been investigated within five different use cases in the form of a Docker default overlay driver with Etcd, Flannel, Weave, OVN and Calico. Each overlay scenario has been deployed in the same environment, and tested separately. The configuration of container networking solutions are kept as the default as much as possible. Interfaces in the implementations are used with their default MTU sizes. Regarding the key value stores, Etcd and Consul are utilized, depending on the scenario. Etcd is used with Libnetwork, Flannel and Calico, while OVN is deployed with Consul. Weave does not require any key value store) (KVS) as it utilizes the consistent distributed configuration model. A consul global cluster store with version 1.0.6 is deployed with OVN, unlike the Flannel, Calico and Libnetwork implementations. Libnetwork with Etcd, Flannel and Weave uses VXLAN tunnelling encapsulation, while OVN uses a Geneve encapsulation. The overlay solutions that use VXLAN and Geneve encapsulation has an additional performance loss because of the encapsulation header size. Calico is configured at the BGP mode, while Flannel is used in the VXLAN mode, Weave uses fastdp and OVN is used with the OVS learning mode in the use cases. These are given in Table 1. Flannel, OVN and Calico use case implementation topologies are shown as samples in Figures 5a–5c, respectively.

Table 1. Use cases.

Use case	Overlay method	MTU sizes (bytes)	Implementation
Libnetwork	VXLAN	1450	with Etcd
Flannel	VXLAN	1450	VXLAN mode
Weave	VXLAN	1376	Fastdp mode
OVN	Geneve	1442	Learning mode
Calico	No overlay	1500	BGP mode



(a) Flannel (b) OVN (c) Calico

Figure 5. Use case topology samples.

The new architecture implementation has been performed on Kubernetes because of its convenience and popularity. Single node Kubernetes environment has been created with two CNI's, which are Calico and Weave. Although, it is possible to add several network interfaces to a container in pure Docker cluster environment [19], custom resource definition with the type of network attachment is applied in Kubernetes environment to add multiple network interfaces to the pods [20]. Kubernetes client v1.18.2 and server v1.18.2 are utilized in the implementation. The same docker version 18.09.7, build 2d0083d, and the same Etcd version 3.3.9 are used during the implementation of all scenarios. Flannel version 0.8.0, Weave version 2.5.2, Calico version 3.2.8 and Open vSwitch version 2.12.0 are deployed in the use cases which are listed in Table 2.

Table 2. Application versions in use cases.

Application	Version
Kubernetes client, server	v1.18.2
Docker	18.09.7,build 2d0083d
Calico	3.2.8
Etcd	3.3.9
Flannel	0.8.0
Weave	2.5.2
Open vSwitch	2.12.0
Consul	1.0.6

3.3. Methodology and tools

This study applies a quantitative analysis as concrete results are educed from the numerical outputs throughout the experiments. This research is based on a physical model as all the experiments are performed on a physical environment. This consists of two computers and one switch in real time. The benchmarking tools are shown in Table 3. Iperf3 3.0.7, Apache JMeter 3.3 r1808647, and Mongo-perf r20190405 which is a micro bechmarking tool for MongoDB are used to measure the networking performance.

Table 3. Test tools.

Tool	Version
Iperf3	3.0.7
Apache Jmeter	3.3 r1808647
Mongo-perf	r20190405
Mongo server	3.2.0

Measurement tools are containerized and tests are executed in containers as it is necessary to compare the container overlay performance. The test results are gathered within 95 percent confidence interval for each use cases. MongoDB database benchmarks are executed with (1, 5, 50, 100, 200) number of threads within Mongo-perf. Commonly used database operations such as insert, update and query are tested with diversified transactions for 5-s time periods, and throughput is gathered within operations per second.

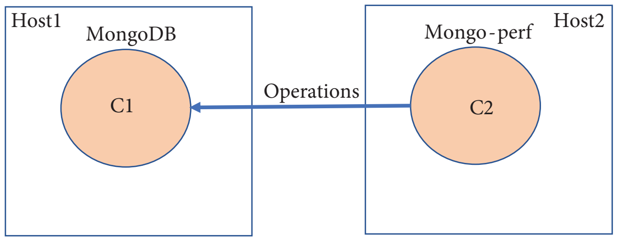
An external web access benchmark test is executed with an Apache Jmeter in a distributed way, which includes one Jmeter master (client) and one slave (server), in order to improvise it to match a real scenario. The number of HTTP requests is increased from 5 to 200 to display the behaviour under load. These are sent to a certain public web site with a thread group duration of 30 s. In this scenario, data is sent through the cluster network to the internet by assuming that the internet speed does not change. Iperf3 is operated for reliability test purposes by considering jitter, retransmitted TCP segments, and UDP lost datagrams to observe the load effects on TCP and UDP bandwidths. Therefore, a parallel (5, 10, 30, 100) number of threads are executed simultaneously for a 10-s time period.

Various measurement indicator results are observed in different numerical ranges after performing the experiments. However, performance and reliability evaluations need to be performed by courtesy of different tests which have similar intentions. Hence, it is necessary to convert the results of all the tests into the same value range in order to evaluate the group of features together. The Z-score normalization method which is formulated at (1) is selected for this purpose as it ensures that every data point has the same scale, and each test becomes equally important overall.

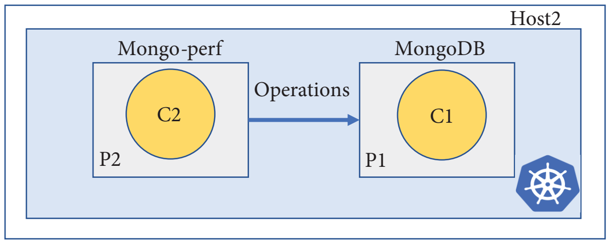
$$Z - score = \frac{value - \mu}{\sigma} \tag{1}$$

By applying normalization, if the value is the mean of all values, it is normalized to 0; if it is higher than the mean it is positive; while if it is lower than the mean it is negative. Normally, the size of negative and positive numbers are decided by the standard deviation. During the evaluation of features which relate to several test results, the normalized results are summed up by giving equal importance to each subfeature.

During the evaluation of the new architecture, MongoDB database pod has been deployed with multiple network interfaces and the performance evaluation has been performed by insert-int-vector operations. Benchmarks, has been realized by (1, 5, 40, 50, 100, 125) threads within Mongo-perf application. In our experiments, the upper bound of thread quantity is limited to 125 as the Kubernetes deployment is different from Docker cluster environment and the number of feasible threads are tightly bound to the deployment. Applying more than 125 threads causes to the crash in the containers. The Kubernetes deployment reaches full CPU utilization quite faster than Docker cluster environment, correspondingly, it could handle less threads than Docker cluster operates. MongoDB performance evaluation setup for Docker cluster environment is presented in Figure 6a and similar setup for Kubernetes environment is given in Figure 6b.



(a) Mongo-Perftest setup in Docker cluster environment



(b) Mongo-Perftest setup in Kubernetes environment

Figure 6. MongoDB performance evaluation setup comparison.

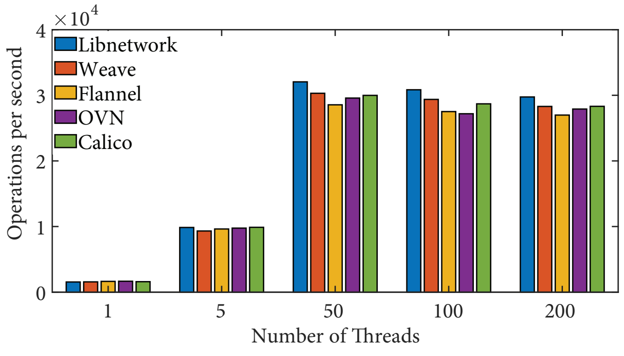
4. Performance results

Container networking solution results with multithreading, exhibit quite different behaviours compared to the single thread analysis outcomes. It is discovered from the experiments that multithreading has affected the performance of all container networking solutions in a favourable way. In addition, the container networking solutions' performance improvements are prone to the specific type of workload operation. However, there is not enough information to explain the different behaviours of the solutions within those operations. This situation is probably related to the container solutions' adequacy with regard to specific types of resource utilization.

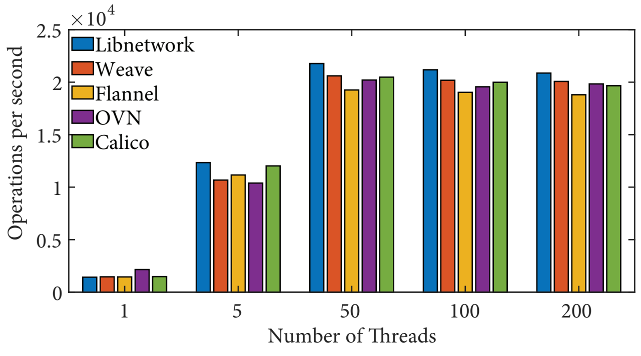
4.1. Database operations throughput

Three different insert operations (insert-empty, insert-int-vector, insert-large-doc-vector) are performed to evaluate the throughput in high traffic situations. The outcomes are illustrated in Figure 7. In insert-empty operations, all solutions make their highest number of operations within 50 threads. Then, when the thread numbers are increased, the throughput starts to decrease slightly. This situation is presented in Figure 7a. In the case of insert-empty operations, Libnetwork gives the highest throughput while Weave and Calico have quite similar performances overall. Flannel has the worst performing solution by doing 10 percent fewer operations than Libnetwork. Multithreading affects the Libnetwork in a more favourable way than the others. The Libnetwork performance is boosted 5.3 times within 5 threads, and 19.7 times within 50 threads, compared to single thread operations. With regard to the insert-int-vector operations which are illustrated in Figure 7b, the Libnetwork performance is the highest overall, while Flannel is the worst performing solution. Although OVN offers a superior performance in single thread operations, multithreading has least improvement in the case of OVN. Nevertheless, multithreading boosts the performance of Libnetwork 7.6 times with 5 threads, and 14.3 times with 50 threads.

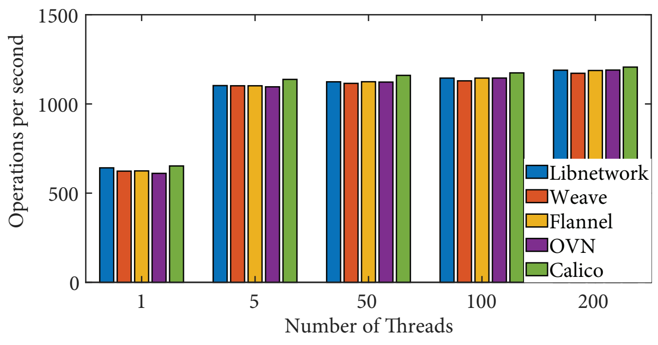
The insert-large-doc-vector operations measurement outcomes are set out in Figure 7c. Although the solutions' performance results are pretty close to each other, Calico made the highest number of operations in all numbers of threads, and its overall performance is 2.4, 2.8, 3.2, 3.66 percent higher than that of Libnetwork, Flannel, OVS and Weave in insert-large-doc-vector operations, respectively.



(a) MongoDB insert-empty operations throughput



(b) MongoDB insert-int-vector operations throughput



(c) MongoDB insert-large-doc-vector operations throughput

Figure 7. Container cluster networking solutions multithreading throughput comparison with MongoDB insert operations.

Only one type of MongoDB update operation – update-field-at-offset – is evaluated. The results are given in Figure 8. Weave has the maximum throughput with regard to all numbers of threads. Its throughput is 9.3 percent more than that of Libnetwork and Flannel when considering the whole of the test results. Networking solutions give their maximum throughput at 50 threads, and throughput starts to dwindle slightly by adding more threads. Flannel gives the worst performance at 50 threads, and Libnetwork makes the minimum number of operations at the maximum number of threads.

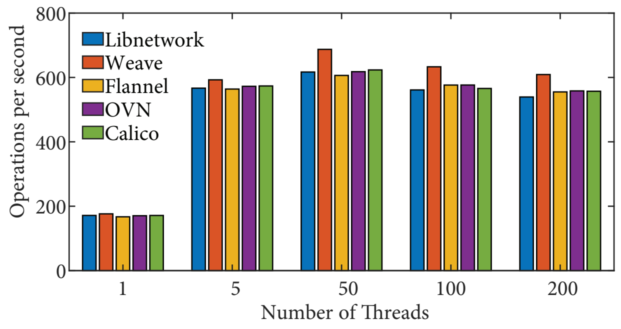


Figure 8. Container cluster networking solutions multithreading throughput comparison with MongoDB update field at offset operations.

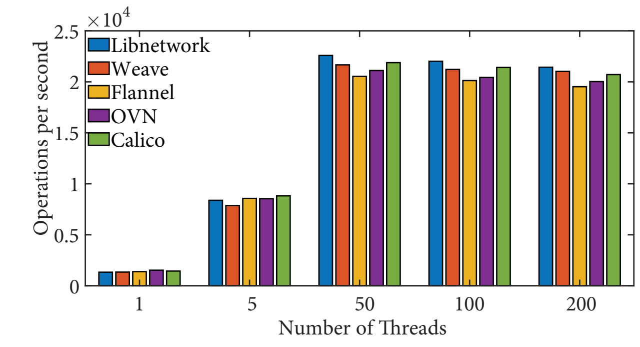
MongoDB query operations behaviour under high traffic loads is investigated with 4 different operations: query-empty, query-find-projection, query-int-id-range, and query-nomatch. The outcomes of the investigation regarding query-empty operations are depicted in Figure 9a. For query-empty operations, OVN which makes the maximum number of operations with a single thread, gives 5.5 percent less throughput than does Libnetwork with multithreading overall. Although Libnetwork gives the worst performance which is 14.7 percent less than OVN with a single thread, its performance is boosted with multithreading and become the best performing solution overall. Flannel is the worst performing solution with 8 percent less throughput than Libnetwork in the entire measurements.

For query-find-projection operations, all the tested solutions distinctively achieved the maximum performance with 50 threads. Their performance started to diminish slightly while increasing the number of threads which is shown in Figure 9b. Although OVN is the best performing solution with single thread operations, Libnetwork's performance is boosted with multithreading and become the first-ranked solution overall. Flannel makes 6.6 percent fewer operations than Libnetwork, and it is the worst performing solution among all the container networking solutions for the query-find-projection workload, which is similar to query-empty, insert-empty and insert-int-vector operations.

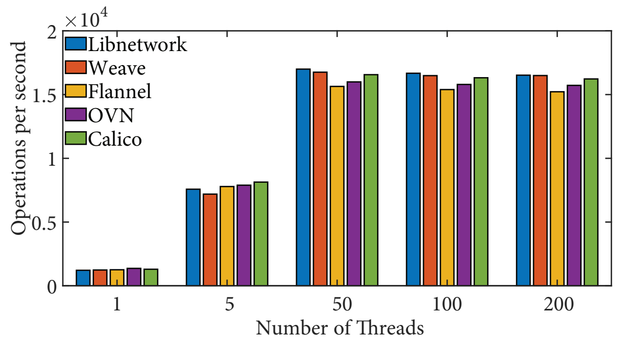
For query-int-id-range database operations as exhibited in Figure 9c, OVN makes the maximum number of operations which is 3 percent higher than that of Calico, 7.2 percent and 8.3 percent higher than Flannel and Weave respectively with a single thread. Libnetwork is the worst performing solution in terms of single thread operations and is 11.6 percent less than OVN. This alignment is changed by the effects of multithreading. Weave becomes the best performing solution with a higher performance than Calico, OVN and Libnetwork by a percentage of 4, 4.8, and 4.9 respectively for all tests including single and multithread operations. Flannel maintains its worst performing rank with 6.8 percent fewer operations than Weave for all query-int-id-range operations.

For query-no-match operations, all the tested container networking solutions achieved the maximum number of operations around 50 threads. However, the number of operations started to decrease with an increase in thread numbers as shown in Figure 9d. Distinguishing from other database operations, Calico captures the best performing solution title from OVN and gives an excellent performance both in single and 5 multithreaded operations. Calico obtains 55 percent more operations with single thread and 72 percent more operations with

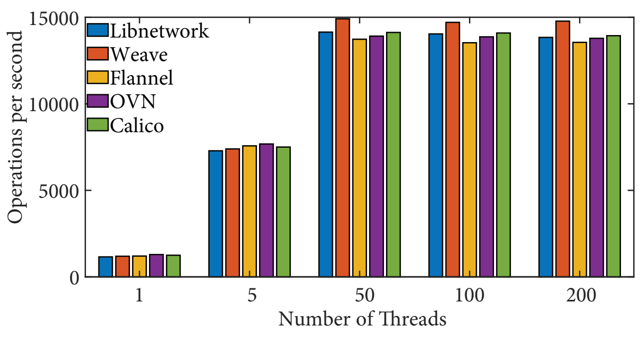
5 multi threads than OVN which is second ranked solution. Libnetwork yields the worst performance in single thread operations. However, Libnetwork performance is ascended extremely while increasing thread numbers to 50, 100 and 200. Libnetwork captures the first ranked solution title by providing 11.3 percent more operations than Calico and 0.6 percent operations than Weave with multithreading.



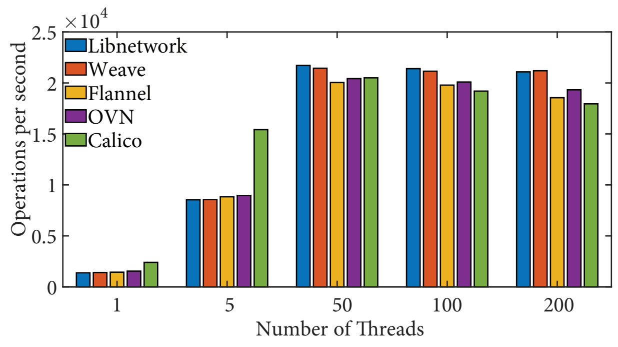
(a) MongoDB query-empty operations throughput



(b) MongoDB query-find-projection operations throughput



(c) MongoDB query-int-id-range operations throughput



(d) MongoDB query-no-match operations throughput

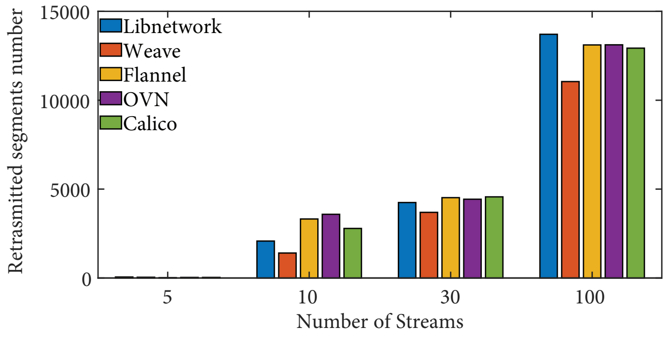
Figure 9. Container cluster networking solutions multi-threading throughput comparison with MongoDB query operations

4.2. Reliability

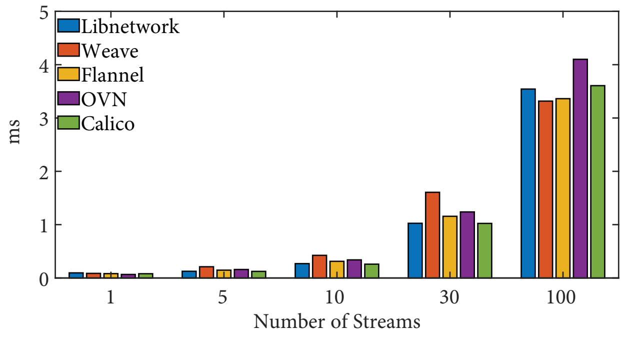
The number of retransmitted TCP segments, UDP jitter and the number of UDP lost datagrams are selected as reliability indicators, and the container cluster networking reliability under load are evaluated over those parameters. The number of TCP retransmitted segments are gathered and are shown in Figure 10a. While increasing the number of threads, Libnetwork TCP retransmissions are mounted up within 100 threads. Weave makes the least number of retransmissions in the entire test scenario. Libnetwork makes the maximum number of retransmissions on 5 and 100 threads which are the minimum and maximum number of threads.

Jitter is observed during the multithreaded evaluations on UDP and shown in Figure 10b. Libnetwork gives the maximum jitter with a single thread. Weave gives the worst jitter with 5, 10 and 30 threads, while giving a minimum jitter which is slightly less than that of Flannel with 100 threads. Calico had the minimum jitter for 5, 10, and 30 threads. Although, OVN gives the minimum jitter with a single thread, it yields the maximum jitter in the case of 100 threads.

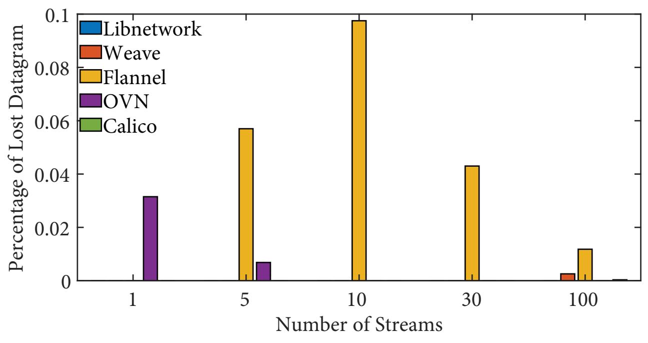
The UDP lost datagram results are displayed in Figure 10c. None of the container networking solutions lose any UDP packages in single thread operations with the exception of OVN. It is supposed that the OVN packet losses in single thread operations is the due to first hit misses and the long OVN data path because of it's learning mode. Although other solutions do not lose any datagrams until 100 threads, Flannel loses around 0.1 percent of datagrams with 10 multiple threads. Experiments reveal that Flannel loses the datagrams in all multithreaded operations, while Weave and Calico lose the insignificant amount of datagrams at the maximum number of threads.



(a) Retransmitted TCP segments



(b) UDP jitter



(c) UDP lost datagram

Figure 10. Container cluster networking solutions multithreading reliability comparison.

4.3. Web access operations throughput

The HTTP traffic average throughput with multithreading is exhibited in Figure 11. In terms of increasing the number of threads, after 100 threads, the throughput decreases for all container overlay solutions. This is probably due to the impact of resource congestion in the host. Although the networking solutions results are quite close to each other, Flannel gives the highest throughput overall. All the solutions give their maximum throughput with 100 threads. The Flannel throughput is 8.7, 10, 11, and 16 percent more than OVN, Calico, Libnetwork and Weave, respectively. Calico and Flannel give the highest performance amongst the tested container networking solutions at the maximum thread number which is 200. In 200 threads, all networking solutions' performance has significantly dropped, which is around 40 percent compared to 100 threads.

5. A new smart container networking architecture

In this research, the performance and reliability of MongoDB and web access operations which are the most commonly applied VNF processes, are evaluated under various traffic loads. It is deduced from the experiments

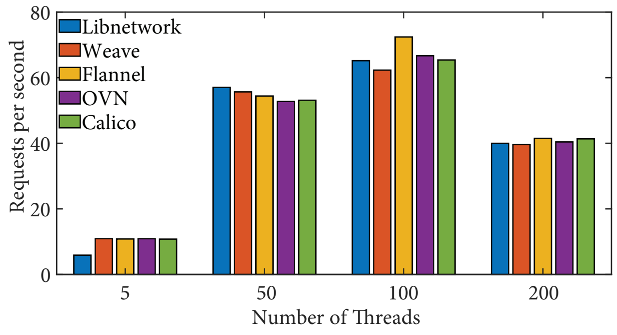


Figure 11. Container cluster networking solutions multithreading throughput comparison with web access operations.

that container networking solutions' performance changes depending on the traffic load and application type. Container networking solutions usually present maximum throughput within 50 threads which is therefore designated as the optimum load for database operations except MongoDB insert-large-doc-vector operation which is not limited with 50 threads and its performance keeps to increase while adding more threads. In addition, web access operation reaches the maximum throughput within 100 threads. Although, investigation of resource consumption is not the subject of this study, it is known that optimum thread number is quite relevant by how quickly threads can utilize CPU with 100 percent or fill network bandwidth totally etc. As different operations have different resource intensities, similar operations consume the same type of resources and have similar optimum load points. However, the operations which has different resource intensities may have different optimum load points in the same environment. For example, insert-large-doc-vector operations performance is not bounded to 50 threads and the optimum load point is taken as maximum thread number (200). The reason why, insert-large-doc-vector operations are I/O intensive and they do not utilize the 100 percent CPU quickly unlike other operations.

Besides, optimum thread number is also related with the architecture of the environment. While evaluating the same type of operations (insert-int-vector in MongoDB) in Docker cluster and Kubernetes environment, as it is observed in Table 4 and Table 5 respectively, optimum load point (40 threads) in Kubernetes deployment is less than Docker cluster environment (50 threads). This situation is explained by Kubernetes deployment reaches full CPU utilization quite faster than Docker cluster environment.

Table 4. Performance results of Weave and Calico networking solutions for MongoDB insert-int-vector operations in Docker cluster environment (operations per second).

No. of thread	Calico (ops/s)	Weave (ops/s)
1	1480.2	1458.44
5	12024.84	10675.06
50	20480.5	20604.12
100	19992.06	20183.7
200	19663.32	20070.28

The tested maximum thread number is 200 for Docker cluster environment. This is assigned as a heavy load state for web access and MongoDB operations. In order to evaluate several features together, Z-score normalization has been applied to the results, and all the values are reduced to the same scale. Under these

Table 5. Performance results of Weave and Calico networking solutions for MongoDB insert-int-vector operations in Kubernetes (operations per second).

No. of thread	Weave on K8S (ops/s)	Calico on K8S (ops/s)	SIM on K8S (ops/s)
1	3002.00	3089.88	3089.88
5	9317.51	9402.50	9402.50
40	10385.06	10416.74	10416.74
50	10061.23	10182.48	10182.48
100	9850.16	9854.25	9854.25
125	9646.20	9348.62	9520.75

circumstances, by utilizing the normalized results, high performing container networking solutions affiliated to application type, traffic load and robustness demand are pieced together in Table 6.

Table 6. High performing networking solutions according to test results by considering application requirements, traffic loads and robustness.

Application	HL,R	HL,UR	OL,R	OL,UR
Web acess	Calico	Flannel	OVN	Flannel
MongoDB insert	Calico	Libnetwork	Calico	Libnework
MongoDB update	Weave	Weave	Weave	Weave
MongoDB query	Weave	Weave	Weave	Libnetwork

HL: High load, OL: Optimum load, R: Robust, UR: Unrobust.

The high performing container networking solutions which have been discovered are as indicated in Table 6. The advice is to use Calico for web access and MongoDB for insert operations, while using Weave for MongoDB updates and for query operations if robustness is required under a heavy load traffic. Peradventure robustness is not taken into account, Flannel provides the best performance for Web access operations, and Libnetwork provides a higher throughput for MongoDB insert operations under heavy load conditions.

The tested networking solutions highest throughput values are obtained at the optimum traffic load points amongst the total range of traffic loads. If robustness is necessary, it is advised that OVN for Web access operations, Calico for MongoDB insert operations, and Weave for MongoDB update and query operations in optimum traffic load is used. If peradventure robustness is not necessary, the use of Flannel for Web access operations, Libnetwork for MongoDB insert and query operations in optimum traffic load is proposed.

It is concluded from the evaluation presented in Table 6 that solution behaviours differ dynamically depending on traffic load and application type. Therefore, it is impossible to use the suggested solutions adaptively with traditional container networking implementation methods. To overcome this problem, a new smart container networking implementation design is proposed, which is shown in Figure 12.

In container cluster environment, a new component, smart container network interface manager (SIM), is inserted between the container runtime and the container networking solutions, which is given in Figure 12b. In smart container networking architecture, containers are linked to available container networking solutions, which is presented at Figure 12a. Initially, SIM tests the available interfaces by the help of monitoring tools, then it selects the highest performing network interface of container as active path. Afterwards, two containers transfer the data through these selected network interfaces. SIM simultaneously observes the performance of all container networking solutions; at that point, testing mechanisms are implemented by the help of container monitoring tools. When the performance of active path has degraded, traffic is switched to more adequate interfaces of containers by SIM. It is experienced with the implementation that switching decision should be given if performance gain is higher than switching loss.

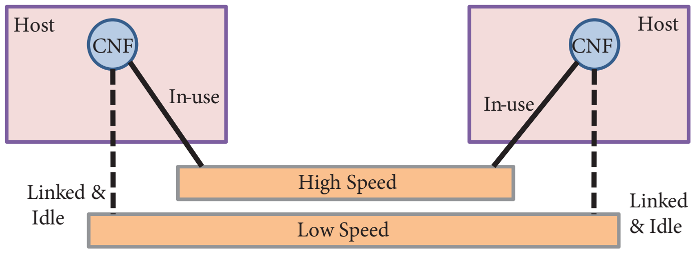
The decision process is quite important to ensure performance gain. For this purpose, SIM decision mechanism needs to be supported by machine learning algorithms such as supervised learning or kinds of methods from artificial intelligence. SIM interface selection flow diagram has given in Figure 12c.

Primary implementation of SIM is realized in single node Kubernetes environment, which is demonstrated in Figure 4. Pods are linked to two available networks, which are Calico and Weave by applying custom resource definition with the type of network attachment [20]. One MongoDB database has been located in one of the pod. Mongo-perf tester together with Python application in another pod has acted as SIM. Initially, standalone performances of Calico and Weave has been evaluated by insert-int-vector operations. Networking solutions' performance results in Docker cluster and Kubernetes environments are shown in Tables 4 and 5, respectively.

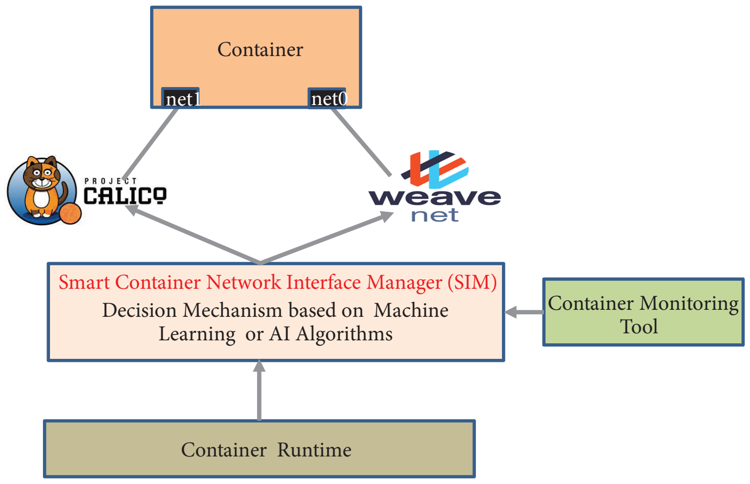
It is clearly seen in Figure 6 that MongoDB container is located in host1 and network bandwidth has been limited by the physical connection between host1 and host2 for Docker cluster networking. For Kubernetes cluster, MongoDB and Mongo-perf pods are located in host2, hence there is no physical network connection which limits the traffic. In addition, host1 and host2 have equal amount of resources in terms of network, memory, I/O interface and disk specifications except host2 processor specifications are better than host1.

In this situation, it is expected to have higher performance results in Kubernetes environment compared with Docker cluster environment, as there is no factor which limits the traffic. In addition, there are twofold number of CPUs with higher frequencies in Kubernetes environment. Single thread operations meet this expectation as shown in Table 5. Number of operations of Calico and Weave networking solutions in Kubernetes environment with single thread, is higher than Docker cluster environment. However, while increasing the number of threads, Docker cluster environment gives higher performance results than Kubernetes environment. Kubernetes is a larger system with more features and several objects compared to Docker cluster environment. Hence, all those components consume host resources. Moreover, the performance of the implementations are generally evaluated by how quickly they can reach 100 percent resource utilization and by number of operations in a second. In addition, database performance is influenced by availability of memory, CPU, physical-virtual network bandwidth and disk I/O bandwidth [21]. It is put forth by [21] that Kubernetes consumes more memory than Docker cluster and spends more CPU time on soft interrupts than Docker cluster. Furthermore, these CPU resources are generally spent for virtual networking between containers. Hence, Kubernetes total (physical and virtual) network usage is also quite higher than Docker cluster. In spite of these, Kubernetes presents better performance than Docker cluster for small number of threads (e.g., 1 thread) that can be explained by its I/O path implementation method which uses hostPath volume. Consequently, Kubernetes deployment reaches full CPU utilization quite faster than Docker cluster environment, correspondingly, Kubernetes performance with respect to number of operations in a second is higher than Docker cluster environment before it is overloaded. This also explains the reason of applicable thread number in Kubernetes environment is less than Docker cluster environment.

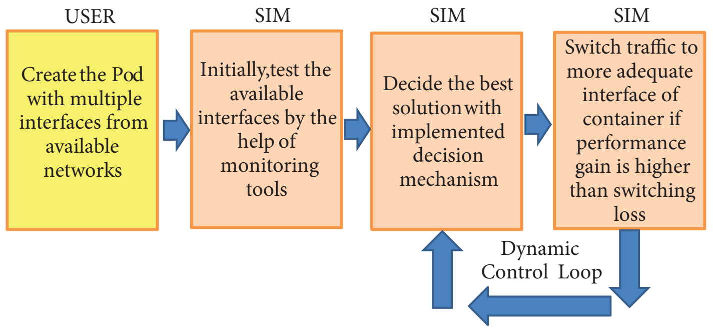
The results given in Tables 4 and 5 output that tested container cluster networking solutions relative performance and their behaviour under load in Kubernetes environment are consistent with the results which are gathered from Docker cluster. In Kubernetes environment, Calico gives highest results between threads 1 to 100, however Weave presents better results under high loads such as 125 threads.



(a) New architecture which supports several container cluster networking solutions at the same time


Container
net1
net0
PROJECT CALICO
weave net
Smart Container Network Interface Manager (SIM)
Decision Mechanism based on Machine Learning or AI Algorithms
Container Monitoring Tool
Container Runtime

(b) Dynamic container network interface selection by Smart Interface Manager


USER
Create the Pod with multiple interfaces from available networks
SIM
Initially,test the available interfaces by the help of monitoring tools
SIM
Decide the best solution with implemented decision mechanism
SIM
Switch traffic to more adequate interface of container if performance gain is higher than switching loss
Dynamic Control Loop

(c) SIM interface selection flow diagram

Figure 12. Smart networking architecture for CNFs.

In smart networking architecture, SIM imitation (Python application) has performed switching between Calico and Weave starting from 1 times to 3000 times during 5000 insert operations in order to analyze switching overhead. During this period, networking solutions performed equal number of operations. It is observed from the results which are given in Table 7 that one switching costs 13.42 (ms) and overall switching loss (ms) does not increase linearly while increasing the quantity of switching which requires further investigation about its trend.

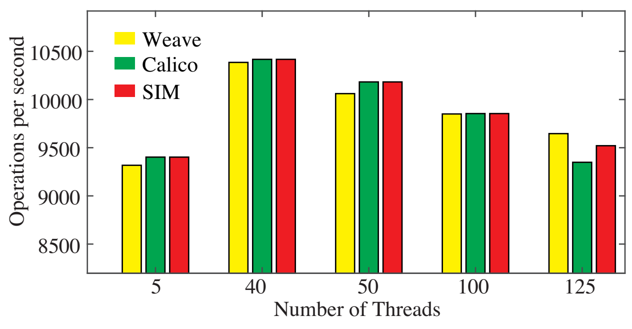


Figure 13. Overall performance improvement with SIM.

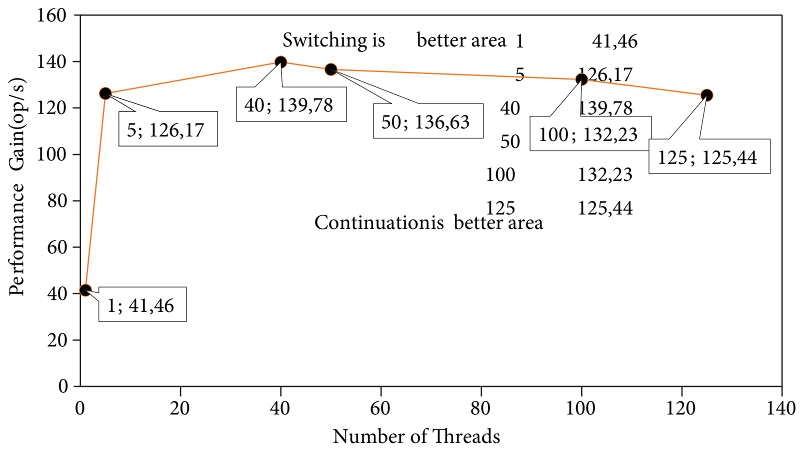


Figure 14. Trade-offs between performance gain and switching loss.

Table 7. Analysis of switching overhead.

Operation type	Completion time (ms)	Switching loss (ms)
Calico wout switching	1640.96	-
Weave wout switching	1650.13	-
1 time switching between Calico and Weave	1658.96	13.42
5 times switching between Calico and Weave	1659.76	14.21
100 times switching between Calico and Weave	1675.89	30.34
500 times switching between Calico and Weave	1678.89	33.35
1000 times switching between Calico and Weave	1701.95	56.40
3000 times switching between Calico and Weave	1800.54	155

Measurements are taken by making 5000 insert operations.

In this experiment, SIM selected the Calico during MongoDB operations between 1 to 100 threads because of its higher performance under these traffic load, then Weave is selected while performing operations under 125 threads. By employing Weave with 125 threads, 172 more operations in a second has been gained compared to traditional Calico (only) performance results, though there is switching loss. This is because Weave has better performance under high traffic loads. Consequently, 1.84 percent performance improvement has been achieved in overall compared with traditional Calico (only) scenario by utilizing dynamic interface selection. These results are presented in Figure 13.

This study also shows that although there is trade-offs between performance gain and switching loss, it is possible to achieve higher performance results with appropriate and on time decision mechanism. According to implemented scenario, a sample trade-offs table is given in Figure 14. This figure has been constituted by switching traffic between Calico and Weave, while performing insert-int-vector operations on MongoDB for different thread quantities. It shows that if the performance gain is higher than switching cost, switching improves the overall performance. The area above the line is indicated as switching better area.

As a result, dynamic interface selection solves the problem of inadequacy of single container networking solution under variety of traffic loads and application types. This implementation achieves higher performance results by dynamically selecting the available networking solutions. The success of smart networking architecture comes from accurately evaluating the measurements and making better decisions.

6. Conclusion

In this paper, performance and reliability has been investigated for the container networking solutions Libnetwork, Flannel, Calico, OVN and Weave. The most commonly used container network functions in the form of MongoDB and web access are tested under different traffic loads in terms of throughput, jitter, retransmitted TCP segments and lost datagrams. By evaluating all the experiments, the tested networking solutions behaviours depending on application type under specific traffic loads for CNF workloads are revealed. It is observed that none of the solutions provided relatively high throughput for all types of workload under optimum or heavy load conditions. As it is impossible to apply the suggested networking solutions adaptively with traditional container networking implementation methods, a new smart networking architecture is proposed. This proposed architecture has a strong decision mechanism which is provided by the component named as SIM. In this paper, the primary implementation of the architecture is given and analysed by allowing containers to use several networking solutions dynamically. This study has shown that although there is trade-offs between performance gain and switching loss, our architecture exhibits better performance than traditional architecture. Our performance results show that the proposed architecture presents better performance as long as appropriate and on time decision making mechanism is provided. Hence, this new smart container networking architecture is promising candidate, which can solve the traditional container networking implementation methods' inadequacy about meeting 5G NFV implementation diversified operational requirements for different kind of applications and traffic density.

As a future work, more sophisticated smart container networking architecture can be enhanced by developing strong decision mechanism supported by machine learning algorithms and artificial intelligence for actual deployments. The SIM networking selection mechanism can be improved in a learning mode by using artificial intelligence. While increasing the number of switching, trend in the switching loss can be investigated. In addition, container monitoring functionality can be integrated into SIM by monitoring the traffic flows concurrently. This facilitates and accelerates the different networking solutions performance testing during the network interface selection process.

Acknowledgment

The authors would like to thank anonymous reviewers for their valuable comments. This work is supported by Bahçeşehir University (İstanbul, Turkey) and Ulak Haberleşme A.Ş. (Ankara, Turkey).

References

[1] Anderson J, Hu H, Agarwal U, Lowery C, Li H et al. Performance considerations of network functions virtualization using containers. In: International Conference on Computing, Networking and Communications; New York, NY, USA; 2016. pp. 9-16. doi: 10.1109/ICCNC.2016.7440668

[2] Rotter C, Farkas L, Nyíri G, Csatári G, Jánosi L et al. Using Linux containers in telecom applications. In: Innovations in Clouds, Internet and Networks; New York, USA; 2016. pp. 234-241.

[3] Struye J, Spinnewyn B, Spaey K, Bonjean K, Latr S. Assessing the value of containers for NFVs: a detailed network performance study. In: 13th International Conference on Network and Service Management; Tokyo, Japan; 2017. pp. 1-7. doi: 10.23919/CNSM.2017.8256024

[4] Bolivar LT, Tselios C, Area DM, Tsolis G. On the deployment of an open-source, 5G-aware evaluation testbed. In: 6th IEEE International Conference on Mobile Cloud Computing, Services and Engineering, MobileCloud 2018; Bamberg, Germany; 2018. pp. 51-58. doi: 10.1109/MobileCloud.2018.00016

[5] Zeng H, Wang B, Deng W, Zhang W. Measurement and evaluation for Docker container networking. In: 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery; Nanjing, China; 2017. pp. 105-108. doi: 10.1109/CyberC.2017.78

[6] Ruan B, Huang H, Wu S, Jin H. A performance study of containers in cloud environment. In: Advances in Services Computing - 10th Asia-Pacific Services Computing Conference; Wuhan, China; 2016. pp. 343-356. doi: 10.1007/978-3-319-49178-3

[7] Martin JP, Kandasamy A, Chandrasekaran K. Exploring the support for high performance applications in the container runtime environment. Human-centric Computing and Information Sciences 2018; 8 (1): 1-15. doi: 10.1186/s13673-017-0124-3

[8] Casalicchio E, Perciballi V. Measuring Docker performance: what a mess. In: ICPE 2017, Companion of the 2017 ACM/SPEC International Conference on Performance Engineering; New York, NY USA; 2017. pp. 11-16. doi: 10.1145/3053600.3053605

[9] Herbein S, Dusia A, Landwehr A, McDaniel S, Monsalve J et al. Resource Management for Running HPC Applications in Container Clouds. In: Kunkel J, Balaji P, Dongarra J (editors). High Performance Computing. ISC High Performance 2016. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9697. Cham, Switzerland: Springer, 2016, pp. 261-268.

[10] Buzachis A, Galletta A, Carnevale L, Celesti A, Fazio M et al. Towards osmotic computing: analyzing overlay network solutions to optimize the deployment of container-based microservices in fog, edge and IoT environments. In: 2018 IEEE 2nd International Conference on Fog and Edge Computing, ICFEC 2018 - In conjunction with 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, IEEE/ACM CCGrid; USA; 2018. pp. 1-10. doi: 10.1109/CFEC.2018.8358729

[11] Kang H, Tao S. Container-based emulation of network control plane. In: HotConNet 2017 Proceedings of the 2017 Workshop on Hot Topics in Container Networking and Networked Systems, Part of SIGCOMM 2017(9781450350587); New York, USA; 2017. pp. 24-29. doi: 10.1145/3094405.3094410

[12] Suo K, Zhao Y, Chen W, Rao J. An analysis and empirical study of container networks. In: IEEE Conference on Computer Communications, IEEE INFOCOM; New York, USA; 2018. pp. 189-197. doi: 10.1109/INFOCOM.2018.8485865

[13] Hermans S, Niet P. Docker overlay networks performance analysis in high-latency environments. Bachelor Thesis, University of Amsterdam, Netherlands, 2016.

[14] Bankston R, Guo J. Performance of container network technologies in cloud environments. In: IEEE International Conference on Electro Information Technology; USA; 2018. pp. 277-283. doi: 10.1109/EIT.2018.8500285

[15] Brown CL. Network performance analysis on a containerized testbed. Master's thesis, Tennessee State University, Nashville, TN, USA, 2018.

[16] Boza EF, Abad CL. A case for performance-aware deployment of containers. In: WOC 2019 - Proceedings of the 2019 5th International Workshop on Container Technologies and Container Clouds, Part of Middleware, (9781450370332); USA; 2019. pp. 25-30. doi: 10.1145/3366615.3368355

[17] Bachiega NG, Souza PSL, Bruschi SM, Souza SRS. Container-based performance evaluation: a Survey and challenges. In: Proceedings - 2018 IEEE International Conference on Cloud Engineering, IC2E 2018; Orlando, FL, USA; 2018. pp. 398-403. doi: 10.1109/IC2E.2018.00075

[18] Burns B, Beda J, Hightower K. Kubernetes: Up and Running. USA: O'Reilly Media, Inc, 2019.

[19] Smith R. Docker Orchestration: A Concise, Fast-paced Guide to Orchestrating and Deploying Scalable Services with Docker. Birmingham, England: Packt Publishing, 2017.

[20] Caban W. Architecting and Operating OpenShift Clusters: OpenShift for Infrastructure and Operations Teams. Berkeley, CA, USA: Apress Media LLC, 2019.

[21] Truyen E, Van Landuyt D, Lagaisse B, Joosen W. Performance overhead of container orchestration frameworks for management of multi-tenant database deployments. In: Proceedings of the ACM Symposium on Applied Computing; New York, NY, USA; 2019. pp. 156-159.