Performance Evaluation of Container based Virtualization on Embedded Microprocessors

Vivian Noronha*, Maximilian Riegel**, Ekkehard Lang** and Thomas Bauschert***
*{name.1.surname.ext}@nokia.com, **{name.surname}@nokia.com, ***{name.surname}@etit.tu-chemnitz.de

***Abstract*—Container based virtualization is widely used in cloud computing. Applying such virtualization to IoT devices with much smaller footprint promises considerable benefits. This paper presents the results of a performance evaluation of system container virtualization like LXC provisioned on small MIPS or ARM based embedded microprocessors that can be used in the edge or fog of an IoT ecosystem. The measurement results using different benchmark tools show that the overhead introduced by virtualization has only negligible effects on the overall performance of the embedded microprocessors.**

Keywords—LXC; linux containers; container based virtualization; system containers; performance evaluation; benchmark; embedded microprocessors.

I. INTRODUCTION

The Internet of Things (IoT) denotes the attachment of embedded systems to the Internet. Through it, a vast number of previously isolated embedded processing systems are getting connected to communication networks. Driving forces of this move towards interconnected embedded systems are the huge demand for collection of state information in embedded systems to facilitate big data applications, and the additional capabilities for intelligent control and operation, which becomes feasible, when embedded systems are connected to data centers or to the cloud. The IoT is facilitated through the availability of very low-cost microprocessors with sufficient computing power [1].

However, connecting embedded devices to a communication infrastructure and running multiple processes also creates various new challenges to embedded systems. The same requirements regarding security, reliability, maintainability, and configurability, which are common for multiprocessing server environments, are now appearing to small embedded systems. These are:

Isolation/security of applications on a shared processor system: Two (or more) processes running on the same system should be separated from each other such that misbehavior of one process does not violate the security of any other process.

Fine grained resource control: It is necessary to control access to shared resources, monitor resource consumption, change the limits based on external events, and collect accounting metrics.

Network virtualization: To support at least basic quality of service (QoS) and traffic flow isolation within a shared-resource network, the use of network virtualization is recommended.

Run multiple versions of packages and its dependencies: Since applications are independently developed, operated and maintained, it should be possible to run multiple versions of packages with their respective libraries in parallel.

Portability: Applications need to be abstracted from the host operating environment to become deployable on many systems. Instances of applications should be easily moved across systems i.e. cloned, backed up and deployed rapidly.

Operating at bare metal speeds with low performance overhead: Application should be able to run at speeds provided by the host OS with little or no performance degradation.

To tackle these challenges, virtualization could be applied to small embedded systems. However, there is hardly any information available about the effects and impacts of virtualization on embedded MIPS and ARM based microprocessors. This motivates our work on a comparative investigation. In this paper, we present the results of a performance evaluation applying different benchmark tools on embedded microprocessors with container-based virtualization. The target is to determine and quantify the overhead introduced by the virtualization technology compared to a non-virtualized environment.

The remainder of this paper is organized as follows. Section II provides some background information about container based virtualization and its benefits. Section III outlines some examples of applying different virtualization solutions. Section IV gives a detailed description of the methodology and the experimental setup used to carry out our performance evaluation study. The impact of using container based virtualization on different embedded microprocessors is evaluated considering various aspects like CPU, networking and memory performance. Section V concludes the paper and provides some ideas for future work.

II. VIRTUALIZATION TECHNOLOGIES

In this section, we provide an overview of different virtualization technologies focused on Linux. The main benefits of virtualization are hardware independence, isolation, secure process environments, and increased scalability. Virtualization enables leveraging the computing, storage and network resources of embedded devices for multiprocessing. In the case of a Virtual Machine (VM), the hypervisor takes over the CPU core and intercepts all privilege calls made by the guest OS to

978-0-9883045-5-0/18/$31.00 ©2018 ITC
DOI 10.1109/ITC30.2018.00019

pretend that it has its own hardware. Although VMs excel in isolation, they introduce overhead when performing interactions between the guest OS and the underlying hardware. Also, the exchange of data requires complex system calls and expensive resource marshaling. The overhead of using multiple kernels on the same system motivates the use of container-based virtualization which enables a single OS instance to provide an extra layer of isolation [2].

A. *Container based virtualization*

We consider OS level virtualization based on the Linux kernel features namespaces[1] and cgroups[2]. Container based virtualization isolation is achieved by the addition of six new flags to the system calls clone(), setns() and unshare(). This allows creating separate instances of previously-global namespaces. Linux implements mount, process identification number (PID), network, user, inter-process communication (IPC), and UNIX Timesharing System (UTS) namespaces. Namespaces can be used to create an isolated container for single or multiple processes, that has no visibility or access to objects outside the container. Processes running inside the container appear to be running on a normal Linux system although they are sharing the underlying kernel with processes located in other namespaces.

The Linux control groups (cgroups) subsystem enables to group processes and manage their aggregate resource consumption. It is commonly used to limit the memory, CPU and I/O consumption of containers. Regarding networking, it's mostly about allowing the network layer to match packets to cgroups. The actual control part still happens at the network side through iptables and Linux traffic control (tc). A container can be resized by simply changing the limits of its corresponding cgroup subsystem. Because a containerized Linux system has only one kernel which has full visibility into the containers there is only one level of resource allocation and scheduling.

Fig. 1 shows the general architecture for container based virtualization. The container instances consist of applications with all their libraries running as isolated processes in user space while sharing the kernel OS of the host.

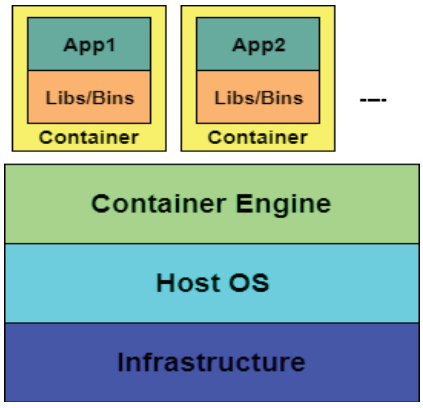

Fig. 1. Container based virtualization architecture

[1] http://man7.org/linux/man-pages/man7/namespaces.7.html
[2] https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt

B. *System containers on embedded microprocessors*

LXC (system container) is a Linux-native, open source container technology that is supported by the core functionalities of the Linux OS itself. LXC is written in the C programming language and is portable to most Linux distributions. LXC provides a "normal" OS environment and therefore LXC is compatible with all applications and any management and orchestration framework. It does not require any changes to the application that is being deployed and it can provide controlled root access to the system hardware. It relies on the functionalities provided by the kernel to enable resource isolation (CPU, memory, device I/O, network, etc.) and separate namespaces to isolate view of applications in the OS. It supports layers i.e. persistent storage and enables Copy-On-Write cloning and snapshots and is also file-system neutral. LXC supports assignment of static IPs and routable IPs, usage of multiple network interface types, management of /etc/hosts file and basically uses the full stack of Linux network capabilities. It therefore yields the advantages of container technology without changing the way applications are deployed in a system. The memory footprint of the container engine LXC is small (~1.5MB) on most Linux OS. Due to these benefits, LXC is selected as virtualization solution for our study.

III. RELATED WORK

In the literature, several studies related to the application of container-based virtualization on various devices can be found. For instance, Docker containers are mainly considered as virtualization solution for Raspberry Pi boards whereas LXC containers are often used for high performance computing (HPC) systems.

A. Kovács [3] applies CPU and network benchmark tests to compare the performance of KVM, HPC singularity, Docker and LXC to the native case of a high-end server hardware. The disk I/O benchmark is ignored due to the use of a NAS or NFS filesystem. D.Bassera et al [4] suggest LXC as the recommended technology for I/O bound applications in System on a Chip (SoC) based systems. Docker showed a high variance in performance and an unpredictable execution behavior that may prevent a proper deployment of computing applications on SoC-based systems. Another study [5] benchmarks KVM, Xen, and Linux Containers (LXC). It compares the runtime of each virtualization environment to that of a physical server, based on a sequence alignment software that arranges sequences of DNA. Further, M.G. Xavier et al. [6] analyze MapReduce clusters in high-performance computing (HPC) environments using containers. The main difference of our work compared to [5] and [6] is that we focus on small scale embedded devices.

H. Chang et al [7] identify and analyze the requirements for efficiently designing IoT gateways. They consider Docker containers to be a suitable technology for meeting the requirements. In their study, several synthetic and application benchmarks are used to quantify the overhead introduced by the virtualization layer. However, the results of the performance analysis are limited as only Raspberry Pi boards are considered. In [8] several synthetic and application bench-

marks of Docker and underlying Linux containers on two versions of the Raspberry Pi SoC platform (representing two generations of loT gateway hardware) are performed. The results show comparable performance to Linux containers on high-end servers (see [2]). R. Morabito [9] assesses the feasibility of running virtualized instances on a broad range of low-power nodes such as Single Board Computers (SBC) using Docker as a container engine. None of the mentioned papers take into consideration lightweight LXC as a container engine for smaller embedded devices.

IV. Performance Evaluation

In this section, we outline the applied performance evaluation methodology and present the obtained results.

A. Methodology

For the performance evaluation, various embedded microprocessors for Wi-Fi routers were used. The hardware specifications of the devices are listed in Table A:

Table A. Hardware features of various devices

Devices	**Linkit Smart 7688**[3]	**TP-Link Archer C7**[4]	**Linksys EA4500**[5]	**Raspberry Pi 3B**[6]
Chipset and CPU	MediaTek MT7688AN (580 MHz)	Qualcomm Atheros QCA9558 (720 MHz)	Marvell 88F6282 (1.2 GHz)	BCM2837 quad-core Cortex A53(1.2 GHz)
Flash	32 MiB	16 MiB	128 MiB	256 MiB
RAM	128 MiB	128 MiB	128 MiB	1 GiB
Ethernet Speed (MB/s)	10/100	10/100/1000	10/100/1000	10/100
Device Type	IoT, Development board	Wireless Router	Wireless Router	SBC
Architecture	MIPS 24KEc V5.5	MIPS 74Kc V5.0	ARMv5	ARMv8
Price range	15$	70$	150$	35$
Notation	**lks7688**	**archc7**	**l4500**	**rspi3**

The most relevant differences are w.r.t. CPU, flash and ethernet capabilities. As host OS, an open source embedded Linux OS named LEDE (Linux Embedded Development Environment)/Openwrt[7-8] with kernel 4.4.92 i.e. 17.01.4 release is used. The firmware image includes the LXC 2.1.1 stable package. The base container image used to virtualize the benchmarking tools consists of the lmbench micro-benchmark [10], netperf[9], nuttcp[10], perf[11], iperf3[12], sysbench[13] and Linpack[14]. The configuration of the LXC containers is similar for all evaluated devices. The benchmark tools measure the CPU, Memory and Network I/O performance by applying generic workloads. The native performance of the devices is determined by running the benchmark tools without the virtualization layer. The networking related measurements are performed using an Intel Core i7 4810MQ PC running Linux Ubuntu 16.04.3 with kernel version 4.10.0.40 with an Intel gigabit ethernet card as test machine (TM). The TM is directly connected to the network interface card (NIC) of the device under test (DUT). Unless stated, all benchmark measurements are repeated 50 times and the results are averaged. Default cgroup version 1 limitations for the resources are configured on the virtualized LXC instances. The floating-point emulation kernel feature is enabled for all devices. The rspi3 is equipped with a 32GB SanDisk Ultra Micro SDHC Card and configured to use 256 MiB as its flash.

B. CPU Performance Evaluation

To avoid the impact of processor affinity (CPU pinning) on the performance, each virtual CPU (vCPU) can run on any physical CPU (pCPU) core providing higher CPU utilization [11]. All evaluated devices except rspi3 have only one physical CPU core and thus avoid altogether the effects of processor affinity.

The sysbench benchmark is used to perform a stress test on the CPU by calculating prime numbers up to a defined value (10000) by incrementing values and performing modulo operations. From Fig. 2 one can see that the execution time increases linearly in the case of lks7688, archc7 and l4500 with the number of instances. This is a expected result as the above devices have a single core only. Due to the 4 CPU cores of rspi3, there is only noticeable overhead when the number of instances are larger than 4. The overhead introduced by LXC is about 4% in the worst case. Thus, rspi3 outperforms the other DUTs.

Another tool used for benchmarking the CPU performance is the High Performance Linpack (HPL) benchmark. The Linpack benchmark measures the computer's floating-point execution speed. It is determined by running a program that solves a dense system of linear equations A * X = B where the matrix A (of size N=1000) and the right-hand side vector B are random. Linpack yields the floating point execution speed in MegaFLOPS (mflops) units i.e. millions of floating point operations per second. Fig. 3 depicts the outcome of the Linpack test. It is obvious that the LXC virtualization layer introduces no relevant overhead. The DUTs have enough memory capacity to run 16 virtual instances in parallel. No performance degradation (change in mflops per virtual instance) is observed even when the number of instances is increased to 16.

[3] https://wikidevi.com/wiki/SeeedStudio_LinkIt_Smart_7688
[4] https://wikidevi.com/wiki/TP-LINK_Archer_C7_v2.x
[5] https://wikidevi.com/wiki/Linksys_E4200_v2
[6] https://wikidevi.com/wiki/RPF_Raspberry_Pi_3_Model_B
[7] https://lede-project.org/
[8] https://openwrt.org/
[9] https://hewlettpackard.github.io/netperf/
[10] https://www.nuttcp.net/Welcome%20Page.html
[11] http://www.brendangregg.com/perf.html
[12] https://github.com/esnet/iperf
[13] https://github.com/akopytov/sysbench
[14] http://www.netlib.org/benchmark/hpl/

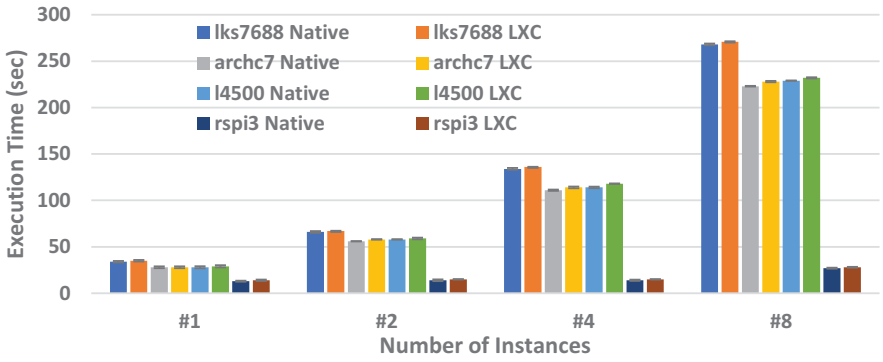

Fig. 2. Sysbench test results

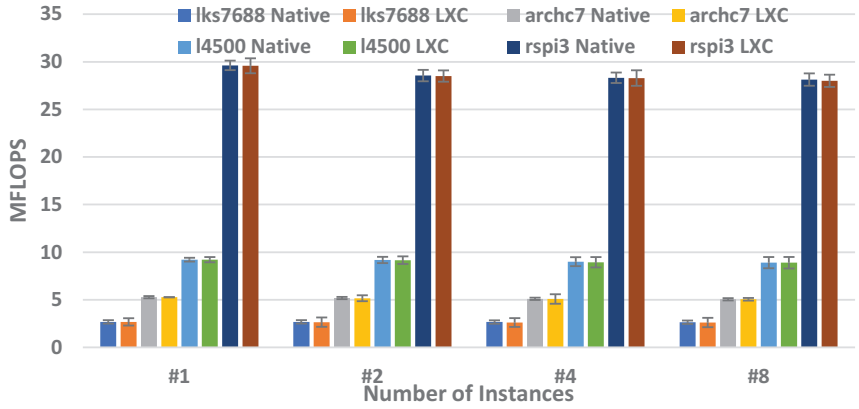

Fig. 3. Linpack test results

C. *Networking Performance Evaluation*

The Virtual NICs for all running containers share the same network bridge/interface, which in turn is mapped to the physical ethernet card. Two types of network drivers are used on the virtualized instances for networking performance evaluation: *Virtual ethernet (veth) bridge and Macvlan* [12]. Fig. 4 shows the data flow from default namespace to container namespace using veth pipes and macvlan network kernel modules, respectively. Macvlan is a kernel module which enslaves the driver of the NIC in kernel space. The module allows for new devices to be stacked on top of the default device. The veth kernel module creates a pair of (virtual) networking devices that are connected to each other. veth pipes are often used in combination with Linux bridges to provide an easy connection between a namespace and a bridge in the default networking namespace.

The Nuttcp tool is used to measure the goodput of a unidirectional bulk data transfer over a single TCP connection with standard 1500-byte MTU. Here the DUT is acting as server (S) and the TM as client (C). From Fig. 5 it can be seen that Macvlan performs better than the Veth bridge for all DUTs and has a performance close to the native case. The lks7688 and rspi3 devices achieve 94 Mbps in both transmit and receive direction which is very close to the theoretical limit of 100 Mbps. l4500 and archc7 are not able to saturate the 1Gbps link. In case of the veth bridge, a lower throughput performance is observed for archc7 and l4500. The veth bridge is efficient in receive direction but shows a significant overhead in transmit direction for the arm based l4500 device. The evaluation of the system wide CPU utilization using the perf performance measurement tool shows CPU underutilization, an increased amount of branch misses and L1 cache access misses despite of having not imposed any restriction on the CPU of the TM or DUT.

The Netperf request response benchmark is used to measure the round-trip latency. For the benchmark, the TM is configured as client and the DUT as server. The client sends a 100-byte request and the server sends a 200-byte response and the client waits for the response before sending another request. The trading process of multiple TCP requests and responses occur in the same TCP connection. This pattern appears frequently in database applications. The round-trip latency is calculated using the following formula:

$$round\ trip\ latency = ((1/\text{Trans.rate}) * (1e6\ \mu s/1\ s))$$

where Trans.rate is the number of transactions (i.e. no. of request-response cycles) per second. Fig. 6 shows the measured round trip latency for both the TCP and UDP variants of the benchmark. As it can be seen, there is no significant round trip delay increase for both the veth and macvlan bridge compared to the non-virtualized case.

The iperf3 tool is used to measure the TCP throughput with the DUT acting as server (S) and TM as client (C). Fig. 7, 8, 9 and 10 show the aggregated throughput for all DUTs running multiple iperf3 instances. In all cases, Macvlan shows a throughput close to the native case.

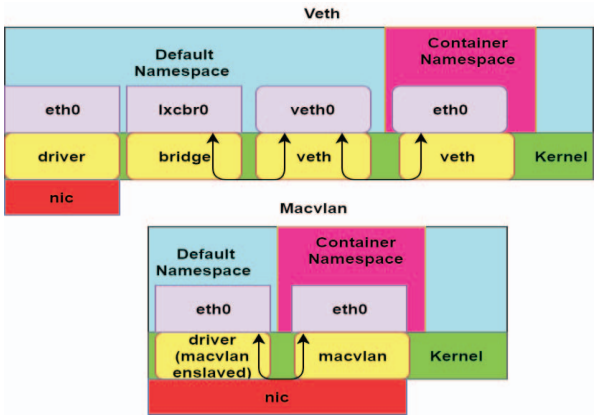

Fig. 4. Visualization of the veth and macvlan kernel module

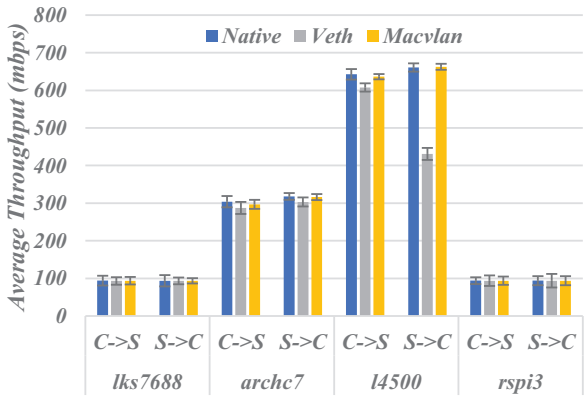

Fig. 5. Nuttcp results

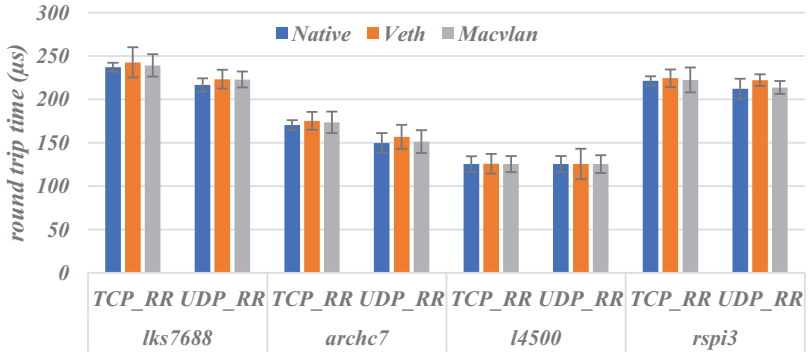

Fig. 6. Netperf results

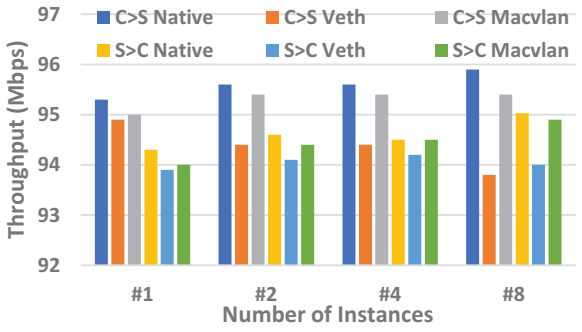

Fig. 7. lks7688 Iperf3 results

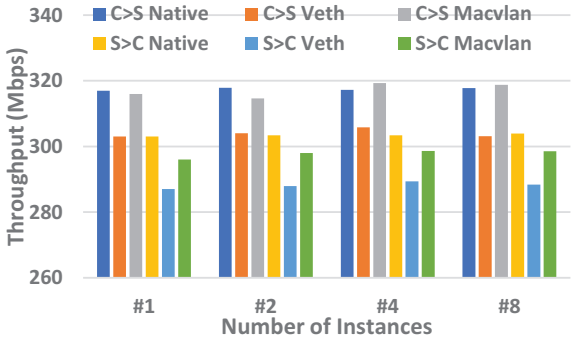

Fig. 8. archc7 Iperf3 results

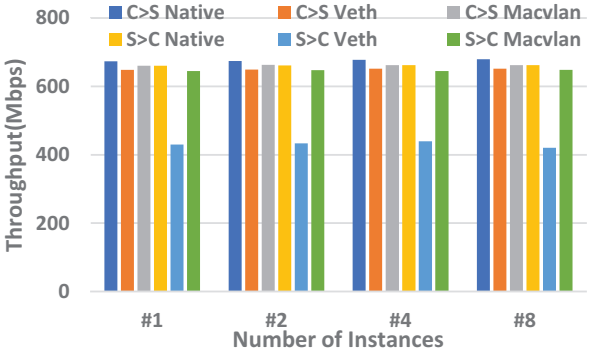

Fig. 9. L4500 Iperf3 results

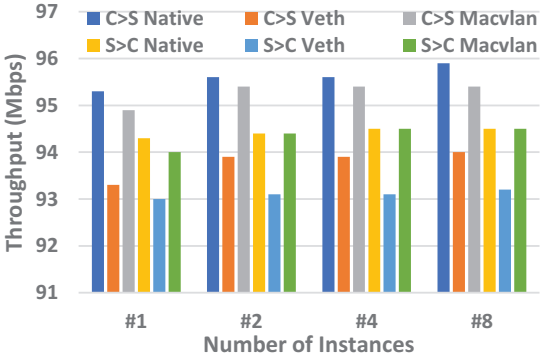

Fig. 10. rspi3 Iperf3 results

TABLE C.STREAM results

Benchmark	STREAM(MB/s)							
Type	Add		Copy		Scale		Triad	
Platform	Native	%change	Native	%change	Native	%change	Native	%change
lks7688	99.98	-0.1	208.43	-0.3	65.57	-0.1	49.65	-0.4
archc7	129.28	-0.4	272.85	-0.2	77.68	-0.3	60.08	-0.5
l4500	170.81	-0.7	752.95	-3.2	262.15	-0.5	156.24	-1
rspi3	1885.37	-0.8	2331.1	-4.5	2131.97	-2.2	1682.01	-0.1

TABLE D.STREAM2 results

Benchmark	STREAM2(MB/s)							
Type	Fill		Copy		Daxpy		Sum	
Platform	Native	%change	Native	%change	Native	%change	Native	%change
lks7688	177.94	-0.8	208.33	-0.6	61.82	-0.6	47.05	-2.0
archc7	285.39	-0.1	273.83	-0.1	69.79	-0.9	56.94	-0.7
l4500	760.8	-0.1	777.2	-3	172.2	-0.5	130.57	-0.1
rspi3	1288.74	-0.2	2339.9	-0.2	1656.04	-0.3	1698.19	-0.1

Similar to the Nuttcp results (Fig. 5) a throughput degradation in transmit direction is observed for the l4500 device when the veth bridge is used. The veth case shows about 7-15% degradation for l4500 and archc7 compared to the native case.

D. Memory Performance Evaluation

The STREAM[15] and STREAM2[16] tools within the lmbench micro-benchmark tool are used to measure the memory I/O performance. They measure the sustainable memory bandwidth when performing simple operations on vectors. The Stream array size is set to at least four times the available cache memory. From Table C and Table D, it can be observed that the differences compared to the native case are quite small in all situations. The worst-case degradation of 4.5% occurs when applying the Copy function on the rspi3. This is due to the use of a SD card as flash memory.

V. CONCLUSION AND FUTURE WORK

In this work, an extensive performance evaluation is carried out to assess the overhead caused by running multiple virtualized instances on a broad range of embedded microprocessors. The following conclusions can be drawn from the study:

- Container based virtualization has a negligible impact in terms of CPU and memory consumption compared to the native (non-virtualized) case.
- Using Macvlan as container network interface yields a comparable throughput and round-trip latency (for both UDP and TCP traffic) like the native ethernet interface.
- The Linkit Smart 7688 and Raspberry Pi 3B boards show a near zero virtualization overhead and therefore are well suited for IoT gateways that are executing lightweight applications.

In future we plan to investigate several other scenarios, e.g. the live resizing of containers, the tradeoff between live migration and redeployment of services, the feasibility of using various management and orchestration tools, dynamic chaining of services using containers.

[15] https://www.cs.virginia.edu/stream

[16] https://www.cs.virginia.edu/stream/stream2/

REMARK

The scripts and firmware images are available at https://gitlab.com/vivian.devops/benchmarktools.

REFERENCES

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, MCC ’12. ACM, 2012, pp. 13-16.

[2] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 171-172, March 2015.

[3] Á. Kovács, ,” Comparison of Different Linux Containers”, International Conference on Telecommunications and Signal Processing (TSP), pp. 47-51, July 2017.

[4] D. Beserra, M.K Pinheiro, C. Souveyet, L. Steffenel and E.D Moreno,” Comparing the performance of OS-level virtualization tools in SoC-based systems: The case of I/O-bound applications” in IEEE Symposium on Computers and Communications (ISCC), July 2017.

[5] Z. J. Estrada, Z. Stephens, C. Pham, Z. Kalbarczyk, and R. K. Iyer, “A performance evaluation of sequence alignment software in virtualized environments,” in 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 730–737, May 2014.

[6] M. G. Xavier, M. V. Neves, and C. A. F. D. Rose, “A performance comparison of container-based virtualization systems for mapreduce clusters,” in 22nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), pp. 299–306, Feb 2014.

[7] H. Chang, A. Hari, S. Mukherjee, and T. V. Lakshman, “Bringing the cloud to the edge," in Proc. IEEE Conf. Comput. Commun. Workshops (INFO- COM WKSHPS), pp. 346-351, May 2014.

[8] K. Alexandr, “Internet of Things gateways meet linux containers: Performance evaluation and discussion," in Proc. IEEE 2nd World Forum Internet Things (WF-IoT), pp. 222-227, 2015.

[9] R. Morabito, “Virtualization on Internet of Things Edge Devices With Container Technologies: A Performance Evaluation”, Access IEEE, vol. 5, pp. 8835-8850, May 2017.

[10] L. McVoy and C. Staelin, “Lmbench: Portable Tools for Performance Analysis” in Proceedings of the USENIX 1996 Annual Technical Conference, January 1996.

[11] C. Xu, Z. Zhao, H.Wang, R. Shea, and J. Liu, “Energy efficiency of cloud virtual machines: From traffic pattern and CPU affinity perspectives" in IEEE System Journal, pp. 835-845, 25 June 2015.

[12] J. Claassen, R. Koning and P. Grosso, “Linux containers networking: performance and scalability of kernel modules”, Network Operations and Management Symposium (NOMS), pp. 713-717, 25-29 April 2016.