Image-Processing Workloads and DDoS Attack Resilience: Evaluating Docker and Podman Containers on Raspberry Pi and ODROID

Eric Gamess
Jacksonville State University
Jacksonville, Alabama, USA
egamess@jsu.edu

Mausam Parajuli
Jacksonville State University
Jacksonville, Alabama, USA
mparajuli@stu.jsu.edu

ABSTRACT

Containerization has revolutionized software development and deployment. This study investigates the performance of Docker and Podman containers on various ARM-based Single-Board Computers (SBCs), including the ODROID-XU4, ODROID-N2+, and Raspberry Pi 4 Model B (RPi 4B), for image processing tasks, in addition to exploring the impact of Distributed Denial of Service (DDoS) attacks on these container technologies. The research assesses parameters such as time, CPU utilization, and memory utilization during image processing tasks. Moreover, the paper inspects the resilience of web servers under DDoS attacks when varying the retrieved file sizes, frequency of malignant traffic generated by the attackers, and web server choices, emphasizing the need to implement adaptive security measures. The ODROID-N2+ with 4 GB memory emerged as a standout performer, navigating the challenges posed by DDoS attacks while maintaining acceptable speeds, suggesting its potential as a cost-effective Docker and Podman engine. At the level of the container technologies, Docker and Podman performed similarly. Overall, the insights gained from this research offer valuable guidance for practitioners who are planning containerized deployments on resource-constrained devices.

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; • **Networks** → **Network reliability**.

KEYWORDS

Docker, Podman, Containerization, DDoS, Raspberry Pi, ODROID, Single-Board Computers, Performance Evaluation, Benchmark

ACM Reference Format:
Eric Gamess and Mausam Parajuli. 2024. Image-Processing Workloads and DDoS Attack Resilience: Evaluating Docker and Podman Containers on Raspberry Pi and ODROID. In *2024 ACM Southeast Conference (ACMSE 2024), April 18–20, 2024, Marietta, GA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3603287.3651219

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ACMSE 2024, April 18–20, 2024, Marietta, GA, USA
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0237-2/24/04…$15.00
https://doi.org/10.1145/3603287.3651219

1 INTRODUCTION

In the ever-evolving domain of modern software development and deployment, containerization has emerged as a transformative technology. It provides a lightweight and efficient means of packaging, distributing, and executing applications, offering portability and scalability. Containerization bumbles applications and their dependencies into isolated units or containers, ensuring consistent execution across various environments.

This study explores the performance of Docker [17, 20] and Podman [3, 21] engines running on ARM-based Single-Board Computers (SBCs), including ODROID-XU4 [11], ODROID-N2+ [9, 10], and Raspberry Pi 4 Model B [18] (RPi 4B). Docker has attained a de facto standard status among leading containerization platforms, while Podman emerges as a daemonless alternative emphasizing security and compatibility. The applications and limitations of Docker and Podman, highlighting their respective strengths in image processing tasks and their resilience under Distributed Denial of Service (DDoS) attacks, are examined in this paper.

Image processing, a fundamental computational task, encompasses manipulating and analyzing images for various applications. It involves a wide range of techniques and algorithms for enhancing, restoring, transforming, or extracting information from images, including rotation, blur, noise reduction, and shape analysis. This paper evaluates the performance of Docker and Podman containers in image rotation tasks with `ImageMagick`, a powerful open-source application for image manipulation. The authors chose this software as one of the benchmarking tools due to its versatility and widespread use in image-processing workflows.

DDoS attacks bring a significant threat to networked systems, aiming to flood servers with overwhelming traffic to deplete their resources and disrupt services. This paper delves into the intricacies of DDoS attacks, highlighting the importance of securing containerized servers. In the DDoS experiments carried out for this paper, `hping3` (a versatile security tool) was employed to simulate the illegitimate traffic with varying frequency, providing insights into performance and CPU/memory utilization under adversarial conditions.

The study shows that the ODROID-N2+ outperformed the other two tested SBCs, at the price of a higher memory consumption. Regarding the container technologies, the differences between Docker and Podman were insignificant. Other valuable suggestions are provided in this study to guide practitioners in navigating the deployment of containerized services in resource-constrained devices.

The structure of the rest of this paper is as follows: Section 2 reviews relevant literature in this research area. Section 3 details the characteristics of the chosen SBCs and the testbed. Sections 4 and 5

explore the performance of image processing tasks in containerized environments, when varying the number of containers and the size of the images, respectively. Network assessments (WiFi vs Ethernet) are presented in Section 6. Section 7 discusses the response of containerized web servers under various DDoS attack conditions. Finally, Section 8 concludes the paper and suggests future avenues for research in this area.

2 RELATED WORK

On the one hand, there have been a lot of efforts to evaluate containerization technologies empirically, but essentially on testbeds that use conventional computers based on the x86/x64 architecture [5, 14, 15, 22] (e.g., Intel and AMD) or VMs running in the cloud [6].

On the other hand, many research groups have benchmarked applications that run natively on SBCs. As an illustration, Damasceno, Dantas, Araujo [4] evaluated the transmission performance of two routers at the edge network on the end-user side: (1) a Cisco 1905 and (2) an RPi 4B with OpenWrt. The authors concluded that the SBC-based solution achieved similar performance to the commercial one, but at a much lower price. Istifanos and Tekahun [13] conducted a performance assessment of web servers (NGINX and Apache) on an RPi 3B and a standard HP laptop. They reported performance parameters such as CPU usage, response time, and number of requests served per second.

According to the search of the specialized literature done by the authors of this work, just a few efforts have been made in the area of assessing the performance of container engines on SBCs. In [7], the authors evaluated Docker on the RPi Zero W, RPi Zero 2 W, RPi 3B, RPi 3B+, and RPi 4B. They did a timed computation of π, and benchmarked web and PostgreSQL servers, all running inside containers. Morabito [16] contrasted native and containerized performance on an RPi 2B, using tools such as `Sysbench`, `mbw` (Memory Bandwidth Benchmark), `Iperf`, and `ab` (Apache HTTP server benchmarking tool). The research team of this paper did not find any other study with these characteristics. Unlike the present work, the authors of [7, 16] considered just one container technology (Docker) and focused on SBCs of the Raspberry Pi Foundation. Hence, the present work brings practical recommendations to practitioners planning to deploy containerized services, regarding the selection of Docker or Podman, and the expected performance of commonly-used SBCs.

3 TESTBED FOR THE EXPERIMENTS

Single-Board Computers (SBCs) are complete and compact computers with all necessary components, including processor, memory, storage, and connectivity, on a single circuit board. Their application ranges from the Internet of Things (IoT) and embedded systems to educational projects. These devices are widely employed due to their small form factor, energy efficiency, affordability, and versatility. Table 1 details the key specifications for the SBCs selected for this research. The ODROID-XU4 [11], ODROID-N2+ [9, 10], and Raspberry Pi 4 Model B [18] (RPi 4B) were selected considering their popularity and strong community support. It is worth clarifying that the Raspberry Pi 5 was not considered in this study since it was just recently released and is affected by the supply chain backlog.

There are various storage options for these SBCs including microSD cards, thumb drives, and SSDs, where the latter two require a USB connection. It is noteworthy to mention that the ODROID SBCs also offer the option of eMMC storage. The experiments in this paper utilized microSD cards, specifically the 64 GB SanDisk Extreme PRO microSDXC UHS-I memory cards. Additionally, to manage the heat generation of the RPi 4B, the tested SBCs were enclosed in cases equipped with small fans. That was unnecessary for the ODROID-XU4 and ODROID-N2+, since they natively have a small fan and a metal-housing heat sink, respectively.

Regarding the operating system, the last version of Armbian [2] was installed in the ODROID SBCs (Armbian armhf v23.11 in the ODROID-XU4 and Armbian aarch64 v23.11 in the ODROID-N2+). They are CLI versions of Debian 12.2 (Bookworm). For the RPi 4B, the Raspberry Pi OS [19] was selected since it is the most popular OS for this SBC. The latest 64-bit light version, released in October 2023, was chosen.

The testbed shown in Figure 1 was used for the `ImageMagick` experiments. It consisted of one test computer that controlled the operations performed on the images and one SBC (ODROID-XU4, ODROID-N2+, or RPi 4B) as the container host. The computer was a Dell OptiPlex 7090 SFF, with an octa-core Intel i7-10700 CPU @ 2.90 GHz, 32 GB DDR4 RAM, 512 GB PCIe NVMe SSD, 10/100/1000 Mbps Ethernet port, and Intel AX201 WiFi NIC (IEEE 802.11a/b/g/n/ac/ax). The last version of Debian amd64 (version 12.4) was installed on the computer. The devices of the testbed were interconnected through a 2.4 GHz WiFi network. A NETGEAR AC1750 Smart WiFi Router R6400v2 (IEEE 802.11a/b/g/n/ac) was used as the WLAN router. At the level of the ODROID-XU4 and ODROID-N2+, a Linksys WUSB6300 WiFi NIC was connected through a USB 3.0 port, since they only have Ethernet connectivity by default (see Table 1).

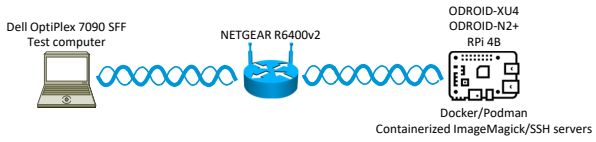

Figure 1: Testbed for the ImageMagick Experiments

For the DDoS attack experiments, the testbed depicted in Figure 2 was used. It consisted of up to six attacker computers, one computer to generate the legitimate petitions, and one SBC (ODROID-N2+ or RPi 4B) to run the containers. All the computers were Dell OptiPlex 7090 SFF, with the same specifications as those of the computer used in the `ImageMagick` experiments. The interconnection of the devices was done through either Ethernet or WiFi. For the wired scenario, a Cisco SG355-10P Ethernet switch (eight 10/100/1000 PoE+ ports) was used. For the wireless scenario, the Ethernet switch was replaced by the NETGEAR AC1750 Smart WiFi Router R6400v2.

4 IMAGE PROCESSING: VARIATION OF THE NUMBER OF DOCKER CONTAINERS

The primary objective of this experiment was to evaluate the performance of image manipulation within Docker containers running on the SBCs under test, when varying the number of containers.

Table 1: Specifications of the Chosen SBCs

	ODROID-XU4	ODROID-N2+	RPi 4B
SoC Type	Samsung Exynos5422	Amlogic S922X	Broadcom BCM2711
Core Type	Quad-core ARM Cortex-A15 @ 2.0 GHz Quad-core ARM Cortex-A7 @ 1.4 GHz	Quad-core ARM Cortex-A73 @ 2.4 GHz Dual-core ARM Cortex-A53 @ 2.0 GHz	Quad-core ARM Cortex-A72 @ 1.8 GHz
Core Architecture	ARMv7-A (32-bit)	ARMv8-A (64-bit)	ARMv8-A (64-bit)
RAM	2 GB LPDDR3	2 or 4 GB DDR4	1, 2, 4, or 8 GB LPDDR4
Storage Type	microSD, eMMC	microSD, eMMC	microSD
Ethernet	10/100/1000 Mbps	10/100/1000 Mbps	10/100/1000 Mbps
WiFi	Not built-in (requires USB adapter)	Not built-in (requires USB adapter)	Dual-band IEEE 802.11b/g/n/ac
HDMI Port	1 x full-size HDMI 1.4a	1 x full-size HDMI 2.0	2 x micro HDMI (supporting up to 4K)
USB Ports	2 x USB 3.0, 1 x USB 2.0	4 x USB 3.0, 1 x Micro USB 2.0 OTG	2 x USB 3.0, 2 x USB 2.0
Price	US$59	US$66 or US$83	US$35, US$45, US$55, or US$75

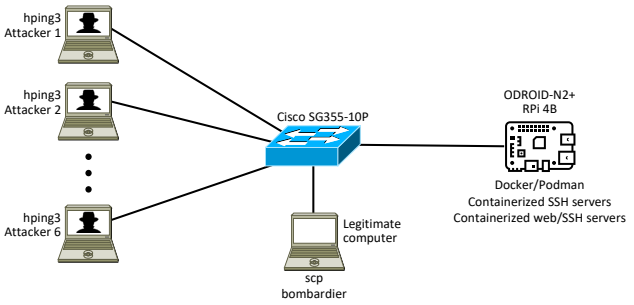

Figure 2: Testbed for the DDoS Attack Experiments (Ethernet Version)

```
01: FROM debian
02: RUN apt-get update
03: RUN apt-get install --yes imagemagick openssh-server
04: RUN echo "PermitRootLogin yes" >> /etc/ssh/sshd_config
05: RUN echo "root:<password-root>" | chpasswd
06: CMD sh -c "/etc/init.d/ssh start; /bin/bash"
07: EXPOSE 22
```

Figure 3: Dockerfile to Create the ImageMagick/SSH Server Image

```
01: #!/bin/bash
02: numContainer=$1
03: basePort=$2
04: for i in $(seq 0 $((numContainer-1))); do
05:     docker container run --detach --tty --name $(printf "d%03d" $i) \
          --hostname $(printf "d%03d" $i) \
          --publish $(printf "%03d" $((basePort+i))):22 debian-imagemagick
06: done
07: exit 0
```

Figure 4: Script to Run ImageMagick/SSH Docker Containers

The experiment was carried out in the testbed of Figure 1. It involved concurrent rotations of a 200-KB image in the containerized ImageMagic/SSH servers that were running on the SBC. All the SBC models available for the work were tested: ODROID-XU4, ODROID-N2+ (2 and 4 GB), and RPi 4B (1, 2, 4, and 8 GB).

A "Dockerfile", shown in Figure 3, was developed to create a customized image for the containers running on the SBCs. The line numbers were added to help reference them. Line 01 specifies the base image to be used for the creation of the customized image, which in this case is the latest Debian image (version 12.4) from Docker Hub. Line 02 updates the package list inside the customized image, while Line 03 installs two packages: ImageMagick and the OpenSSH server. ImageMagick was used as an image-processing benchmarking tool, and the OpenSSH server was installed for securing the remote access. Lines 04-05 enable the usage of the SSH services for the root user and set the password. Line 06 specifies the default command to be executed when a container is instantiated, while Line 07 exposes the SSH port.

Figure 4 depicts the script that was developed to run the Docker containers. As shown in Lines 02-03, it is based on two arguments: the number of containers to be instantiated and the base port for the OpenSSH servers. In each iteration of the loop of Lines 04-06, a container is created and started in the background with a specified name and hostname, publishing incrementally assigned ports for the OpenSSH servers.

The test was carried out as specified as follows: For each containerized ImageMagick/SSH server that was running on the SBC, a process was started on the test computer by executing the script client-magick-script.sh (see Figure 5), forming a one-to-one association between a client process and a container, where each client controlled the rotation of a specified image several times within its associated container, thereby enabling several parallel image transformations simultaneously.

In Line 08 of Figure 5, the script client-magick-script.sh starts by copying the input image (specified by inputImage) from a client process to its associated Docker container running on the SBC, using the scp utility. scp is a command-line utility that allows the secure copying of files and directories between two locations on a network, using the SSH protocol. Line 09 creates an initially empty string (cmdStr) that should contain all the commands to be executed in the containerized ImageMagick/SSH server. In each iteration of the loop of Lines 10-12, a new command is appended to cmdStr and consists of rotating the input image by 90 degrees, using the convert utility of ImageMagick. Line 13 uses SSH to open a connection with the container and execute the commands stored in cmdStr. After performing all the rotations in the associated containerized ImageMagick/SSH server, Line 14 retrieves the resulting image with scp. In Line 16, the transfer and processing

time (total time for the whole process) is calculated by subtracting two timestamps recorded at the beginning (Line 07) and end (Line 15) of the test.

During the assessment, in addition to recording the transfer and processing time, the authors also monitored the CPU and total memory usages with `mpstat` and `free` on the container host (SBC). `mpstat` (Multiple Processor Statistics) is a command-line utility that provides detailed information about the utilization of processors. On the other hand, the `free` command is used to obtain information about the system's memory usage.

```
01: #!/bin/bash
02: sbcIPAddress=$1
03: portSSH=$2
04: inputImage=$3
05: rotationCount=$4
06: myID=$5
07: start_time=$(date +%s%N)
08: scp -q -P $portSSH $inputImage root@$sbcIPAddress:.
09: cmdStr=""
10: for((i=0; i<rotationCount; i++)); do
11:     cmdStr=$cmdStr" convert /root/$inputImage \
            -rotate 90 /root/outputImage.png;"
12: done
13: ssh -p $portSSH root@$sbcIPAddress "$(echo $cmdStr)"
14: scp -q -P $portSSH root@$sbcIPAddress:/root/outputImage.png \
        outputImage$(printf %03d $myID).png
15: end_time=$(date +%s%N)
16: time_taken=$(echo "scale=2; ($end_time - $start_time)/1000000000" | bc)
17: # Save the result in the container host
18: exit 0
```

Figure 5: Script to Assess a Sequence of Rotations Using ImageMagick (`client-magick-script.sh`)

For the experiment, the number of rotations was set to 4 (controlled by `rotationCount` in `client-magick-script.sh`). The number of containerized `ImageMagick`/SSH servers was varied: 1, 2, 4, 8, 16, 32, 64, 128, and 256 containers. The results for the transfer and processing time, the CPU usage, and the total memory usage in the SBCs under test are shown in Tables 2, 3, and 4, respectively. An `N/A` in the tables stands for "Not Applicable", and means that the associated experiment failed due to resource exhaustion. In Table 4, the columns labeled "`Prestart`" correspond to the total memory utilization in the SBCs after creating the containers, but before starting them. In this way, it is possible to appreciate the additional amount of memory required by the execution of the benchmark itself (i.e., the execution of the script `client-magick-script.sh` that performs rotations). An analysis of the results is done below.

Analysis of the Transfer and Processing Time

- The transfer and processing time degraded as the number of containers increased across all the SBCs under test. That is, as more containers were instantiated, the overall workload increased, leading to resource contention and a subsequent longer transfer and processing time.
- All the SBCs under test showed a similar transfer and processing time while increasing the number of containers, up to their number of cores. For example, the ODROID-XU4 demonstrated a transfer and processing time between 2.06 and 2.82 seconds, when going from 1 to 8 containers. That is, almost the same value. However, when passing from 8 to 16 containers, the transfer and processing time went from 2.82 to 5.46 seconds (almost twice the time). The reason may be the following. For up to 8 containers, each container was assigned to its own core. With 16 containers, each core had to take care of 2 containers, doubling the value of the transfer and processing time. Similar results can be seen for the ODROID-N2+ (6 cores) and the RPi 4B (4 cores).
- For the same model of SBC, having more memory increased its performance, as more memory allows for better handling of concurrent tasks. This is clearly demonstrated for the two versions of the ODROID-N2+ (2 and 4 GB) and the four versions of the RPi 4B (1, 2, 4, and 8 GB).
- The number of cores was important in this experiment. The ODROID-N2+ excelled as the fastest, benefiting from six cores and robust hardware with ample CPU power and memory. In contrast, the RPi 4B lagged behind due to having just four cores, resulting in a higher number of containers assigned to each core.

Analysis of the CPU Usage

- The CPU usage soared rapidly with the growing number of containers, up to its limit (100%).
- The maximum usage of the CPU (100%) is reached as soon as the number of containers is equal to or surpasses the number of cores for the reasons explained in the previous analysis. For example, the ODROID-XU4 with eight cores reached 99.75% of CPU usage with eight containers. The RPi 4B with four cores reached more than 96.49% of CPU usage with four containers.
- The chosen benchmark (rotations of images) is clearly CPU intensive, since the maximum usage of the CPU is reached as soon as the number of containers is equal to or exceeds the number of cores.

Analysis of the Total Memory Usage

- Across all SBCs under test, an evident increase in memory consumption is observed as the number of containers soars.
- For an equivalent setting, the ODROID-N2+ showed the highest total memory usage, especially the 4 GB configuration.
- Having more memory led to more scalability. For example, the RPi 4B with 1, 2, 4, and 8 GB RAM could handle up to 32, 64, 128, and 256 containers, respectively. The ODROID-N2+ with 2 and 4 GB RAM could manage up to 64 and 128 containers, respectively. This highlights the importance of choosing SBCs with sufficient memory for tasks involving containerization and scalability.

Based on these findings, this research team continued the rest of the experiments with the best-performing devices for each model: ODROID-XU4, ODROID-N2+ (4 GB), and RPi 4B (8 GB). This strategic choice ensures a smart and effective continuation of the study since most practitioners who are planning to implement a container host will probably choose the version of the SBC that has more memory.

Table 2: Transfer and Processing Time in Seconds

Number of Containers	OD-XU4	OD-N2+		RPi 4B			
	2 GB	2 GB	4 GB	1 GB	2 GB	4 GB	8 GB
1	2.06	1.48	1.45	2.04	1.98	1.96	1.91
2	2.09	1.51	1.49	2.24	2.07	1.99	1.97
4	2.33	1.72	1.54	2.47	2.45	2.31	2.19
8	2.82	2.64	2.24	4.24	4.21	4.15	4.09
16	5.46	4.49	4.24	8.46	8.43	8.17	8.17
32	10.90	8.93	8.19	17.14	16.63	16.39	16.18
64	22.39	17.81	15.95	N/A	33.72	32.38	32.21
128	N/A	N/A	32.36	N/A	N/A	66.71	65.16
256	N/A	N/A	N/A	N/A	N/A	N/A	126.67

Table 3: CPU Usage in Percents

Number of Containers	OD-XU4	OD-N2+		RPi 4B			
	2 GB	2 GB	4 GB	1 GB	2 GB	4 GB	8 GB
1	13.32	13.92	13.68	23.72	25.83	25.77	24.58
2	25.88	26.96	26.45	46.60	50.71	48.95	47.06
4	50.48	57.12	55.39	96.49	99.33	98.92	98.54
8	99.75	98.61	97.77	100.00	100.00	100.00	100.00
16	100.00	100.00	100.00	100.00	100.00	100.00	100.00
32	100.00	100.00	100.00	100.00	100.00	100.00	100.00
64	100.00	100.00	100.00	N/A	100.00	100.00	100.00
128	N/A	N/A	100.00	N/A	N/A	100.00	100.00
256	N/A	N/A	N/A	N/A	N/A	N/A	100.00

Table 4: Total Memory Usage in MB

Number of Containers	ODROID-XU4		ODROID-N2+				Raspberry Pi 4 Model B							
	2 GB		2 GB		4 GB		1 GB		2 GB		4 GB		8 GB	
	Prestart	Docker	Prestart	Docker	Prestart	Docker	Prestart	Docker	Prestart	Docker	Prestart	Docker	Prestart	Docker
1	281	293	321	345	358	370	230	242	219	230	224	234	285	298
2	289	313	352	384	386	410	244	263	230	256	235	257	297	323
4	309	356	396	454	421	477	267	316	251	302	252	302	318	369
8	344	438	484	590	496	605	301	402	295	395	301	398	357	456
16	409	600	630	841	651	861	379	576	369	575	374	568	430	630
32	536	928	937	1352	976	1390	552	820	516	926	524	920	572	972
64	806	1528	1548	1766	1544	2365	N/A	N/A	820	1573	819	1620	868	1666
128	N/A	N/A	N/A	N/A	2806	3625	N/A	N/A	N/A	N/A	1403	2988	1450	3043
256	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	2670	5774

5 IMAGE PROCESSING: VARIATION OF THE SIZE OF THE PROCESSED IMAGES

The experiment discussed in Section 4 was extended to include different sizes of the transferred image (200, 500, and 1000 KB) for 1, 8, and 16 containers. Additionally, Podman was incorporated alongside Docker in this experiment. The transfer and processing time for the experiment was recorded as well as the total memory utilization for the SBCs under test. It is important to note that the scripts done for Docker (e.g., the script of Figure 4) were adjusted for Podman. In most cases, simply replacing the command `docker` with the command `podman` worked fine as they share a similar command-line interface.

The results for the transfer and processing time (in seconds) are depicted in Figures 6, 7, and 8 for a 200-KB, 500-KB, and 1000-KB image size, respectively. Similarly, the corresponding total memory usage (in MB) is illustrated in Figures 9, 10, and 11. The main findings and their analysis are discussed subsequently.

Analysis of the Transfer and Processing Time

- As the file size increased, the transfer and processing time escalated. Larger images contain more data, requiring more transmission and processing time, aligning with expectations.
- The observed increase in transfer and processing time with a higher number of containers suggests potential resource contention. As multiple containers compete for resources, the overall performance is affected, leading to longer experiment times. This finding highlights the importance of resource management and allocation in a multi-container environment.
- The ODROID-N2+ consistently outperformed the other two tested devices. This can be attributed to superior hardware capabilities, including the SoC and memory.
- The RPi 4B consistently exhibited the longest transfer and processing time, especially as the number of containers increased. This is probably due to fewer cores in its SoC, resulting in poorer multi-container management.
- The similarity in performance between Docker and Podman suggests that for this specific experiment, the choice between these two container technologies had a marginal impact.

Analysis of the Total Memory Usage

- The observed increase in the total memory usage with both larger file sizes and a higher number of containers underscores the demands on this resource during the test.
- The ODROID-N2+ consumed more memory resources than the other two SBCs under test. As mentioned in the previous analysis, this higher price came with the benefit of a faster test.
- Podman consumed less memory than Docker across the different scenarios. This could be due to differences in their architecture or resource handling. Podman is known for its lightweight design and the absence of a daemon, which

might contribute to a lower total memory usage compared to Docker.

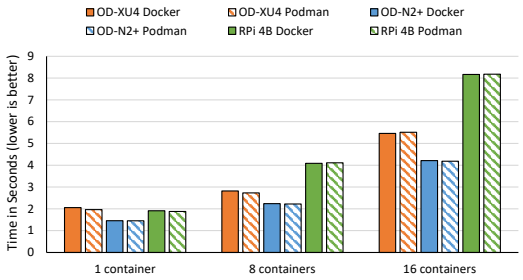

Figure 6: Transfer and Processing Time for a 200-KB Image

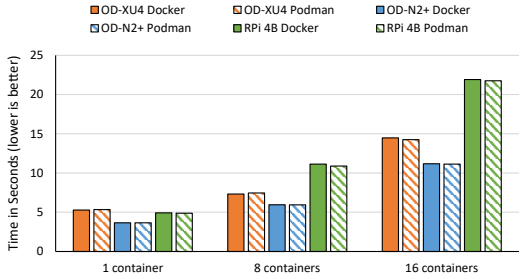

Figure 7: Transfer and Processing Time for a 500-KB Image

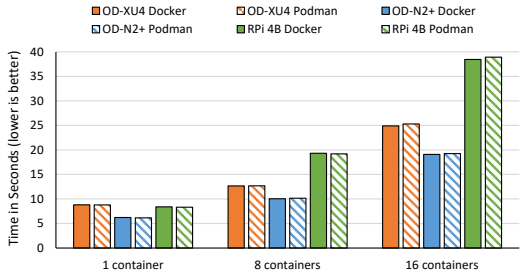

Figure 8: Transfer and Processing Time for a 1000-KB Image

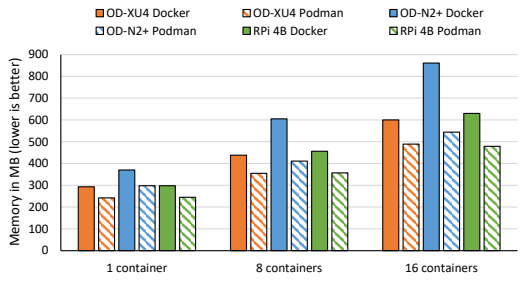

Figure 9: Total Memory Usage for a 200-KB Image

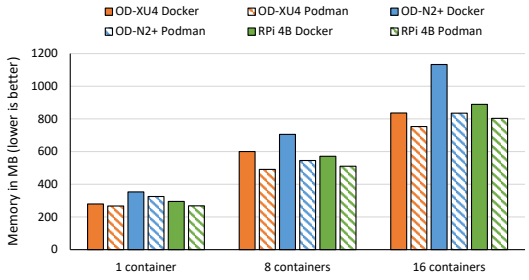

Figure 10: Total Memory Usage for a 500-KB Image

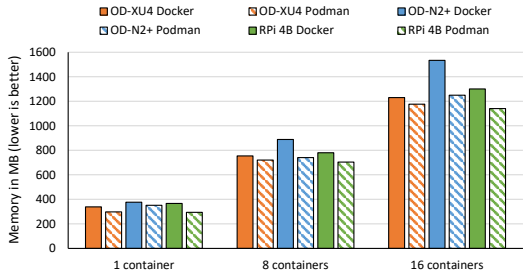

Figure 11: Total Memory Usage for a 1000-KB Image

6 IMPORTANCE OF THE NETWORK TECHNOLOGY IN DISTRIBUTED DENIAL OF SERVICE ATTACKS

Several technologies (WiFi and Ethernet) may be used to connect the container host (SBC) to the network. This section studies the importance of the network technology in Distributed Denial of Service (DDoS) attacks on containers. To do so, an experiment was carried out. It consisted of analyzing the differences between a 2.4 GHz WiFi network and an Ethernet network when transferring files with SSH, while a DDoS attack was taking place.

The experiment involved concurrent transfers of a 10,000-byte file from the legitimate computer to containerized SSH servers running on the SBC, while simultaneously carrying out a DDoS attack using "`hping3`" from the attacker computers (see Figure 2).

A customized image was created and built on the SBC using the "`Dockerfile`" shown in Figure 3 (without `ImageMagick`). A basic script named `client-scp-script.sh` (see Figure 12) was also developed to assess the transfer of a binary file from the legitimate computer to a remote container running on the SBC through `scp` (Secure Copy). For a better evaluation, the transfer was repeated several times (controlled by `transferCount`) and the average transfer time was reported. Lines 02-06 accept five command-line parameters: the SBC's IP address, the published port number of the SSH server, the name of the binary file to be transferred, the number of transfers to be performed, and a process identifier to retrieve the results. Line 07 obtains a timestamp before the beginning of the transfers. In the loop of Lines 08-10, the `scp` command is used to copy the binary file securely into the specified containerized SSH server using the provided published port number. Once all the transfers are done, a second timestamp is captured in Line 11, and the average transfer time is computed in Line 12.

```
01: #!/bin/bash
02: sbcIPAddress=$1
03: portSSH=$2
04: binFile=$3
05: transferCount=$4
06: myID=$5
07: start_time=$(date +%s%N)
08: for((i=0; i<transferCount; i++)); do
09:     scp -q -P $portSSH $binFile \
          root@$sbcIPAddress:outputFile$(printf %03d $i).bin
10: done
11: end_time=$(date +%s%N)
12: avg_time=$(echo "scale=2; ($end_time - $start_time) \
       /1000000000/$transferCount" | bc)
13: # Save the result in the container host
14: exit 0
```

Figure 12: Script to Assess a File Transfer Using `scp` (`client-scp-script.sh`)

The legitimate transfer traffic was created as specified: for each containerized SSH server that was in execution in the SBC, a process was started in the legitimate computer by executing the script `client-scp-script.sh` (see Figure 12), forming a one-to-one association between a client process and a container, where each client copied the specified file several times (controlled by `transferCount`) into its associated container, resulting in several parallel transfers at a time.

To start a DDoS attack against the containerized SSH servers using the `hping3` tool before the transfers of the file, a script named `start-attack-ssh.sh` was written (see Figure 13). The script takes five command-line parameters: the SBC's IP address, the number of containers to be attacked, the base published port number for the containers, the `hping3` attack frequency, and the attack duration. The loop of Lines 07-09 creates several instances of `hping3` so that the process that executes this script will be flooding all the containerized SSH servers. This is a TCP `SYN` flood attack on the SSH port of the containers as stated by option "`-S`".

```
01: #!/bin/bash
02: sbcIPAddress=$1
03: numContainer=$2
04: basePort=$3
05: hping3Freq=$4
06: duration=$5
07: for((i=0; i<numContainer; i++)); do
08:     hping3 -q -S -p $((basePort+i)) -V \
          -i u$(echo "1.0/$hping3Freq*1000000" | bc -l) \
          -c $((hping3Freq*duration)) $sbcIPAddress &
09: done
10: exit 0
```

Figure 13: Script to Perform a DDoS Attack Using `hping3` (`start-attack-ssh.sh`)

The script `start-attack-ssh.sh` (see Figure 13) was executed in each attacker computer (see Figure 2) at the same time, with the same parameters. That is, when having three attackers, each containerized SSH server received malicious `SYN` segments from the three attackers. When having six attackers, each containerized SSH server received malicious `SYN` segments from the six attackers.

The experiment was conducted with the ODROID-N2+ (4 GB RAM), using the Docker technology, with four containerized SSH servers, for both 2.4 GHz WiFi and Ethernet networks, when the numbers of attackers varied (0 to 6). The `hping3` attack frequency was set to 500 (i.e., each attacker sent 500 `SYN` segments to each containerized SSH server, every second), and the duration of the attack was determined in such a way that it ended after the transfers of the file.

Results and Analysis

Figure 14 shows the average time in seconds to transfer a 10,000-byte file for 2.4 GHz WiFi and Ethernet. The results were reported for several numbers of attackers, starting with zero (no attackers) up to six. The findings are discussed below:

- The consistently higher transfer times for WiFi, especially as the number of attackers increased, highlight the inherent limitations of WiFi compared to Ethernet (i.e., a lower bandwidth and just one transmission at a time).
- For this experiment, Ethernet exhibited consistent transfer times regardless of the number of attackers, indicating the `hping3` attack frequency was too low to carry out a DDoS attack with Ethernet.

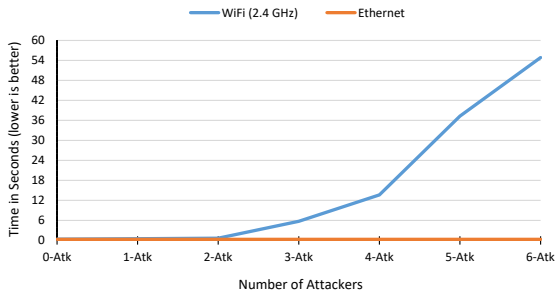

Figure 14: Average Transfer Time for the ODROID-N2+ with a 10,000-Byte File During a DDoS Attack (Docker)

This study underscores that the network capability was the limiting factor in the DDoS attack with the 2.4 GHz WiFi network scenario. That is, the resources of the ODROID-N2+ were not exhausted at any point, limiting the viability of doing additional experiments to try to overload the CPU or deplete the RAM of this SBC, when using this network technology. Therefore, all the subsequent DDoS experiments were done with the Ethernet network.

7 DISTRIBUTED DENIAL OF SERVICE ATTACKS ON CONTAINERIZED WEB SERVERS

This section details the experimental setup and methodology employed to investigate the performance and resilience of containerized web servers under a DDoS attack. The testbed was based on Figure 2, using Ethernet (as recommended in Section 6) and with a maximum of three attacker computers. The legitimate computer generated the normal/authorized HTTP petitions, and the containerized web/SSH servers were running in the SBC (ODROID-N2+ with 4 GB RAM or RPi 4B with 8 GB RAM). To create a consistent environment for the containerized web/SSH servers, the "`Dockerfile`"

shown in Figure 15 was developed. This file specifies the instructions for building a new container image based on the latest Apache web server [12] (version 2.4.58 in this case). Four different HTML test files of varying sizes (1024, 4096, 8192, and 16384 bytes) and a customized configuration file (`httpd.conf`) were copied into the image. The "`httpd.conf`" file was adjusted to access the containers' logs effectively. The default port for web servers was exposed.

```
01: FROM httpd
02: COPY test1024.html /usr/local/apache2/htdocs
03: COPY test4096.html /usr/local/apache2/htdocs
04: COPY test8192.html /usr/local/apache2/htdocs
05: COPY test16384.html /usr/local/apache2/htdocs
06: COPY httpd.conf /usr/local/apache2/conf
07: EXPOSE 80
```

Figure 15: Dockerfile to Create the Web/SSH Server Image

Figure 16 depicts the deployment script responsible for instantiating multiple Docker containers based on the previously created customized image, from parameters such as the number of containers and a base port. The script assigns container names, hostnames, and domain names to each instance, publishing incrementally assigned ports for the web servers. It is worth mentioning that a similar script was written for Podman.

```
01: #!/bin/bash
02: numContainer=$2
03: basePort=$3
04: for((i=0; i<numContainer; i++)); do
05:     docker container run -d --name $(printf "a%02d" $i) \
           -h $(printf "a%02d" $i) --domainname jsu.edu \
           -p $(printf "%02d" $((basePort+i))):80 apache2-img \
           /usr/local/apache2/bin/httpd -D FOREGROUND
06: done
07: exit 0
```

Figure 16: Script to Run Web/SSH Docker Containers (Apache)

To generate the legitimate HTTP traffic and simulate a DDoS attack on the containers, the script of Figure 17 was written. Initially (see Lines 11-15), several instances of `bombardier` [1] (a high-performance HTTP benchmarking tool) were started to generate legitimate HTTP petitions toward the web servers. A one-to-one correspondence existed between the instances of `bombardier` and the web servers. Each `bombardier` process sent concurrent HTTP requests (controlled by `bombardNumConcurrent`) to its associated web server for the duration of the experiment (specified by `bombardTime`). Subsequently, the script orchestrated DDoS attacks against the web servers in the loop of Lines 16-21. As shown, in each iteration of the loop, a `hping3`-based malignant process is periodically started on one of the attacker computers (the period is controlled by `atkGapTime`).

7.1 Variation of the `hping3` Attack Frequency

In this experiment, the `hping3` attack frequency was varied. The experiment was conducted using the parameters shown in Table 5.

Figures 18, 19, and 20 present the results obtained from the experiment. The outcome showcases the average HTTP throughput

```
01: #!/bin/bash
02: sbcIPAddress=$1
03: numContainer=$2
04: basePort=$3
05: fileName=$4
06: bombardNumConcurrent=$5
07: bombardTime=$6
08: atkGapTime=$7
09: hping3Freq=$8
10: numAttacker=$(($#-8))
11: lstPid=()
12: for((i=0; i<numContainer; i++)); do
13:     ./bombardier-linux-amd64 -c $bombardNumConcurrent -d ${bombardTime}s \
          http://${sbcIPAddress}:$((basePort+i))/${fileName} &
14:     lstPid+=($!)
15: done
16: for((i=0; i<numAttacker; i++)); do
17:   sleep $atkGapTime
18:   index=$((9+i))
19:   atkIPAddr=${!index}
20:   ssh root@$atkIPAddr "bash /root/start-client-hping3.sh $sbcIPAddress \
        $numContainer $basePort $hping3Freq" &
21: done
22: wait ${lstPid[@]}
23: exit 0
```

Figure 17: Script for Bombardment and DDoS Attack

Table 5: Parameters for the Web Experiments

Number of Containerized Web/SSH Servers:	4
Retrieved File Size:	4096 bytes
Bombardier Concurrent Connections:	10
Bombardier Runtime:	45 seconds
Number of Attackers:	3 (maximum)
Attack Gap Time:	10 seconds
hping3 Attack Frequency:	1200, 1400, and 1600

in requests per second (i.e., successful HTTP requests achieved by `bombardier` per second) for each combination of SBC (ODROID-N2+ and RPi 4B), container technology (Docker and Podman), and attack frequency (1200, 1400, and 1600 TCP `SYN` segments sent by `hping3` per second). In the three figures, the interval 0:10 seconds (0 to 10 seconds) had no attackers, 10:20 seconds had one attacker, 20:30 seconds had two attackers, and there were three attackers after 30 seconds. Findings and implications are discussed below:

- It is evident how each additional attacker created stress on the system.
- The ODROID-N2+ consistently outperformed the RPi 4B across all configurations, highlighting the superior capabilities of the ODROID-N2+ in handling web server workloads, even under DDoS attacks.
- As the attack frequency increased from 1200 to 1600, there was a noticeable decrease in throughput for all configurations. This aligns with the expected behavior, demonstrating the sensitivity of web servers to higher attack rates.
- Docker and Podman exhibited similar performances, suggesting both container technologies handled the attack load similarly.

Choosing the appropriate SBC is critical for containerized web server deployments. The ODROID-N2+ demonstrated superior performance, making it a preferred choice for resource-intensive tasks.

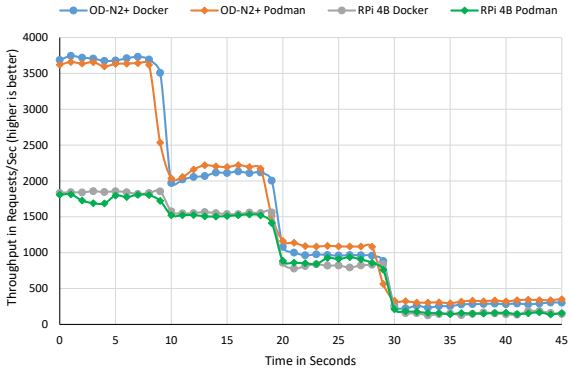

Figure 18: HTTP Throughput during a DDoS Attack with an `hping3` Frequency of 1200 TCP SYN Segments per Second

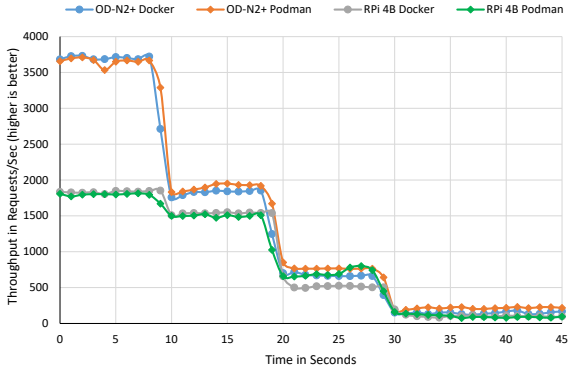

Figure 19: HTTP Throughput during a DDoS Attack with an `hping3` Frequency of 1400 TCP SYN Segments per Second

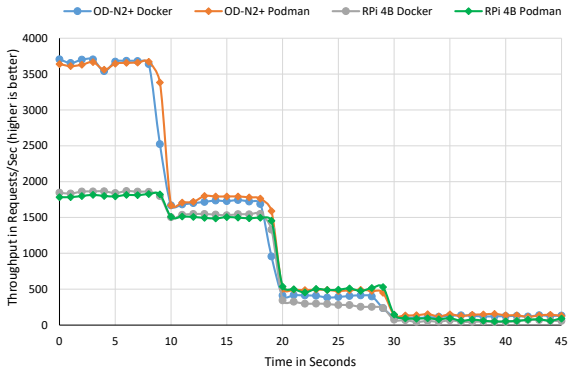

Figure 20: HTTP Throughput during a DDoS Attack with an `hping3` Frequency of 1600 TCP SYN Segments per Second

The experiment also underscores the importance of detecting and reinforcing security measures to mitigate the impact of DDoS attacks on web servers, especially in resource-constrained environments.

7.2 Variation of the Retrieved File Size

This section delves into and analyzes the results of an experiment where the retrieved file size was varied while keeping all other experimental parameters constant. That is, the experiment parameters were the same as in Table 5, where the `hping3` attack frequency was set to 1200, and the file size was varied (1024, 4096, 8192, and 16384 bytes). The focus is on understanding the impact of the retrieved file size on the performance and resilience of containerized web servers under a DDoS attack.

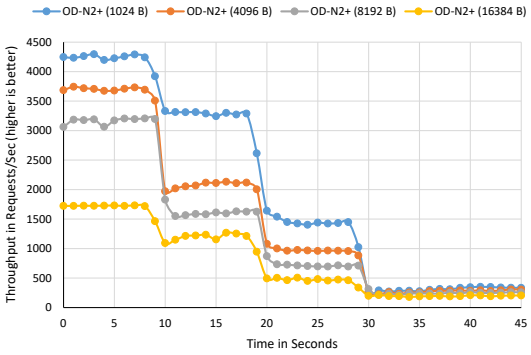

Figure 21: HTTP Throughput during a DDoS Attack with an `hping3` Frequency of 1200 TCP SYN Segments per Second when Varying the Retrieved File Size (Docker)

The experiment results, showcasing the average HTTP throughput in requests per second for each file size, are presented in Figure 21. For reasons of space, only the results of the ODROID-N2+ for Docker are shown. This experiment provides valuable insights into the nuanced relationship between file size and web server resilience under DDoS attacks. Smaller file sizes demonstrate better throughput, highlighting the importance of optimizing file size in web server deployments.

7.3 Variation of the Web Server

This section explores and investigates the results of an experiment where two different web servers were tested, while maintaining all other parameters constant. That is, the experiment parameters were the same as in Table 5, where the `hping3` attack frequency was set to 1200, and the web servers were Apache [12] and NGINX [8]. The aim was to understand the impact of the choice of the web server on the performance and resilience of containerized web servers under DDoS attacks.

Figure 22 presents the results of the experiment with Docker, showcasing the average throughput in requests per second for Apache and NGINX, on the ODROID-N2+ and RPi 4B.

The main findings of this experiment are presented and analyzed below:

- In this experiment, NGINX consistently outperformed Apache across both SBCs, showcasing its efficiency in handling concurrent requests, including when under DDoS attacks.
- As in previous experiments, the ODROID-N2+ demonstrated a significantly better performance than the RPi 4B, for both

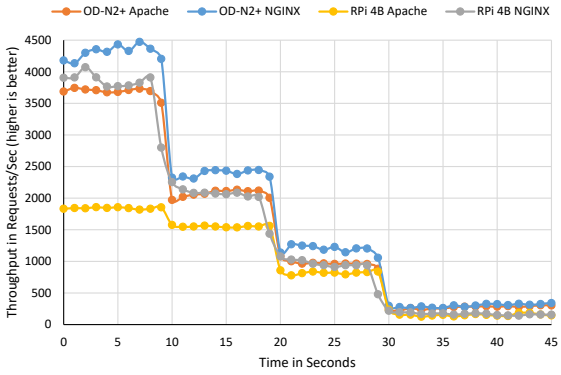

Figure 22: HTTP Throughput during a DDoS Attack with an `hping3` Frequency of 1200 TCP SYN Segments per Second when Varying the Web Server (Docker)

Apache and NGINX, indicating that its hardware specifications (SoC, memory, and network capabilities) are more suited for handling the increased load associated with DDoS attacks.

The experiment highlights the significant impact of the choice of web servers on the performance and resilience of containerized web servers, when under DDoS attacks. For this test, NGINX emerged as the preferred choice, consistently outperforming Apache in terms of throughput and stability, when deployed under the same conditions. The findings emphasize the need for practitioners to carefully consider web servers' characteristics when designing containerized solutions, especially in resource-constrained environments.

8 CONCLUSIONS AND FUTURE WORK

This research thoroughly investigated the performance dynamics of Docker and Podman containers on multiple ARM-based SBCs. The experiments were mainly focused on image-processing workloads and the resilience of these containerized systems against potential DDoS attacks.

The findings seem to indicate that at the level of the container technologies, Docker and Podman demonstrated similar performance. The ODROID-N2+ with 4 GB RAM emerged as a standout performer. For example, it showcased its prowess by efficiently handling up to 128 containerized `ImageMagick`/SSH servers simultaneously. This research also sheds light on the significant impact of DDoS attacks on containerized services, encouraging the implementation of security policies to mitigate them.

It is worth clarifying that the SBCs of the Raspberry Pi Foundation are well supported by the community, and obstacles are quickly resolved by searching in specialized forums. This is not the case for the SBCs of Hardkernel, and the authors had a hard time solving the issues they encountered due to the limited support. So practitioners should consider this parameter in their selection, especially if they plan to connect numerous additional components to the SBCs, that may not be supported by the different models of ODROID (due to, for example, the lack of drivers).

Future work could delve deeper into adaptive security measures to mitigate DDoS attacks in containerized deployments and assess their effectiveness on SBCs. Additionally, incorporating other SBCs into forthcoming studies could result in further valuable guidance for practitioners, especially after the recent release of the Raspberry Pi 5.

ACKNOWLEDGMENTS

We are grateful to "Faculty Commons" and the "College of Arts, Humanities, & Sciences" at Jacksonville State University for partially funding this project.

REFERENCES

[1] [n. d.]. *Bombardier: A HTTP(S) Benchmarking Tool.* https://github.com/codesenberg/bombardier
[2] Armbian Project. [n. d.]. *Linux for ARM Development Boards.* https://www.armbian.com
[3] Alessandro Arrichiello and Gianni Salinetti. 2022. *Podman for DevOps: Containerization Reimagined with Podman and its Companion Tools.* Packt Publishing.
[4] Jeorgithon Damasceno, Jamilson Dantas, and Jean Araujo. 2022. Network Edge Router Performance Evaluation: An OpenWrt-Based Approach. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI 2022).* Madrid, Spain.
[5] Guillaume Everarts de Velp, Etienne Rivière, and Ramin Sadre. 2020. Understanding the Performance of Container Execution Environments. In *2020 6th International Workshop on Container Technologies and Container Clouds (WOC 2020).* Delft, Netherlands.
[6] Ramzi Debab and Walid Khaled Hidouci. 2020. Containers Runtimes War: A Comparative Study. In *2020 Future Technologies Conference (FTC 2020).* Virtual Event.
[7] Eric Gamess and Mausam Parajuli. 2023. Performance Evaluation of the Docker Technology on Different Raspberry Pi Models. In *2023 5th International Electronics Communication Conference (IECC 2023).* Osaka City, Japan.
[8] Glaucio Guerra. 2023. *NGINX Simplified: Practical Guide to Web Server Configuration and Optimization.* Independently published.
[9] Hardkernel. [n. d.]. *ODROID-N2+ with 2 GB RAM.* https://www.hardkernel.com/shop/odroid-n2-with-2gbyte-ram-2
[10] Hardkernel. [n. d.]. *ODROID-N2+ with 4 GB RAM.* https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram-2
[11] Hardkernel. [n. d.]. *ODROID-XU4.* https://www.hardkernel.com/shop/odroid-xu4-special-price
[12] Darren Harkness. 2022. *Apache Essentials: Install, Configure, Maintain* (2nd ed.). Apress.
[13] Meron Istifanos and Israel Tekahun. 2020. *Performance Evaluation of Raspberry Pi 3B as a Web Server.* B.S. Thesis. Blekinge Institute of Technology, Karlskrona, Sweden.
[14] Ákos Kovács. 2017. Comparison of Different Linux Containers. In *2017 40th International Conference on Telecommunications and Signal Processing (TSP 2017).* Barcelona, Spain.
[15] Ashish Lingayat, Ranjana R. Badre, and Anil Kumar Gupta. 2018. Performance Evaluation for Deploying Docker Containers on Baremetal and Virtual Machine. In *2018 3rd International Conference on Communication and Electronics Systems (ICCES 2018).* Coimbatore, India.
[16] Roberto Morabito. 2016. A Performance Evaluation of Container Technologies on Internet of Things Devices. In *2016 IEEE International Conference on Computer Communications (IEEE IFOCOM 2016).* San Francisco, California, USA.
[17] Nigel Poulton. 2023. *Getting Started with Docker.* Nielson Book Services.
[18] Raspberry Pi Foundation. [n. d.]. *Raspberry Pi 4 Model B.* https://www.raspberrypi.com/products/raspberry-pi-4-model-b
[19] Raspberry Pi Foundation. [n. d.]. *Raspberry Pi OS.* https://downloads.raspberrypi.org
[20] Gabriel Schenker. 2023. *The Ultimate Docker Container Book: Build, Test, Ship, and Run Containers with Docker and Kubernetes* (3rd ed.). Packt Publishing.
[21] Daniel Walsh. 2023. *Podman in Action: Secure, Rootless Containers for Kubernetes, Microservices, and More.* Manning.
[22] Yang Zhao, Nai Xia, Chen Tian, Bo Li, Yizhou Tang, Yi Wang, Gong Zhang, Rui Li, and Alex X. Liu. 2017. Performance of Container Networking Technologies. In *2017 Workshop on Hot Topics in Container Networking and Networked Systems (HotConNet 2017).* Los Angeles, California, USA.