Santa Clara University

Department of Computer Science and Engineering

Date: June 11, 2022

I HEREBY RECOMMEND THAT THE THESIS PREPARED

UNDER MY SUPERVISION BY

Gagan Gupta
Kyle Fenole
Siddharth Venkatesh

ENTITLED

Container Migration in Ad-hoc Wireless Mesh Networks

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

DocuSigned by:
Behnam Dezfouli
6ED175D6B91A4FF...
____
Thesis Advisor
Dr. Behnam Dezfouli

N. Ling
N. Ling (Jun 9, 2022 14:23 PDT)
____
Department Chair
Dr. Nam Ling

Container Migration in Ad-hoc Wireless Mesh Networks

by

Gagan Gupta

Kyle Fenole

Siddharth Venkatesh

SENIOR DESIGN PROJECT REPORT

Submitted in partial fulfillment of the Requirements
for the Degree of Bachelor of Science
in Computer Science and Engineering
in the School of Engineering at
Santa Clara University

June 11, 2022

Santa Clara, California

2021-2022

Acknowledgements

We would like to thank our advisor Dr. Behnam Dezfouli for all of his help throughout the past year on this project. We would also like to thank our dear friend Eason Liu for his participation in the project in the Fall 2021 quarter. Finally, we would like to thank our friends and professors for all of their support during our time spent at SCU.

Container Migration in Ad-hoc Wireless Mesh Networks

Gagan Gupta
Kyle Fenole
Siddharth Venkatesh

Department of Computer Science and Engineering
Santa Clara University
Santa Clara, California

June 11, 2022

ABSTRACT

This project aims to implement container migration on an ad hoc mesh network, which would allow for a mobile fog computing mesh network in areas with no connectivity. This kind of system would allow for a rapidly deployable and mobile compute cluster that could be used in rural scenarios or situations where existing connectivity infrastructures are out of commission i.e. disaster recovery. We use technologies like Docker, CRIU, and batman-adv to create a mesh compute network that is decentralized, flexible, and consistent. Docker with CRIU facilitates container live migration by checkpointing a container in one network node and restoring that container in another node. The batman-adv protocol routes network traffic on layer 2 therefore allowing any layer 3 protocol to be built on top of the existing network. Since this system is aimed to be used in rural areas/areas with no connectivity, power and time considerations were measured and analyzed to ensure reasonable mobile applications were possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we will discuss different types of mesh wireless networks and software container packaging technologies and how they can be combined to create a remote layer of computation. We'll first talk about the motivation and context of our project and other solutions that are used in the market. Then we will talk about how we implemented our system and discuss power and time considerations for our system. Finally, we will talk about future steps to improve on our current system and wrap it up with some concluding remarks.

1.1 Motivation

The location of computing has been changing in recent years from local to remote or edge computing. Given the increase in data, bandwidth limitations, and latency issues, the use of remote computing architecture has become more and more prevalent. Gartner predicts by 2025, 75% of enterprise generated data will be computed via remote computing [1]. Given the abundance of devices connected to the internet, there is a wide reach of mobile devices increasing global connectivity. As a result of the new developments of many networking and software packaging technologies, both concepts have merged, introducing us to a new field known as Fog Computing which is a paradigm that extends cloud computing and services to the edge of the network [2].

1.2 The Role of Fog Computing

Wireless connectivity is a key factor when trying to communicate in the current world. From having around 350 million users in 2000, the internet has easily over 5 billion users today (World Internet Users Statistics…) [3]. In fact in 2016, the UN Human Rights Council decided that internet access is a human right which's importance has been further shown through events like the COVID-19 pandemic (United Nations). [4] This means that getting connectivity everywhere in the world is essential for staying updated about any and all kinds of information. All of this information also now requires a high amount of computation, whether that means more powerful remote data centers or better local computational power in the hands of retail end-users. The need for connectivity and computation led us to start thinking about the possibilities that lie in between the datacenter and local hardware with new emerging concepts such as Fog Computing. Fog computing is a natural step in the realm of computation as it allows for computation to be done right on the edge of the local network which lies in between the end-user and the data center [2]. Fog computing aims to be close to the end-user and have high mobility while providing access to application, storage, and compute resources. Using the high mobility of this concept alongside existing mesh networking solutions leads to a system that can be widespread, be mobile, and provide semi-local computation.

1.3 The Role of Software Containers

Software containers are used very often in industry for deploying quick, highly controllable software packages that can run in various computational environments [5]. These software packages contain dependency information, code, and other elements like system and runtime tools that allow for containers to be run in numerous environments. Containers can run on top of any existing traditional OS with various kinds of hardware in any computational location. Containers are also extremely customizable in terms of hardware and interface allocation while maintaining runtime isolation from the environment the container resides in. These reasons make containers highly used in software development as they are very flexible for deployment and controllability. This ease of deployment however can be used for other applications pertaining to the previously discussed Fog Computing. If a wireless mesh network's nodes each had the capability to run containers, one could theoretically use their own wireless network as a platform for computation. This combination of mesh networks and containers essentially would establish a Fog Computing layer where computation and data management is done on the edge of the network between the end-user and the cloud.

1.4 Thesis Structure

This thesis will first look into related work pertaining to current solutions for computing in the market and how their downsides led to the design choices made for our project. Next, we will discuss existing wireless mesh, software container, and container migration technologies that were considered while designing our final system before looking at our proposed solution. After we look at our system, we will then mention our considerations and the measurements we took to benchmark our system. Finally, we will explain possible applications of our system, the future steps the system could take and then conclude our findings.

Chapter 2

Related Work

2.1 Current Solutions

When personal computers first came around in the 1970s, computation was primarily local as computers lacked powerful processors and computer-to-computer communication was lacking greatly. Over time as PCs became more powerful and more connected, computation started to shift slowly from local to remote computing. When the era of data came around, the emergence of the cloud led even more computing to be done in remote locations like data centers. These steps solidified two traditional ways of computing in the present day: local computing and remote/cloud computing. Local computing options allow for the end-user to have complete control of the hardware and the software of a system and the system can be completely isolated from the internet. However, using local computing means that hardware isn't instantly, dynamically scalable and mobile devices can also suffer greatly trying to do large/constant computational tasks locally both in terms of raw power and battery life. Cloud computing, on the other hand, primarily circulates around the three big cloud vendors: Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure. Cloud computing allows customization of most of the cloud compute environment but the hardware still isn't directly accessible. Cloud computing also can have high latency and security concerns due to the data moving before and after computation through the internet. Another majority issue with cloud computing for end-users is that cloud computing is only cost effective if it's done at scale.

With many technologies for both processing and networking improving rapidly, both of these methods of computing are still very viable in today's world. However with the scale of data only getting larger and the number of connected devices increasing rapidly, this duopoly of computing can get very congested and inhibited. This issue led to new computational layers being developed in between the two existing layers such as edge computing and fog computing. This project falls into the fog computing category as the resources available sit on the edge of the network. The style of wireless connectivity networks have also evolved over the years to include concepts like mesh networking or ad-hoc mesh networking. This project aims to combine the elements of ad-hoc networking, software containers, and container checkpoint and restore to create a fog computing layer that has high mobility and has no need for outside connectivity.

There are already many papers and projects discussing ad-hoc networking and software containers and their migration. One paper published by IEEE titled *Performance Evaluation and Optimization of B.A.T.M.A.N. V Routing for Aerial and Ground-Based Mobile Ad-Hoc Networks* describes a setup created by researchers which uses B.A.T.M.A.N to create a wireless mesh network specifically designed to be set up on ground and air vehicles [6]. Although this paper does not include any information on containers, it does provide a plethora of information and work on using B.A.T.M.A.N for an optimized dynamic wireless mesh network. For this reason, it was quite relevant for the creation of the system in our Senior Design project. In addition, because of this research's emphasis on vehicular mesh networks, it is especially helpful for our project's potential use case of mounting access points on drones or rovers.

Another paper published *Aura: An Incentive-Driven Ad-Hoc IoT Cloud Framework for Proximal Mobile Computation Offloading* mentions how the growth of mobile applications requires enhanced computational resources in order to ensure better performance, security, and usability requiring developments such as IoT to handle the computation [7]. Hence they developed Aura which is a cloud computing model allowing mobile clients to create ad hoc and flexible clouds providing localized computational capability from untapped computing resources. This is very applicable to our senior design project as this project gave clients full control to start, stop, migrate, and restart computations in localized IoT devices as the mobile users move between different physical locations akin to us running containers on our Raspberry Pi's which would give users the ability to perform similar actions on containers.

Additionally we used the paper published by IEEE titled, *Con-Pi: A Distributed Container-Based Edge and Fog Computing Framework* which discusses how edge and fog computing paradigms overcome the limitations of cloud-centric execution for differing latency-sensitive IoT applications by offering simputing resources closer to the data source such as single board computers like a Raspberry Pi 4 [8]. To address some of the issues that come with this they developed a fully distributed framework, named Con-Pi which exploits the concept of containerization through the use of Docker containers. This paper is very applicable to our senior design project as we are using Raspberry Pi's to perform operations such as freezing and migration on containers. Additionally the paper discusses energy and resource management to maximize the efficiency of our distributed model which we found to be beneficial in the development of our project.

Docker Container Deployment in Distributed Fog Infrastructures with Checkpoint/Restart is a paper which addresses the issue of container start times being long, and proposes a solution to speed up the creation of new Docker containers [9]. It does so by starting new programs from an already created but snapshotted container, thereby skipping the step of starting up the program in the container. This is useful for our project because of how much our design relies on the freezing and unfreezing of containers in Docker, as well as the requirement of having fast load times (and consequently less overall power draw). The research in this text gives helpful insight and information on freezing docker containers.

Lastly we looked into another IEEE paper titled *MACE: a Mobile Ad-Hoc Computing Emulation Framework* which introduces a framework which facilitates and accelerates the development and testing of edge computing applications [10]. It mentions how with the adoption of IoT in many sectors the need for edge and fog computation has grown due to their low latency on the edge of networks. The issue is that there has been a lack of emulation tools for mobile computing as certain features are missing. This study proposes a framework that enables the emulation of mobile distributed applications in a virtual environment, so that system designers can easily modify scenarios and topologies composed by mobile wireless nodes. MACE is very similar to the idea we have for our project however through container live migration along with the mobile nature of the batman-adv protocol, we hope to retain a more mobile and topology flexible network.

Chapter 3

Existing Technologies

3.1 Wireless Mesh Networks

3.1.1 IEEE 802.11

The IEEE 802.11 standard is part of the IEEE 802 set of local area network (LAN) technical standards which provide the basis for wireless network devices to communicate with each other and access the Internet [11]. The IEEE 802.11 standard has been talked about in mesh-related papers as the expansions and optimizations since the standard was released have significantly improved its functionality [12]. The most applicable 802.11 standards for this project's use cases are the 802.11p, 802.11s, 802.11ah, and the still active 802.11ax standards.

The 802.11p standard amendment was released in which there were initial talks of V2X (vehicle-to-x) connections [13]. 802.11p specifically describes the services and functions required by stations to operate in a rapidly varying environment and to exchange messages without joining a basic service set (BSS). The addressment of rapidly changing/varying environments in this standard amendment directly connects to our use case of deploying our system in any environment. The 802.11s standard amendment came out in 2011 and directly pertains to mesh networking [14]. They state that "This amendment describes protocols for IEEE 802.11 stations to form self-configuring multi-hop networks that support both broadcast/multicast and unicast data delivery.". The standard then proceeds to delve into topics like mesh basic service sets (MBSS), mesh data frames, mesh paths, mesh peering, mesh peering management (MPM), and even mesh power modes. The 802.11ah standard amendment released in 2016 talked about 802.11-based systems with sub 1 GHz bands (excluding 470 to 698 Mhz as they are TV White Space bands) that include a transmission range up to 1 km and a minimum data rate of at least 100 Kb/s [15]. More importantly this amendment is focused on enabling the operation of these types of systems in a license-exempt manner. The 802.11ax standard amendment modified the PHY and MAC layers for high-efficiency operation in frequency bands ranging from 1 GHz to 7.125GHz [16]. Finding the 802.11 hybrid-protocol that is able to support mesh systems and long range connections while being efficient and robust will allow for our system to behave as close to intended as possible.

3.1.2 Ad-Hoc Networking/B.A.T.M.A.N.

Ad-Hoc Networking was also introduced through a IEEE 802.11 standard in 2007 and was implemented through the Independent Basic Service Set (IBSS) 802.11 Local Area Network (LAN) [17]. This form of LAN allows for stations to communicate directly with each other without any prior need of preplanning the network. This makes ad hoc networks fully decentralized as access points in the IBSS mesh can all act independently or together as a mesh-like network where nodes can enter and leave the mesh without affecting other nodes.

Classical routing protocols are typically not well suited for wireless ad-hoc networks because such networks dynamically change their topology and are based on an inherently unreliable medium. The B.A.T.M.A.N (The Better Approach to Mobile Ad-hoc Networking) algorithm divides the knowledge about the best end-to-end paths between nodes in the mesh to all participating nodes [18]. Each node perceives and maintains only the information about the best next hop towards all other nodes. The algorithm then selects the node whom it receives quicker and more reliable originator messages from as the currently best next hop configures its routing table respectively. The nodes of the network connect to each other which forms the wireless mesh network. The access points use B.A.T.M.A.N. which operates on layer 2 to connect to each other, therefore simplifying the process of making the mesh network which increases efficiency and reduces the room for errors. These reasons make the B.A.T.M.A.N. protocol an applicable routing protocol for multi-hop ad-hoc mesh networks.

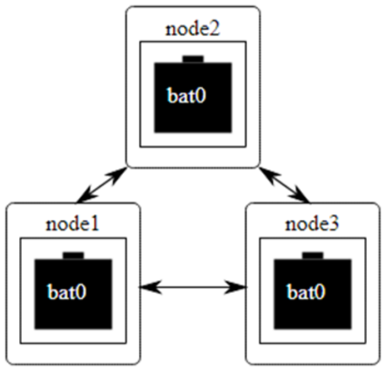


Figure 3.1: Nodes using Batman to connect with each other via the bat0 interface [18]

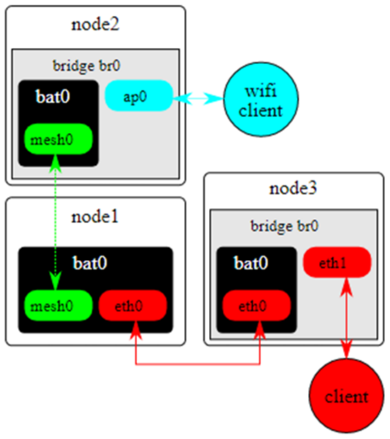


Figure 3.2: Demonstrating the network interfaces for the Access Points which can communicate via the mesh, eth, or ap interface to both other nodes and clients [18]

3.1.3 OpenWrt

OpenWrt is an open source project based on Linux which is primarily used on embedded devices to route network traffic [19]. It is an open source command line interface that is based on Linux to manage route network traffic while retaining the ability to have a writable filesystem with package management. OpenWrt supports mesh networks therefore allowing users to reliably utilize the software for authentication and encryption as well. Additionally, OpenWrt is easily configurable and it can be used on various platforms including personal computers or other kinds of embedded systems.

3.2 Containers/Software Packaging

3.2.1 Docker

Docker is a tool designed to make it easier to create, deploy, run applications by using containers as it packages your application and all its dependencies together in the form of a container [20]. Docker's architecture includes a client and a daemon where the daemon is responsible for managing Docker objects like containers, images, volumes, and networks. The client sends commands to a daemon such as build, pull, and run and a single client has the ability to control multiple daemons. The Docker container framework has widespread adoption and also supports experimental features such as CRIU (Checkpoint/Restore in Userspace) which is the one of the current best implementations of live migration in userspace for Linux.

3.2.2 Kata Container

Kata Containers are virtual machines that act and perform like containers, but use hardware virtualization technology to increase security and provide stronger workload isolation [21]. Kata containers provide the best of both worlds as they have the speed and lightweight of containers while also integrating with the container management layer, such as Docker, to provide the security advantages of Virtual Machines. One major downside for Kata Containers is that to implement container checkpoint and restore one has to use another technology like QEMU or Docker within Kata Container.

3.2.3 Podman

Podman provides a daemonless, open source solution for managing, creating, and running containers (including Docker) on Linux [22]. Podman is also good at isolating running containers which increases security. Overall, Podman provides an efficient and effective means of handling Docker containers in a Linux environment.

3.3 Container Live Migration

3.3.1 Checkpoint and Restore

Since containers are running on top of an existing host OS, when that user/device roams from one access point to another access point in a network, it is possible to migrate a container from the old access point to the new access point using the checkpoint and restore (C/R) functionality [23]. Through this feature, containers are “freezed” using the checkpoint function and will hold all its existing information in various image files. The state of the container will then be stored to the disk via the image files where it can then be transported to another host system to resume operation using the restore function. This movement of the container and its state is known as container live migration. One of the most interesting and useful features of live migration is that the whole state of the container is freezed and not only the data. This means that one could freeze a container while it’s executing its code as well as the state of the memory it is using and port that container to a new machine where it will continue to run from exactly the point at which it was frozen. This scalability and the flexibility that our system wishes to offer will be directly enhanced by the addition of live migration as all code/data containers will have the option to move hosts to ensure both data privacy and optimized connectivity.

3.3.2 CRIU

Checkpoint and Restore in Userspace for Linux (CRIU) is Linux software that started development in 2012 to help introduce userspace checkpoint and restore functionality for containers in Linux [24]. This was developed due to checkpoint and restore only being supported when the host OS of the container was either Windows or MacOS. CRIU makes it possible to perform operations like container live migration and snapshots for Linux host OS users. Live migration also allows for creation of execution level snapshots that could be used for debugging, application behavior analysis, process duplication, and even adding a pseudo-save feature for all applications. This type of snapshot management will allow for our system to save the state of all device containers on the mesh network as well as move these state snapshots into storage whenever a device completely disconnects from our network. If the device reconnects to the network then the state snapshots of the device's container can be brought back from storage and can be redeployed on the access point that the device reconnected to. CRIU was also implemented into Docker 1.13 as an experimental feature meaning that most Docker containers simply can be checkpointed and restored with client Docker commands sent to the daemon. Since most code development environments including ours are based with Linux, this feature is essential to get our live migration functionality working.

Chapter 4

Proposed Solution

4.1 System Requirements

Through the papers and solutions viewed in chapter 2 and the technologies discussed in chapter 3, we were able to narrow down three important traits we deemed to be the most significant for our system to follow. The first important trait of our system is that it is decentralized meaning that there is no master node in our network. This is accomplished through the use of a batman-adv ad hoc network which runs on the basis that no single node has all the data about the best route through the network. This creates a network of collective intelligence where every node can act independently as well as collectively if nodes are within range of one another. It is also required that our system is flexible which is achieved by using containers which are highly configurable for both software, hardware, and interfaces. This will be done through the use of Docker containers which are used frequently in practice and have significantly more support compared to other solutions. Additionally, Docker containers can run any kind of custom tasks and if loaded onto DockerHub, internet-connected nodes can automatically pull and use those containers. It is important that our system also maintains consistency by allowing nodes to enter and leave the mesh network at no cost to other nodes which is supported through the sd hoc nature of our network. Also if a node leaves the network, containers can be stopped and migrated between nodes through use of CRIU. Containers will start execution in the new node right from where they stopped execution in the previous node.

The mesh network consists of an arbitrary amount of access points (APs) where some nodes can have more computing power than others. All the access points will run the containers and these containers can be checkpointed and restored using the CRIU functionality implemented in Docker. Our most powerful node is a Kingdel Mini Desktop Computer which is aimed to be a vertically scaled node in the mesh network and could be used as a gateway node or as a storage location for containers that have been checkpointed and not restored for some time to reside. The smaller access points are going to be Raspberry Pi 4’s which are more mobile than the Kingdel and are able to be deployed anywhere in the field in order to extend the reach of the network. The access points will also have USB wireless adapters, in our case TP-Link AC600, which will potentially allow for stronger node-to-node connections by using the antenna's wireless interface. After the system is implemented, we will use a MakerHawk Fnirsi C1 digital power meter to measure the power of the Raspberry Pi's in the field.

4.2 System Components

4.2.1 Hardware

The hardware elements used in this project were chosen with the aforementioned design traits in mind. For our mobile, smaller access point we decided to go with Raspberry Pi 4's that have important features like 4GB of RAM, general purpose I/O (GPIO), and numerous ports like USB, micro HDMI, and gigabit ethernet. The Pi's consume relatively less power than other systems and are easy to deploy which allows for our whole system to be mobile. The Broadcom network chipset in the Pi also supports Independent Basic Service Set (IBSS) which allows for simple ad hoc networks to be established quickly. Our vertically scaled access point, we are using a Kingdel mini desktop computer which has an Intel i5 Dual Core CPU, 8GB RAM, and a variety of access ports including two NICs, four USB, and four COM RS232 (serial ports). This device is a Linux based machine and gives the mesh more computation power than any of the Pi's can provide independently. However, the higher power consumption of the Kingdel means that it would most likely not be as mobile as the Pi's in the network. Each one of the access points has an additional TP-Link AC600 USB 802.11 WiFi antenna mainly to provide extra wireless interfaces for network configuration.

4.2.2 Software

The software elements used in our project were chosen with our design traits and hardware in mind. For our OS we decided to use Linux Ubuntu Focal Long Term Support (LTS) 20.04 as it is a tried-and-true OS for many of our other system components such as CRIU which does not work with Ubuntu LTS 21.04 or Ubuntu 22.04. Ubuntu Focal also has had LTS for longer which leads to more compatible software options and potentially more documentation. To implement the ad hoc network we are using batman-adv which uses the B.A.T.M.A.N. (Better Approach to Mobile Adhoc Networking) protocol. It runs on the MAC Address level and the ad hoc mesh can have any number of nodes. Nodes can also join and leave without affecting other nodes and no one node is aware of the whole network topology. Since batman-adv is layer 3 agnostic it can run protocols like IPv4, IPv6, DHCP, IPX, etc. on top of it [18]. This protocol also allows for non-mesh clients to autoroam from node to node without any additional code. Our container solution that we decided to go with was Docker by itself as Docker has significant online support and DockerHub or Github is fantastic for online container storage and retrieval. Another reason we chose Docker was that Checkpoint and Restore in Userspace for Linux (CRIU) was recently integrated into Docker and we plan to implement CRIU. We chose CRIU as we are running Linux host OS's on all the access points and we have the ability to use live container migration between environments.

4.2.3 Architecture

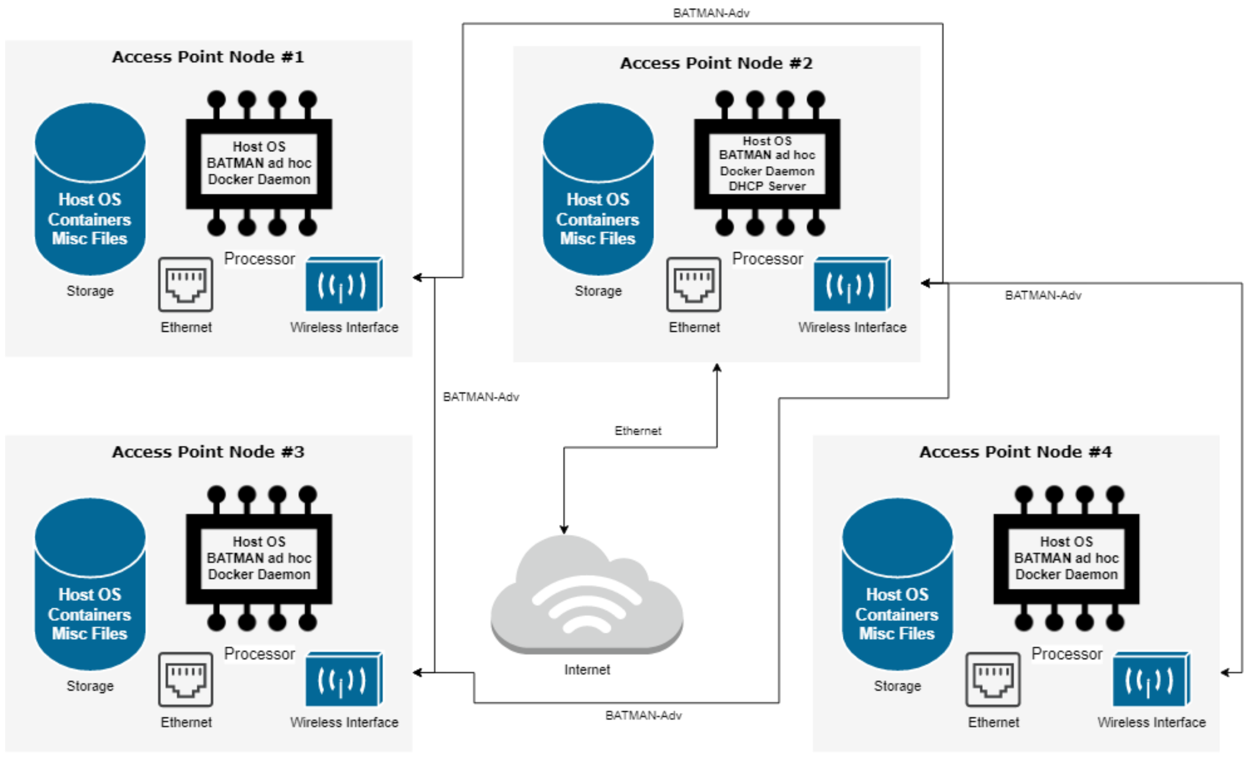


Figure 4.1: Diagram of the WMN System

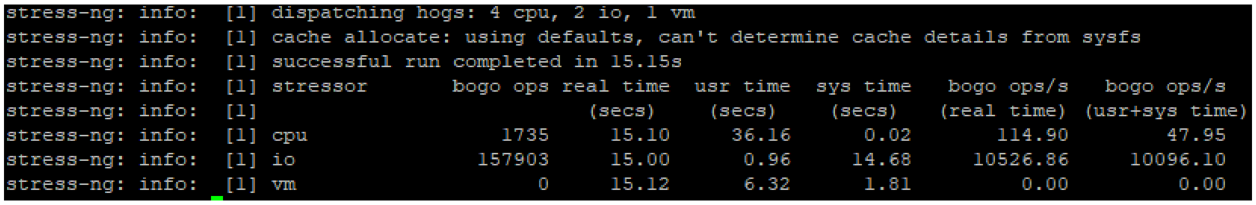
Figure 4.1, we can see multiple access points connected together via their wireless interfaces through the batman-adv protocol. Node #2 is configured to be a gateway node and can therefore connect to the internet while running a DHCP server to assign IP addresses on top of the layer 3 agnostic ad hoc network. Each node has storage, a processor, and various interfaces which allow for container processing to be done on any node.

4.3 System in Operation

```
ubuntu@ubuntu: ~$ sudo batctl n
[ B.A.T.M.A.N. adv 2019.4, MainIF/MAC: wlan0/e4:5f:01:77:8b:88
(bat0/36:85:86:59:23:cb BATMAN_IV)]
IF                  Neighbor                 last-seen
wlan0               e4:5f:01:78:42:ba        0.588s
```

This command line snippet shows the use of the batman-adv control library batctl to query for all the nodes within range of the current node in the ad hoc network.

```
ubuntu@ubuntu:~$ sudo docker run --security-opt=seccomp:unconfined –rm --name cr -d
alexeiled/stress-ng --cpu 4 --io 2 --vm 1 --vm-bytes IG --timeout 15s --metrics-brief
c33fa4a77d6a7f49cl5b8ab6ddl7c574b66b4aac60437cedl10a0f10118366867
```



```
stress-ng: info:  [1] dispatching hogs: 4 cpu, 2 io, 1 vm
stress-ng: info:  [1] cache allocate: using defaults, can't determine cache details from sysfs
stress-ng: info:  [1] successful run completed in 15.15s
stress-ng: info:  [1] stressor       bogo ops real time  usr time  sys time   bogo ops/s   bogo ops/s
stress-ng: info:  [1]                           (secs)    (secs)    (secs)   (real time) (usr+sys time)
stress-ng: info:  [1] cpu                1735     15.10     36.16      0.02       114.90        47.95
stress-ng: info:  [1] io               157903     15.00      0.96     14.68     10526.86     10096.10
stress-ng: info:  [1] vm                    0     15.12      6.32      1.81         0.00         0.00
```

Figure 4.2: Screenshot of container stress test results

This command and screenshot shows a node running a stress test container with 4 cpu stressors, 2 I/O stressors, and 1 VM stressor. This container was run as part of the tests done for the performance evaluation section

```
ubuntu@ubuntu:~$ sudo scp ~/cpi.tar.gz ubuntu@192.168.123.2:/home/ubuntu/cpi.tar.gz
ubuntu@192.168.123.2’s password:
```

cp1.tar.gz

This command uses the IPv4 addresses assigned on top of the layer 3 agnostic ad hoc network with a simple scp call to transfer the container checkpointed image files from one node to another.

4.4 Analysis of Surrounding Issues

Computer networks allow for a near limitless amount of data transfer and communication, and they have become a staple in modern computing. Our project heavily uses networking technology, and is built on the assumption that the data transmitted and run on our mesh network will be secure and reliable through the B.A.T.M.A.N. V protocol. Because of how ubiquitous wireless data transmission is in modern computing, a number of security measures and techniques exist in order to protect the integrity and confidentiality of information sent over a network. An ethical consideration of our project is whether anyone who is legitimately using the wireless mesh network has their data secured. In our case, it would be ensuring that the containers that are sent from node to node cannot be tampered with during transmission, nor can the information be read by an attacker analyzing network traffic. For this reason, it is essential that our project incorporates standard and up to date security practices, such as encrypting network traffic and using protocols that guarantee reliable data transfer. This adds some overhead to the process of sending and receiving containers between nodes, however it is an important and typically worthwhile tradeoff. In order to uphold professional responsibility, a potential configuration for the network would be to default to encryption of data while providing the user the option to disable encryption if they had a high performance and not sensitive container they needed to run on the system. It is near impossible to be completely secure, as it is not unheard of for researchers to uncover vulnerabilities even in universally adopted security standards. Our project makes sure to also stay within wireless broadcast laws as we are using conventional wavelengths and are not interfering with other networks.

Chapter 5

Performance Evaluations

5.1 Considerations

When designing the system around the hardware, software, and design traits, we decided to have a performance focus on the power and time during intensive container live migration through the ad hoc network. Our main concerns while building this mobile fog computing platform was its usability in locations that lack widespread connectivity and power. The power measurements were done with the aforementioned MakerHawk Fnirsi C1 power measurement tool through the USB Type C power interface on the Pi's. The data was extracted over Micro USB from the power measurement tool and fed into the Fnirsi software on our local machine. The MakerHawk device and software allowed for the monitoring of voltage, current, power, and capacity (for battery applications) while being able to set triggers on when to start and stop measurements. We chose to do the testing with a stress test container to demonstrate the worst case scenario for both the power draw and the CRIU checkpoint/restore time. The stressors generated by the stress test saturate elements like the page tables in the Pi which leads to the checkpoint and restore of CRIU being fully tested. The power graphs below show tests varying from base power measurements to the power when the whole system is connected with batman-adv ad hoc network and doing a checkpoint on a container.

5.2 Measurements

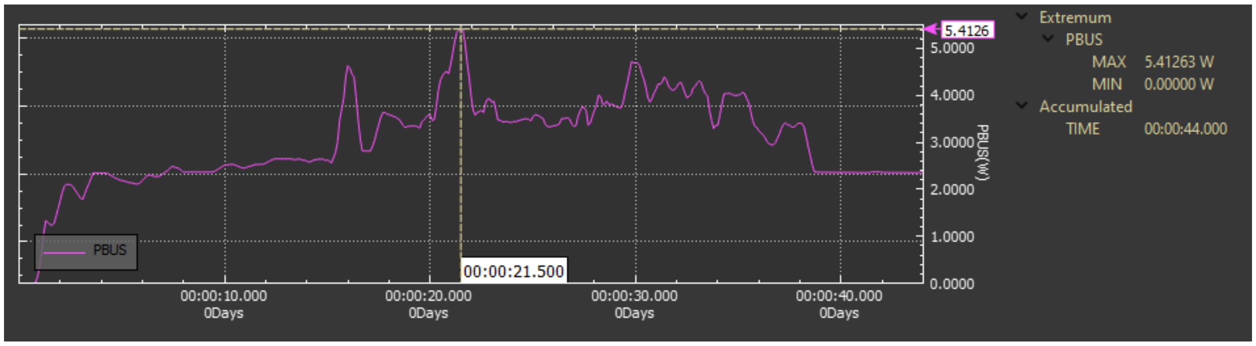


Figure 5.1: Maximum Power Consumption on Startup of Linux

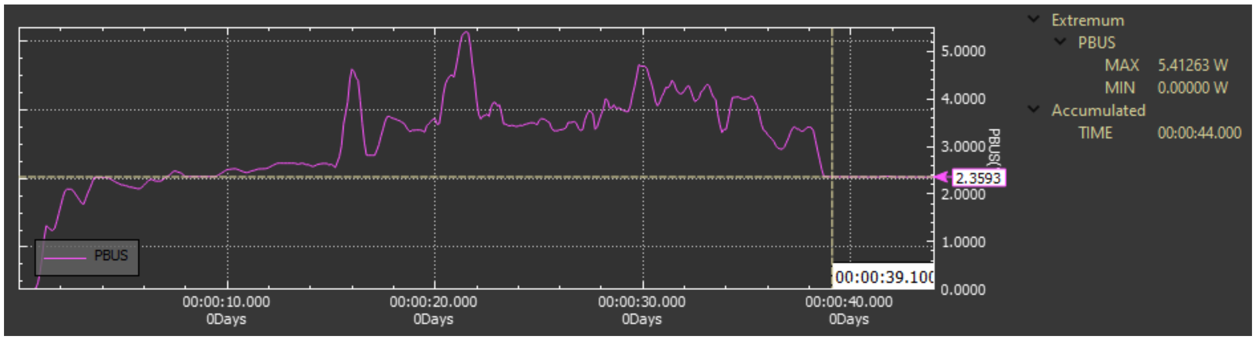


Figure 5.2: Power Consumption of Node Running Linux after Startup

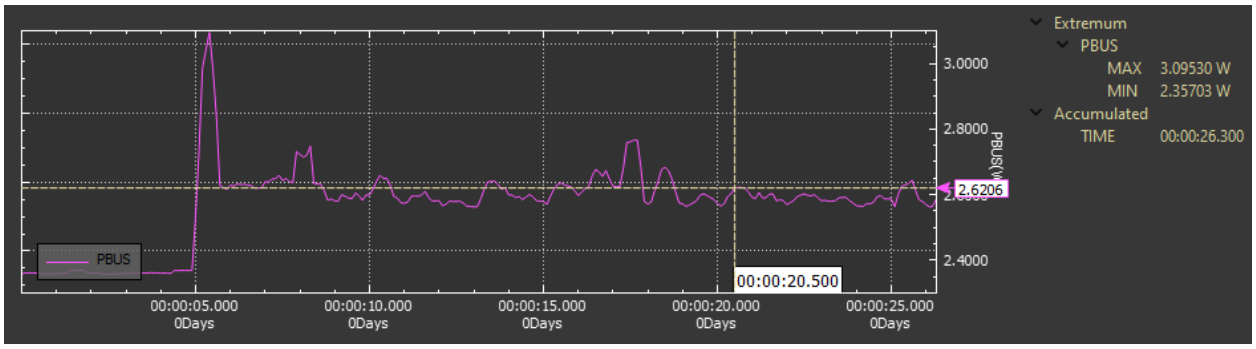


Figure 5.3: Increase in Power Consumption When Activating 8

Figure 5.1 shows the graph of the wattage draw during the startup of linux on the Raspberry Pi and shows that the max wattage draw was around 5.41W draw. Figure 5.2 shows that after around 40 seconds of initialization, the Raspberry Pi 4 settles around 2.36W of constant, stable power draw. Figure 5.3 shows the startup of the batman-adv ad hoc wireless mesh network where we can see the power draw goes from around 2.36W to 2.62W draw which is about an 11% increase in power draw while the linux and the network are active. Since the activity of the mesh is of an ad hoc and decentralized nature, nodes continually look for neighbors which leads to not-smooth power consumption in the end of Figure 5.3 compared to the end of Figure 5.2 or beginning of Figure 5.3.

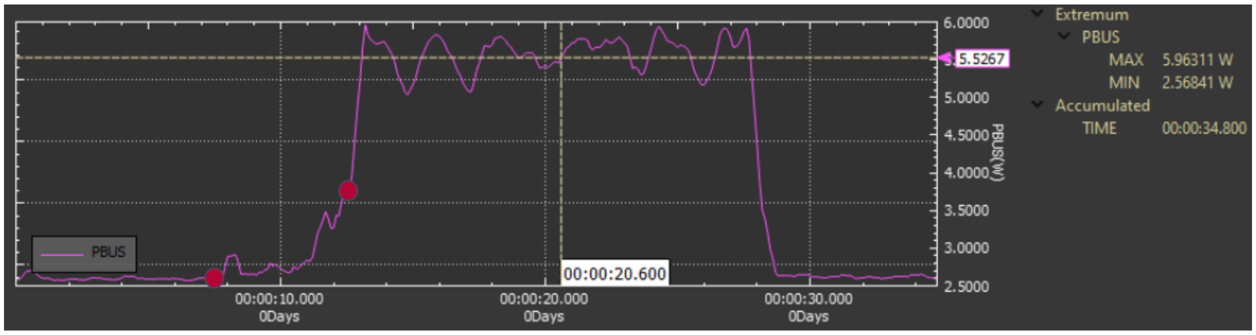


Figure 5.4: Power Consumption When Running Stress Test Container

Figure 5.4 shows the Raspberry Pi which is running linux and the batman-adv network grab and run a Docker container stress test from DockerHub. Between the two red dots the container is being grabbed from DockerHub after which the container runs for about 15 seconds at about 5.5W draw. This stress test container is the same one shown in Figure 4.2 where there are 4 cpu stressors, 2 I/O stressors, and 1 VM stressor.

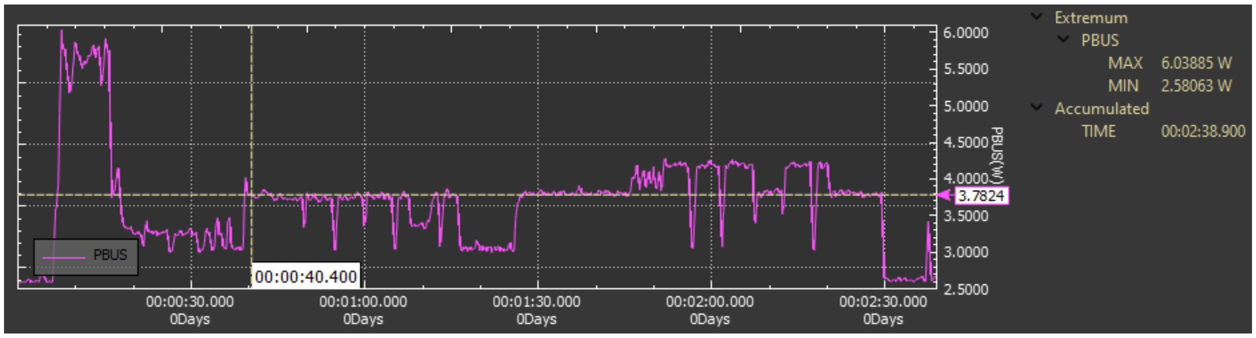


Figure 5.5: Power Consumption When Checkpointing Stress Test Container

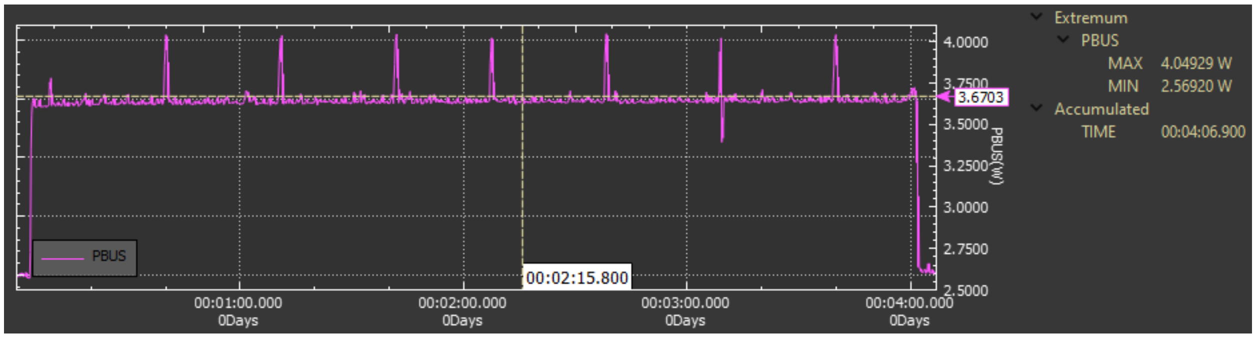


Figure 5.6: Power Consumption When Compressing Stress Test Container

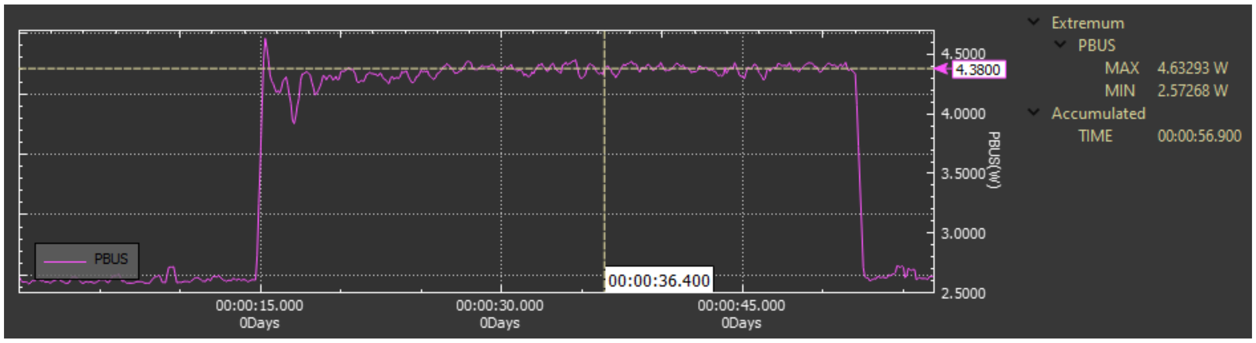


Figure 5.7: Power Consumption When Transferring Stress Test Container

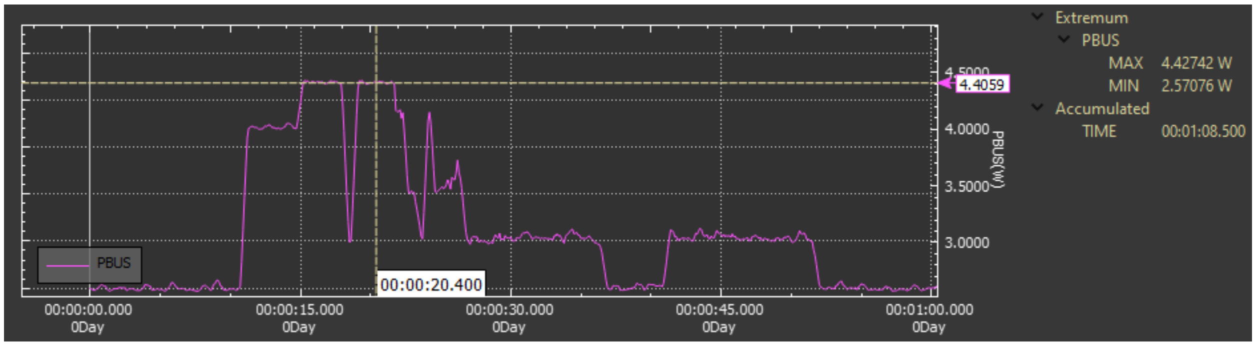


Figure 5.8: Power Consumption When Decompressing Stress Test Container

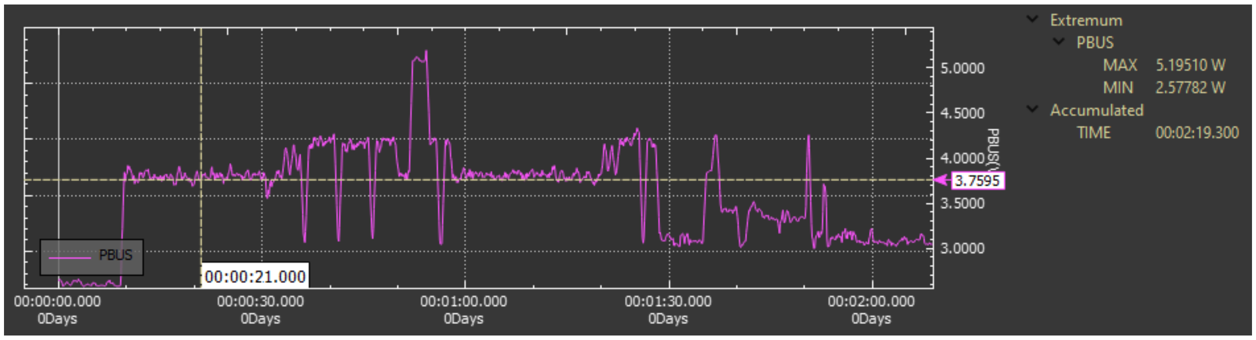


Figure 5.9: Power Consumption When Restoring Stress Test Container

Figure 5.5 shows the stress test container being checkpointed using CRIU which takes around 134 seconds at a power draw of around 3.78W draw. Figure 5.6 shows the TAR.GZ compress running which takes around 239 seconds at about 3.67W draw where the TAR.GZ size is 218 MB. Figure 5.7 shows the compressed checkpoint transfer power draw and time which ends up being around 38 seconds with a power draw of around 4.38W. This transfer through the ad hoc network was at a data rate of 5.8 MB/s. Figure 5.8 shows the TAR.GZ uncompress running which takes around 17 seconds at about 4.41W draw. Finally, Figure 5.9 shows the stress test container being restored on the new node using CRIU which takes around 121 seconds at a power draw of around 3.76W draw.

5.3 Summary and Analysis

Step	Time (sec)	Approx. Watts	Per Step Wh
Checkpoint	133.7	3.78	0.1404
Compress	238.9	3.67	0.2435
Transfer	37.5	4.38	0.0456
Uncompress	16.6	4.41	0.0203
Restore	120.1	3.75	0.1251

Table 5.1: Table of Power and Time measurements for container migration

After gathering all of the measurements we were able to see that the whole checkpoint, migration, and restore process had a total power usage of about 0.58 Wh for the worst case scenario container migration. For some context modern phones have about 15Wh capacity and portable power stations contain upwards of 1-2kWh capacity. If we were to run the stress test on the Raspberry Pi 4 constantly for an hour, it would only consume approximately 5.5-6.0Wh. Overall, the power measurements showed us that it would be viable to run multiple Raspberry Pi 4’s in remote locations however the time of the container migration took over 8 minutes stop-to-start. This large amount of time is because the container migration we implemented was neither pipelined or put in parallel however with more optimizations, time can be drastically reduced.

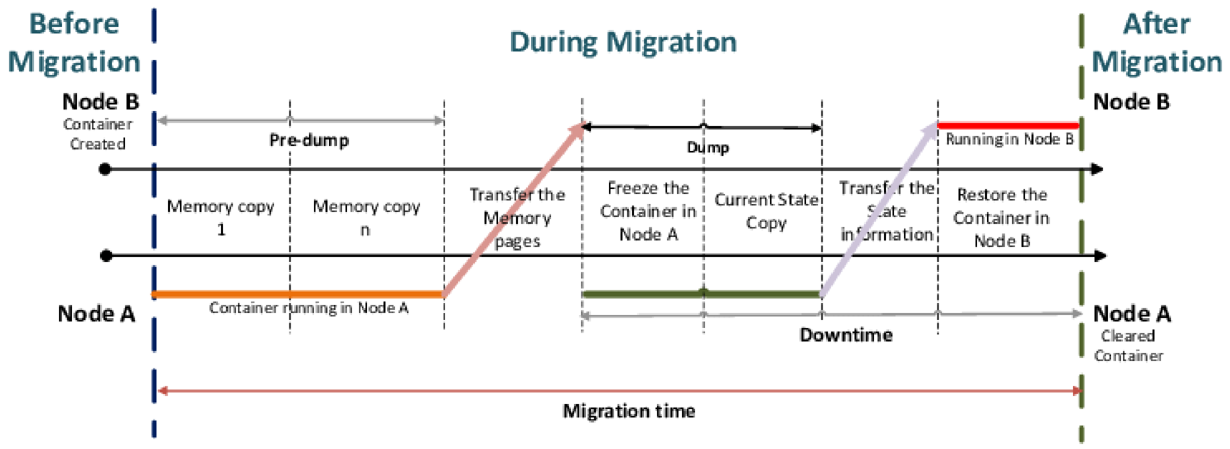


Figure 5.10: CRIU-based Container live migration [25]

Figure 5.10 shows a pipelined and parallized implementation of container migration through CRIU. Since all the files created from a checkpoint are image files and containers can be checkpointed while continuously running, there can be some major improvements in the migration time and the container downtime. If node A is running the container, a checkpoint can be made without stopping the container and the memory pages can be sent to the second node B. These memory pages files are a large chunk of all the image files sent during live container migration and can be done so without the compression and decompression shown in our implementation. While these image files are being dumped into node B, node A can freeze the container in parallel. Then, the newly changed memory page images and the remaining state image files can be transferred from node A to B. Node B can then restore the container and continue its operation from where it left off. This pipeline implementation also places node A's checkpoint operation in parallel with node B's memory page dump. The pipeline from Figure 5.10 increases the uptime of the container by delaying the final freeze, removes the need for compression or decompression, and it does the migration and restore process in a more parallel manner leading to a significant reduction in overall container migration time.

Chapter 6

Future Improvements

6.1 Experimental Container Technology

Checkpoint and restore is still a new and experimental feature that was only added to Docker in late 2016 [26]. CRIU was first introduced in 2012 and was introduced in Docker in 2017 as an experimental feature. The CRIU PPA used to install the feature on top of Docker also shows that each new major upgrade of Ubuntu leads to a need for a different version of CRIU. After running into multiple issues while testing, we noted that Docker and its use of a daemon might pose issues resulting in container restore/dump failing. The CRIU github and other forums recommend a switch to Podman is a possible improvement since the use of Docker containers is possible in a daemonless manner.

6.2 Ad-Hoc Network Expansions

The batman-adv ad hoc network we were able to establish runs well on layer 2 however trying to interact with non-node devices outside the network requires extra code configuration and device management. As shown in figure 4.1, one node in the batman-adv ad hoc network has the ability to act as a gateway node running a DHCP server to assign IP addresses so that all nodes in the network can access the internet. Any non-gateway node can be a bridge node which can allow for non-mesh clients to connect to the network. Most of the implementations only allow the bridge node interface to be ethernet however with some work a wireless interface could replace ethernet. The batman-adv network we were able to establish had node-to-node links working but using the aforementioned bridge node to connect clients wasn't completely viable. Future work could include using the batman-adv ad hoc network as a mesh backhaul network with additional IEEE 802.11s antennas acting as an interface for non-mesh clients to connect to directly.

Chapter 7

Conclusion

This thesis shows that container live migration and ad hoc networking technologies have great potential for the future of mobile computing and its ability to reach those in areas with little to no connectivity. Mobile ad hoc networks have recently gained even more traction with the emergence of drones and other robotic vehicles and the improvements made in the release of the B.A.T.M.A.N. V ad hoc protocol used within batman-adv. With more work being done in both Docker and CRIU for container checkpoint and restore, the bugs we encountered during the restore process will hopefully be dealt with, This will hopefully allow for the container live migration process to become more seamless in various environments, leading to more usage of the concept.

In conclusion, we hope that this thesis showed the possibilities that mobile fog computing has to offer. We also hope to have provided an insight into the world of container technologies and special forms of networking. With the right steps taken in the future, these concepts can expand into having vastly more effective applications. We are happy to have been able to design and implement this project and hope our work is continued and improved on in the future.

References

[1] Gartner_Inc. “What Edge Computing Means for Infrastructure and Operations Leaders.” *Gartner*, https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders.

[2] Perspectives IoT, et al. “IOT, from Cloud to Fog Computing.” *Cisco Blogs*, 1 June 2021, https://blogs.cisco.com/perspectives/iot-from-cloud-to-fog-computing.

[3] “World Internet Users Statistics and 2022 World Population Stats.” *Internet World Stats*, https://www.internetworldstats.com/stats.htm.

[4] “The Promotion, Protection and Enjoyment of Human Rights on the Internet :” *United Nations*, United Nations, https://digitallibrary.un.org/record/845728?ln=en.

[5] “What Is a Container?” *Docker*, 20 Apr. 2022, https://www.docker.com/resources/what-container/.

[6] “Performance Evaluation and Optimization of B.A.T.M.A.N. V Routing for Aerial and Ground-Based Mobile Ad-Hoc Networks.” *IEEE Xplore*, https://ieeexplore.ieee.org/abstract/document/8746361?casa_token=yL-q3CJmuSkAAAAA%3Aw-A-cqvu6c25BR7HJSLKSNpOXlHog75CmVJbznkrs9QrcMVvCAENrAzarnOq_NPgmOHzu2cs.

[7] Hasan, Ragib, et al. “Aura: An Incentive-Driven Ad-Hoc IOT Cloud Framework for Proximal Mobile Computation Offloading.” *Future Generation Computer Systems*, North-Holland, 11 Dec. 2017, https://www.sciencedirect.com/science/article/abs/pii/S0167739X1732602X.

[8] “Con-PI: A Distributed Container-Based Edge and Fog Computing Framework.” *IEEE Xplore*, https://ieeexplore.ieee.org/abstract/document/9508413?casa_token=E95E2PqkmckAAAAA%3A

Wmzl3EggRE4RKdtziFX5hqN3HgYHACwuoCxjXmZ-iPqyIXDtj5BRTxsfhfkcXAA09cSzhDI0.

[9] “Docker Container Deployment in Distributed Fog Infrastructures with Checkpoint/Restart.” *IEEE Xplore*, https://ieeexplore.ieee.org/abstract/document/9126743?casa_token=DKFAnmHSNCUAAAAA%3ALeQQ9mWow_kE5Mq3Bk3e5NeBNb1dBxQcWRh6kZdY7CMe8xBa9OE3jmPLnaFi8f_OooZk8MK3.

[10] “Mace: A Mobile Ad-Hoc Computing Emulation Framework.” *IEEE Xplore*, https://ieeexplore.ieee.org/abstract/document/9522185?casa_token=Ie-_gNRc5J8AAAAA%3AESlhtr3TIMPp0EHNlVvnpL4JnV-fDzDhOuDs9I09WD66YHmcccZdhIX76bIArDwXqVfAhbVS.

[11] "IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications," in IEEE Std 802.11-1997 , vol., no., pp.1-445, 18 Nov. 1997, doi: 10.1109/IEEESTD.1997.85951. https://ieeexplore.ieee.org/document/654749

[12] "IEEE Standard for Information Technology--Telecommunications and Information Exchange between Systems - Local and Metropolitan Area Networks--Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," in IEEE Std 802.11-2020 (Revision of IEEE Std 802.11-2016) , vol., no., pp.1-4379, 26 Feb. 2021, doi: 10.1109/IEEESTD.2021.9363693. https://ieeexplore.ieee.org/document/9363693

[13] "IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 6: Wireless Access in Vehicular Environments," in IEEE Std 802.11p-2010 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008, IEEE Std 802.11r-2008, IEEE Std 802.11y-2008, IEEE Std 802.11n-2009, and IEEE Std 802.11w-2009) , vol., no., pp.1-51, 15 July 2010, doi: 10.1109/IEEESTD.2010.5514475. https://ieeexplore.ieee.org/document/5514475

[14] "IEEE Standard for Information Technology--Telecommunications and information exchange between systems--Local and metropolitan area networks--Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications Amendment 10: Mesh Networking," in IEEE Std 802.11s-2011 (Amendment to IEEE Std 802.11-2007 as amended by IEEE 802.11k-2008, IEEE 802.11r-2008, IEEE 802.11y-2008, IEEE 802.11w-2009, IEEE 802.11n-2009, IEEE 802.11p-2010, IEEE 802.11z-2010, IEEE 802.11v-2011, and IEEE 802.11u-2011) , vol., no., pp.1-372, 10 Sept. 2011, doi: 10.1109/IEEESTD.2011.6018236. https://ieeexplore.ieee.org/document/6018236

[15] "IEEE Standard for Information technology--Telecommunications and information exchange between systems - Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 2: Sub 1 GHz License Exempt Operation," in IEEE Std 802.11ah-2016 (Amendment to IEEE Std 802.11-2016, as amended by IEEE Std 802.11ai-2016) , vol., no., pp.1-594, 5 May 2017, doi: 10.1109/IEEESTD.2017.7920364. https://ieeexplore.ieee.org/document/7920364

[16] "IEEE Standard for Information Technology--Telecommunications and Information Exchange between Systems Local and Metropolitan Area Networks--Specific Requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: Enhancements for High-Efficiency WLAN," in IEEE Std 802.11ax-2021 (Amendment to IEEE Std 802.11-2020) , vol., no., pp.1-767, 19 May 2021, doi: 10.1109/IEEESTD.2021.9442429. https://ieeexplore.ieee.org/document/9442429

[17] "IEEE Standard for Information Technology - Telecommunications and Information Exchange Between Systems - Local and Metropolitan Area Networks - Specific Requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," in IEEE Std 802.11-2007 (Revision of IEEE Std 802.11-1999) , vol., no., pp.1-1076, 12 June 2007, doi: 10.1109/IEEESTD.2007.373646. https://ieeexplore.ieee.org/document/4248378

[18] “B.A.T.M.A.N. advanced,” *Wiki - batman-adv - Open Mesh*, 2020. [Online]. Available: https://www.open-mesh.org/projects/batman-adv/wiki/Wiki.

[19] “About the openwrt/LEDE project,” *OpenWrt Wiki*, 31-Jan-2022. [Online]. Available: https://openwrt.org/about.

[20] “Docker overview” *Docker*, 2021. https://docs.docker.com/get-started/overview/.

[21] “Kata containers - open source container runtime software,” *Kata Containers - Open Source Container Runtime Software*. [Online]. Available: https://katacontainers.io/.

[22] “What is podman?,” *Podman*, 04-May-2021. [Online]. Available: https://podman.io/whatis.html.

[23] “Docker checkpoint,” *Docker Documentation*, 2021. [Online]. Available: https://docs.docker.com/engine/reference/commandline/checkpoint/.

[24] “Main page,” CRIU. [Online]. Available: https://criu.org/Main_Page.

[25] Ramanathan et. al., “A Comprehensive Study of Virtual Machine and Container Based Core Network Components Migration in OpenROADM SDN-Enabled Network,” Aug. 2021. https://www.researchgate.net/publication/354236780_A_Comprehensive_Study_of_Virtual_Machine_and_Container_Based_Core_Network_Components_Migration_in_OpenROADM_SDN-Enabled_Network

[26] “Docker engine release notes,” *Docker Documentation*, 2017. [Online]. Available: https://docs.docker.com/engine/release-notes/prior-releases/.

Container_Migration_Adhoc_Wireless_Mesh_Networks_Publication

Final Audit Report 2022-06-09

Created:	2022-06-09
By:	Darcy Yaley (dyaley@scu.edu)
Status:	Signed
Transaction ID:	CBJCHBCAABAAEBj9kx2z7Gli0PdhHKvAwAW-3Xfga7PR

"Container_Migration_Adhoc_Wireless_Mesh_Networks_Publication" History

Document created by Darcy Yaley (dyaley@scu.edu)
2022-06-09 - 4:23:47 PM GMT

Document emailed to N. Ling (nling@scu.edu) for signature
2022-06-09 - 4:24:17 PM GMT

Email viewed by N. Ling (nling@scu.edu)
2022-06-09 - 9:22:45 PM GMT

Document e-signed by N. Ling (nling@scu.edu)
Signature Date: 2022-06-09 - 9:23:11 PM GMT - Time Source: server

Agreement completed.
2022-06-09 - 9:23:11 PM GMT