11-30-2023 10:00 AM

Migration in Edge Computing

Arshin Rezazadeh, *Western University*

Supervisor: Lutfiyya, Hanan, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy degree in Computer Science

Abstract

Mobile IoT applications often require low response time and high bandwidth. These applications include virtual reality, augmented reality, and online gaming. Currently, most data processing is done in the cloud. However, for latency-sensitive applications, the latency may need to be reduced. Edge and fog computing can be used to place application services close to mobile devices to reduce latency. However, as mobile devices move, latency increases, which can be decreased by moving the service to a closer edge/fog server. This can be addressed by migrating services so that the mobile device can receive services from the new server. These services can be run on a single or multiple virtual machines or containers.

Application services must be migrated together. Delays in multi-service migration can occur because some application services migrate faster than others, requiring them to wait for the rest of the services to migrate before continuing their tasks. However, in the last decade, most migration research has focused on cloud computing rather than edge and fog computing. Furthermore, migration methods and models are primarily intended for cloud applications.

This thesis focuses on migration techniques for managing IoT applications in edge and fog computing environments while considering the characteristics of IoT applications, the networking characteristics of the edge and fog servers, and the migration parameters of running IoT applications. This thesis contributes to the current state of the art by presenting the following contributions in fog and edge computing environments:

1. A comprehensive literature review and limitations on the migration of IoT applications from different perspectives, namely downtime and migration time reduction strategies, optimization techniques, and identifying critical parameters of migration.

2. A new migration method for latency-sensitive IoT applications to reduce downtime and migration time, including performance evaluations and comparisons to well-known migration methods.

3. New comprehensive models with non-average parameter values for higher precision and accuracy, including comparative analysis of the parameters that impact the performance of the investigated migration methods.

4. A new bandwidth allocation strategy for multi-service migrations of IoT applications to reduce downtime and migration time, including performance evaluations.

Keywords

Edge computing, fog computing, multi-service migration, downtime, migration time, bandwidth adjustment, machine learning, reinforcement learning.

Summary for Lay Audience

Internet-of-things (IoT) and cloud computing have been well-known in the last decade. However, there will be a data surge soon with massive data generation by IoT applications that the cloud cannot tolerate. Furthermore, mobile IoT applications often require low response time and high bandwidth. These applications include virtual reality, augmented reality, and online gaming. Currently, most data processing is done in the cloud. However, for latency-sensitive applications, the latency may need to be reduced. Edge and fog computing can be used to place application services in servers close to mobile devices to reduce latency. However, as mobile devices move, latency increases, which can be decreased by moving the service to a closer edge/fog server. Migrating services can address this so the mobile device can receive services from the new server.

Mobile device service migration for multi-service applications is critical in edge and fog computing. With the growing use of multi-service applications, it is critical to ensure that all application services complete their migration in parallel to avoid waiting time and additional delays in latency-sensitive applications. Delays in multi-service migration can occur because some application services migrate faster than others, requiring them to wait for the rest of the services to migrate before continuing their tasks and remaining operational. However, in the last decade, most migration research has focused on cloud computing rather than edge and fog computing. Furthermore, migration methods and models are primarily intended for cloud applications. The advancement of technology, the introduction of 5G and 6G networks, and evolving applications create a demand for migration techniques in edge and fog computing to support mobility with low response time.

The edge and fog computing paradigms are dynamic, distributed, and heterogeneous. Therefore, it is challenging to fully exploit the capabilities of this computing paradigm for various IoT-driven application scenarios in the absence of effective migration strategies for IoT application management.

This thesis focuses on various migration techniques for managing IoT applications in edge and fog computing environments while considering the characteristics of IoT applications and networking of the edge and fog servers and the migration parameters of running IoT applications.

Co-Authorship Statement

This thesis contains several articles written by Arshin Rezazadeh and supervised by Professor Hanan Lutfiyya. The articles are based on research done at the University of Western Ontario as part of the author's doctoral research. These works are discussed in chapters two through five. I primarily contributed to all chapters under the supervision of Professor Hanan Lutfiyya, and Davoud Abednejad assisted with parts of the experiments in Chapter 3.

Acknowledgments

Ph.D. is an exciting journey filled with incredible experiences that would not be possible without the encouragement and support of many people. Now that my journey is coming to an end, I would like to take this opportunity to share my deepest gratitude to everyone who has supported me on this journey.

Before anything else, I would like to express sincere thanks to my supervisor, Professor Hanan Lutfiyya, for providing me with the opportunity to pursue my studies under her supervision. I would like to express my appreciation for her ongoing encouragement, productive comments, advice, and support throughout all of the challenging and delightful moments of my Ph.D. adventure.

I would also like to thank Dr. Dan Lizotte, Dr. Kostas Kontogiannis, and Dr. Roberto Solis-Oba for their informative courses that provided very insightful perspectives on concepts related to my research. My research has also been enhanced by the support of fellow students and friends Mohsen Shirpour, Amir Haghighati Maleki, Duff Jones and Davoud Abednejad.

I would also like to thank the faculty and staff at Western University's Department of Computer Science for their support. I also thank the Computer Science Department's admin staff, especially Janice M Wiersma and Ange Muir, who continuously supported and responded to many queries throughout my Ph.D. candidature.

Lastly, and most importantly, I am grateful to my spouse for her everlasting love, devotion, appreciation, and understanding.

Arshin Rezazadeh
London, Ontario, Canada
October 2023

Table of Contents

List of Tables

List of Figures

Preface

Main Contributions

This thesis research was conducted in the Department of Computer Science at the University of Western Ontario. The main contributions of the thesis are discussed in Chapters 2-6 and are based on the following publications:

- **Arshin Rezazadeh**, Hanan Lutfiyya "A Novel Sustainable Bandwidth Allocation Strategy for Multiple Service Migration in 5G/6G Edge Computing," *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, IEEE, Dec. 2023, pp. 1211–1217

- **Arshin Rezazadeh**, Hanan Lutfiyya "Multi-microservice Migration Modelling, Comparison, and Potential in 5G/6G Mobile Edge Computing: A Non-average Parameter Values Approach," IEEE Access –accepted

- **Arshin Rezazadeh**, Davoud Abednejad, Hanan Lutfiyya "Hybrid-MiGrror: An Extension to the Hybrid Live Migration to Support Mobility in Edge Computing," *Journal of Ubiquitous Systems & Pervasive Networks, Vol. 18, No. 1,* pp. 39-48, January 2023 [INVITED PAPER]

- **Arshin Rezazadeh**, Davoud Abednejad, Hanan Lutfiyya "MiGrror: Mitigating Downtime in Mobile Edge Computing, An Extension to Live Migration, "*Procedia Computer Science, Vol. 203,* pp. 41-50, 2022 [BEST PAPER AWARD]

- **Arshin Rezazadeh**, Hanan Lutfiyya "Migration in Edge Computing: Review and Challenges," –submitted

Chapter 1

Introduction

Supporting latency-sensitive applications to be used by mobile devices requires placing one or more application services in remote computing resources to address the limited capacity of a mobile device. Mobile IoT applications often require low response time and high bandwidth [1]–[3]. These applications include virtual reality (VR), augmented reality (AR), online gaming, and smart vehicle applications. Currently, data processing is typically done in the cloud. However, the latency may be too high for latency-sensitive applications [3]. Edge and fog computing can be used to place application services in servers close to mobile devices in order to reduce latency [4]. However, as mobile devices move, the latency can increase, which can be mitigated by moving the service to another edge/fog server [1]. Migrating services can address this so the mobile device can receive services from the new server.

There is an increasing need to integrate multiaccess edge computing (MEC) with future mobile Internet-of-Things (IoT) devices and associated applications in the 5th Generation of Mobile Technology ("5G") [5], [6]. In this context, application services run on edge nodes close to devices, offer low response times and reduce remote traffic between devices and the cloud. Edge presence is fundamental to the success of the 5G/6G and the associated MEC standard demonstrated by the European Telecommunications Standards Institute (ETSI) [7]–[10].

When user mobility is considered, there is a need to migrate services for mobile devices. The edge servers host the applications to maintain proximity between mobile devices and edge [1], [6]. Therefore, services must follow users to maintain proximity. This requires service migration. These services can be run on single or multiple Virtual Machines (VMs) or containers of an application. Regardless of whether the services are hosted on VMs or containers, application services must be migrated together.

Service migration for multi-container applications is critical for mobile devices in edge and fog computing [5], [11]. With the increasing use of multi-containerized applications [5], including dependent tasks, it is crucial to ensure that all application containers finish their migration in parallel to prevent waiting time and further delays in latency-sensitive applications. This is especially important in the presence of dependent tasks that can be found in more than 75% of multi-containerized applications [12]. Recent research focuses on migrating multiple services [13]–[24]. Delays in multi-container migration can occur, as some application containers can complete their migration sooner than others, requiring them to wait to complete the migration of the rest of the containers before continuing their tasks and remaining operational.

Furthermore, when considering load balancing, services should migrate to balance the load; some edge nodes may become overloaded due to changing workloads, while others may remain underutilized [14]. However, most migration research in the last decade has focused on cloud computing rather than edge and fog computing. Migration techniques and models are fundamentally designed for cloud applications as well. The advancement of technology, the introduction of 5G and 6G networks, and evolving applications, create a demand for migration techniques to support mobility with low response time in edge and fog computing.

The edge paradigm aims to decrease latency by transferring intensive computational services from the cloud to the network edge. Though transferring these tasks to the edge may cause some delay, it will be minor compared to the benefits of performing these heavy tasks on higher-performance edge devices [25].

Researchers may have a different definition of fog computing. Some regard fog computing as a subset of edge computing, while we consider it as a superset of edge computing [26]. The fog extends from a user device to the cloud, while edge computing is only present at the network's edge. As a result, the fog paradigm includes the edge paradigm. Although fog definitions may lack coherence, existing literature indicates that most fog research employs similar language when describing the topology of fog networks.

1.1 Methodologies

In our research, we use a systematic research methodology.

- Qualitative Comparison: We conduct a thorough analysis of each research problem, identifying critical parameters, thoroughly researching literature techniques, and comparing them to our proposed method.

- Modeling: We use modeling techniques to address our research challenges. We 1) develop a comprehensive model that represents the key elements of our system and 2) formulate the problem by establishing specific topics of interest, including downtime and migration time. We extend the current models with an innovative approach that increases the precision and accuracy of models.

- Algorithms: We propose migration algorithms for mobile IoT applications in Edge and Fog computing environments. These fundamental algorithms are designed to solve migration problems. In addition, these algorithms can integrate into machine learning-based approaches to solve optimization problems.

- Evaluation: The proposed techniques in this thesis have been evaluated using three methodologies, namely analytical, discrete event-driven simulation, and practical implementation. Due to limited accessibility and management costs, simulation is a common evaluation methodology to evaluate the proposed algorithms in complex and large-scale systems. In this thesis, we used Python 3 as an analytical tool and MobFogSim simulation toolkit [27] for simulation.

Innovative method, algorithms, and models resulted from our research methodology.

1.2 Research Questions and Objectives

Mobile IoT applications often require low response time and high bandwidth [1], [3]. Currently, they are mostly run on the cloud, but the latency may be high for some mobile IoT applications. To satisfy the need for a lower response time, edge computing is an option [28]. However, as mobile devices move, the latency may increase. Migration can solve this problem, but presently, some limitations, e.g., lengthy downtimes and migration times, need to be improved for latency-sensitive services. Downtime and migration time are the two primary parameters in migration [14]. This thesis aims to enhance the Quality of Experience (QoE) for the user of IoT applications in Edge and Fog Computing environments. In order to accomplish this objective, we address the following research questions in an effort to resolve significant migration issues:

Q1: *How to reduce migration downtime for latency-sensitive applications?* Most research uses the pre-copy migration method [28], which produces significant downtime ranging from hundreds of milliseconds (ms) to several seconds (s) [29]–[36]. Consider a head-mounted device with an AR application. Some components of this application require migration since they must preserve the user's data, such as the assets in front of the user and their position. This app requires a latency of less than 17 ms to function properly [3], whereas most stated research output latencies and even only downtimes exceed 17 ms. The user's QoE may suffer as a result of such a long delay, especially for latency-sensitive applications, and there is a demand for a new approach to more seamless migration that replaces the pre-copy migration technique.

Q2: *How to reduce migration time for latency-sensitive applications?* Due to the limited resources of Edge/Fog servers, it may not always be possible to execute real-time IoT applications on these servers. Therefore, with a shorter migration time, resources can be made available to other services faster. Consequently, it is necessary to develop new migration methods to support latency-sensitive and real-time service migrations for IoT users while minimizing service interruptions during migration.

Q3: *How to more accurately model multi-service migration time and downtime?* Migration modeling has been used in studies [13]–[24], [37]–[42] to predict the future behavior of a system. Most research, however, is based on average parameter values and assumes that the input parameter values will remain unchanged. One of our objectives is to model downtime and migration time more accurately so that we can better understand future behavior.

Q4: *How to effectively increase bandwidth utilization for multi-service migrations to decrease downtime and migration time?* It has been a long time since migrations made use of dedicated bandwidth. This means that each VM/container has a set amount of bandwidth to use for data transfer during migration. One of our objectives is to redesign this bandwidth assignment model and use bandwidth more efficiently. The new design is expected to result in less downtime and migration time.

1.3 Thesis Contributions

This thesis makes the following contributions to the above-mentioned research problems:

1. A comprehensive review of the existing migration approaches and their limitations in Edge and Fog computing environments.

2. Investigates efficient migration methods to reduce downtime and migration time while considering mobility support in Edge and Fog computing environments (addresses Q1 and Q2).

- A migration method designed to reduce downtime and migration time in Edge and Fog computing environments.
- Algorithms for migration in Edge and Fog computing environments
- Mobility support with a use-case scenario.
- Reducing the data transfer amount during hand-off.
- Performance evaluation and comparisons to well-known migration methods, e.g., pre-copy method.
- Capable of integrating with additional techniques, including compression and machine learning (ML).

3. Puts forward models with non-average parameter values for higher precision and accuracy in Edge and Fog computing environments (addresses the Q3).

   - Mathematical migration models for heterogeneous multiple VMs/containers in Edge and Fog computing environments.
   - Utilize non-average and classical average values for parameters, such as the transfer rate and memory dirtying rate, during each migration period of every single VM/container.
   - A new downtime model
   - A new migration time model
   - A new migration overhead model
   - A comparative analysis of the input parameters that impact the performance of the investigated migration methods, such as MiGrror and pre-copy methods.

4. Proposes bandwidth allocation strategy for multi-service migrations of IoT applications in Edge and Fog computing environments (addresses the Q4).
    - A novel bandwidth allocation strategy for multi-service migration in Edge and Fog computing environments.
    - Adjusting migration bandwidth for multi-containerized applications.
    - Enhanced utilization of bandwidth for parallel service migration.
    - Reduced bandwidth requirements for migrating multiple services in Edge and Fog computing environments.
    - Performance evaluation and comparisons were made.

1.4 Thesis Organization

Figure 1.1 shows the structure of this thesis. The rest of this thesis is organized as follows:

- Chapter 2 presents the background and related work of current migration approaches and their limitations in Edge and Fog computing environments. This chapter is derived from:
    - **Arshin Rezazadeh**, Hanan Lutfiyya "Migration in Edge Computing: Review and Challenges," –submitted
- Chapter 3 presents the new migration method, MiGrror, for latency-sensitive IoT applications to reduce downtime and migration time. This chapter is derived from:

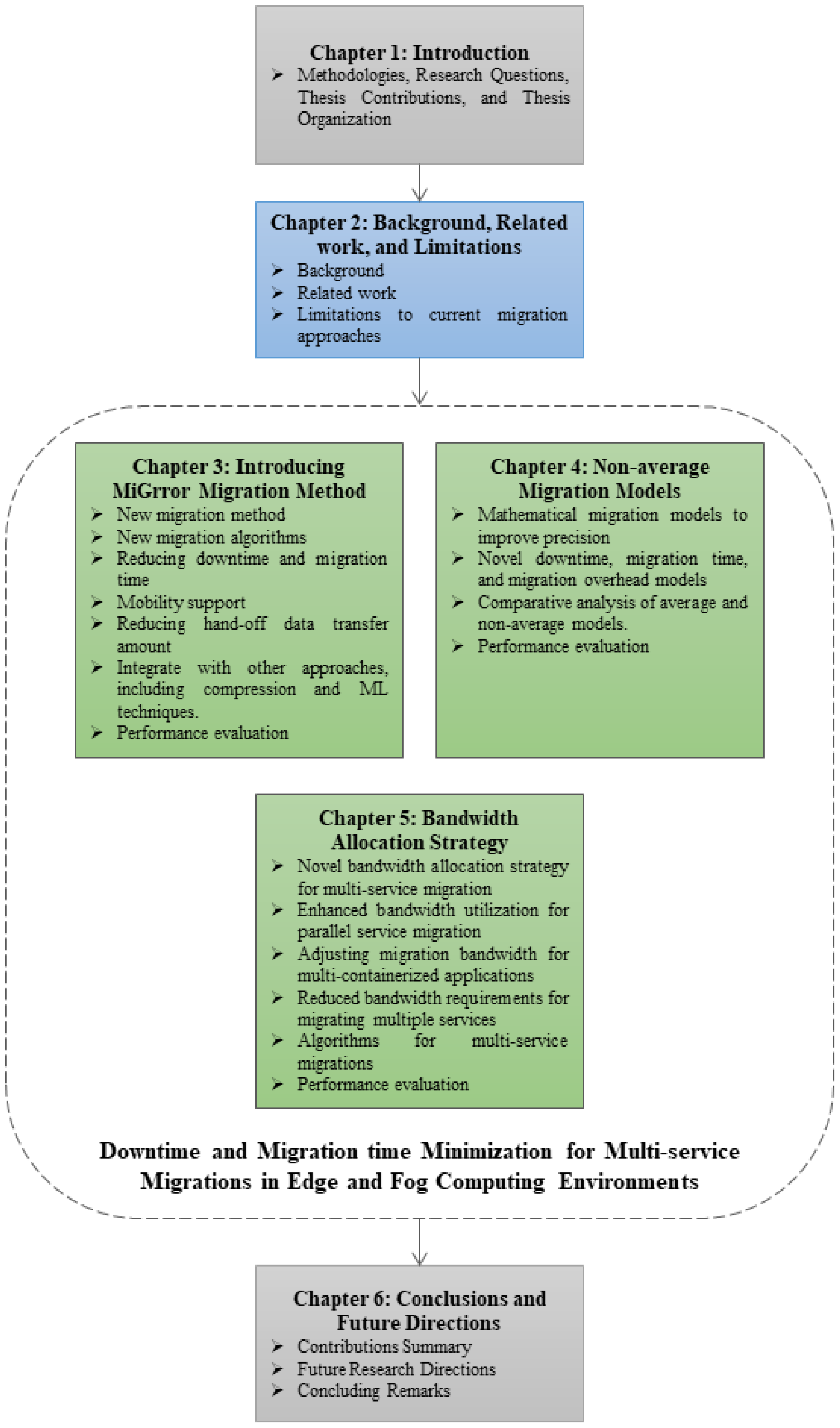
Chapter 1: Introduction
➢ Methodologies, Research Questions, Thesis Contributions, and Thesis Organization
Chapter 2: Background, Related work, and Limitations
➢ Background
➢ Related work
➢ Limitations to current migration approaches
Chapter 3: Introducing MiGrror Migration Method
➢ New migration method
➢ New migration algorithms
➢ Reducing downtime and migration time
➢ Mobility support
➢ Reducing hand-off data transfer amount
➢ Integrate with other approaches, including compression and ML techniques.
➢ Performance evaluation
Chapter 4: Non-average Migration Models
➢ Mathematical migration models to improve precision
➢ Novel downtime, migration time, and migration overhead models
➢ Comparative analysis of average and non-average models.
➢ Performance evaluation
Chapter 5: Bandwidth Allocation Strategy
➢ Novel bandwidth allocation strategy for multi-service migration
➢ Enhanced bandwidth utilization for parallel service migration
➢ Adjusting migration bandwidth for multi-containerized applications
➢ Reduced bandwidth requirements for migrating multiple services
➢ Algorithms for multi-service migrations
➢ Performance evaluation
Downtime and Migration time Minimization for Multi-service Migrations in Edge and Fog Computing Environments
Chapter 6: Conclusions and Future Directions
➢ Contributions Summary
➢ Future Research Directions
➢ Concluding Remarks

Figure 1.1 The thesis structure.

  - **Arshin Rezazadeh**, Davoud Abednejad, Hanan Lutfiyya “MiGrror: Mitigating Downtime in Mobile Edge Computing, An Extension to Live Migration, “Procedia Computer Science, Vol. 203, pp. 41-50, 2022 [BEST PAPER AWARD]

  - **Arshin Rezazadeh**, Davoud Abednejad, Hanan Lutfiyya “Hybrid-MiGrror: An Extension to the Hybrid Live Migration to Support Mobility in Edge Computing,” Journal of Ubiquitous Systems & Pervasive Networks, Vol. 18, No. 1, pp. 39-48, January 2023 [INVITED PAPER]

- Chapter 4 presents a new migration model with non-average parameter values to improve the precision of the output parameters. This chapter is derived from:

  - **Arshin Rezazadeh**, Hanan Lutfiyya “Multi-microservice Migration Modelling, Comparison, and Potential in 5G/6G Mobile Edge Computing: A Non-average Parameter Values Approach,” IEEE Access –accepted

- Chapter 5 presents a novel bandwidth allocation strategy for migrating multi-containerized IoT applications to reduce downtime and migration time in Edge and Fog computing environments. This chapter is derived from:

  - **Arshin Rezazadeh**, Hanan Lutfiyya “A Novel Sustainable Bandwidth Allocation Strategy for Multiple Service Migration in 5G/6G Edge Computing,“ *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, IEEE, Dec. 2023, pp. 1211–1217

- Chapter 6 concludes the thesis with a summary of the findings and offers new directions for future research.

Since the terms fog computing and edge computing are used interchangeably in the literature, we will primarily use the term edge computing in the rest of this thesis for clarity, which can refer to either edge or fog computing.

Chapter 2

Background, Related Work, and Limitations

This chapter investigates and reviews existing edge computing migration techniques from various perspectives, including downtime, migration time, migration models, bandwidth utilization, optimization, and performance evaluation. Separate limitations for each perspective on migration in edge computing environments are proposed based on an in-depth review of the literature. Finally, research gaps for enhancing the edge computing paradigm are identified and discussed.

2.1 Introduction

Supporting latency-sensitive applications to be used by mobile devices requires placing one or more application services in remote computing resources to address the limited capacity of a mobile device. Mobile IoT applications often require low response time and high bandwidth [1]–[3]. These applications include virtual reality (VR), augmented reality (AR), gaming and smart vehicle applications. Currently, data processing is

This chapter is derived from:

- **Arshin Rezazadeh**, Hanan Lutfiyya "Migration in Edge Computing: Review and Challenges," – submitted typically done in the cloud. However, the latency may be too high for latency-sensitive applications [3]. Edge computing can be used to place application services in servers close to mobile devices in order to reduce latency [4]. However, as mobile devices move, the latency can increase, which can be mitigated by moving the service to another edge server [1].

There is an increasing need to integrate multiaccess edge computing (MEC) with future mobile Internet-of-Things (IoT) devices and associated applications in the 5th Generation of Mobile Technology ("5G") [5], [6]. In this context, application services run on edge nodes close to devices, offer low response times and reduce remote traffic between devices and the cloud. Edge presence is fundamental to the success of the 5G/6G and the associated MEC standard demonstrated by the European Telecommunications Standards Institute (ETSI) [7]–[10].

When user mobility is considered, there is a need to migrate services for mobile devices. The edge servers host the applications to maintain proximity between mobile devices and edge [1], [6]. Therefore, services must follow users to maintain proximity. This requires service migration. Furthermore, when considering load balancing, services should migrate to balance the load; some edge nodes may become overloaded due to changing workloads, while others may remain underutilized [14].

The advancement of technology, the introduction of 5G and 6G networks, and evolving applications, creates a demand for techniques to support mobility with low response time. This Chapter focuses on the following:

1. Migration papers with the goals of reducing downtime and migration time.
2. New migration techniques within edge computing that provide better service availability, user mobility support, and low response time compared to the traditional migration approaches.

3. Papers focusing on migration models to analyze and optimize downtime and migration time; These models can apply to single or multiple services undergoing migration [13]–[24], [37]–[42].

There are different research topics related to the edge computing that researchers are currently working on around. This includes but are not limited to resource management [43], data security and privacy [44], offloading modeling [45], Data analytics in MEC [46], Vehicular MEC [47], Edge-computing-enabled Smart Cities [48], Edge Computing for Internet of Things [49], Integrated blockchain and edge computing [50], edge computing in 5G [51], and Mobile Augmented Reality with 5G MEC [52].

2.1.1 Paper Selection Method

The concept of computing resources closer to mobile devices is found in MEC, fog and edge computing [53]–[55].

Over ten thousand papers on MEC, fog, and edge computing have been published in the **last five years**, mostly from 2018, using the search terms "fog computing," "edge computing," and "MEC multiaccess edge computing." However, some older studies are included as they are a base of other research, such as the pre-copy migration technique [56], and fundamental studies that researchers still cite, such as fog computing [57]. All references are in English, and the vast majority, have been peer-reviewed. The exceptions are references from ETSI and the comsoc.org websites.

In the second pass of paper selection, we added "survey" to the above search terms ("fog computing survey," "edge computing survey," and "MEC multiaccess edge computing survey"), followed by "migration" ("migration fog computing survey," "migration edge computing survey," and "migration MEC multiaccess edge computing survey").

We then selected papers that focused on migration. We **did not include** papers that focused on aspects of computing that did not focus on migration and were not addressed by this document, such as the energy consumption and load balancing of migration techniques.

In the subsequent passes of paper selection, we surveyed **migration algorithms** and **migration modeling**. Selected papers included research on reducing **downtime** and **migration time** in both migration algorithms and migration modeling since users expect **low-latency** response time in modern mobile IoT applications **while on the move**. These papers are categorized in the following sections based on how the authors handle downtime and migration time, as well as how they model migration techniques to handle single and multiple service migrations. Multiple service migration has received increased focus recently, especially in light of modern containerized applications.

2.1.2 Outline

The remainder of this Chapter is organized as follows: Section 2.2 provides background information about fundamental migration techniques borrowed from cloud computing. Section 2.3 discusses how researchers reduce downtime and migration time from various perspectives. Section 2.4 describes multi-service migration in edge computing. Section 2.5 presents the models used to characterize migration techniques, followed by research directions in Section 2.6.

In this Chapter, a node refers to a MEC/edge node, and source and destination refer to source and destination MEC/edge nodes. Edge and fog Computing concepts are also referred to as "edge."

2.2 Background on Traditional Migration Methods

Edge computing reduces remote network traffic while providing low latency for client-server communications. These services must run on edge nodes that are physically close to devices in order to consistently provide such low latencies. We investigate migration methods and requirements for future mission-critical IoT applications, which demand **mobility support** and a real-time response [9], [58]. IoT mobile devices have limited resources for running multiple containerized services, and client-server latency worsens when fog-edge services must **migrate** to maintain proximity with users as the user devices move, as seen in figure 2.1.

One of the developing fog-edge computing solutions for the IoT infrastructure is the container.

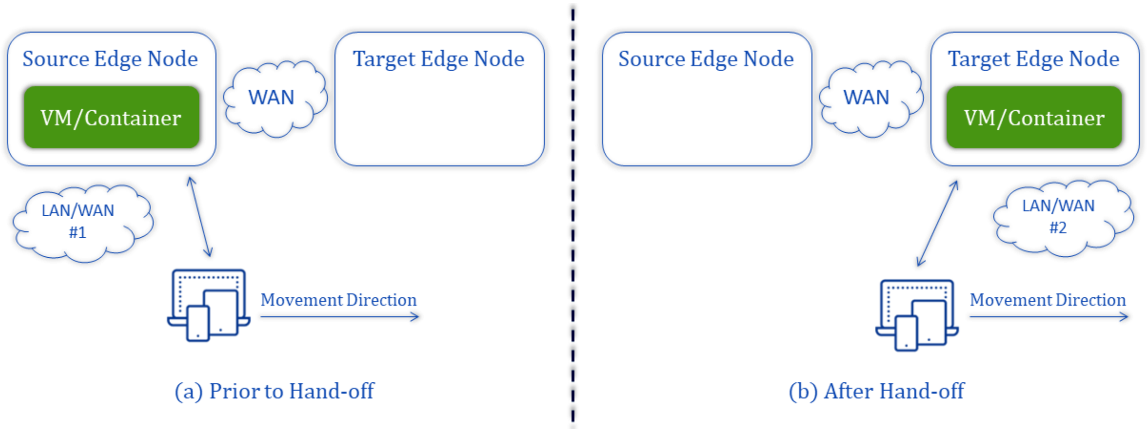

Figure 2.1 Prior to and after the VM/container hand-off.

2.2.1 Migration Phases

Migration is the process of transferring a running virtual machine (VM) or container from one edge node to another or the cloud without disrupting applications [59]. The **hand-off** is a component of migration [29] that is triggered when a device disconnects from one access point (AP) and connects to another AP. Downtime caused by VM/container migrations ranges from seconds to a few minutes [29]–[36]. The amount of downtime and page faults significantly impact the end-to-end delay [60]–[62]. Furthermore, because the user equipment (UE) must migrate from the old to the new connection point throughout the migration process, it is unable to access services or data during hand-off. There has been considerable work focused on reducing downtime [29], [31]–[35], [39], [63], [64]. **Downtime** occurs when a virtual machine or a container is unavailable during migration, which occurs when UE moves and hands off from one edge node to another. Data transfers are required when migrating VMs or containers. Recent research has focused on reducing the amount of data transferred during hand-off using different metrics such as runtime and offline characteristics. The research focuses on predicting the best time to trigger a hand-off and improving the selection of edge nodes to allow for shorter processing times [32], [34], [65], [66]. The stated techniques mostly use live migration approaches borrowed from cloud computing, which uses the pre-copy migration method [56] to reduce migration downtime [2], [29], [31]–[35], [65]–[68]. The following subsection will cover the techniques used in the stated studies.

2.2.2 Traditional Migration Methods

Stateful migration occurs when the service's state (including CPU, register, signal, and memory states) is stored; otherwise, stateless migration occurs [62], [65]. This document focuses on stateful migration methods.

The rest of this section provides an overview of the traditional stateful migration methods: **cold**, **pre-copy**, **post-copy**, and **hybrid-copy** migration methods. In recent years, these methods have been widely used in cloud computing and are currently employed in edge computing as well [13]–[24], [37]–[42].

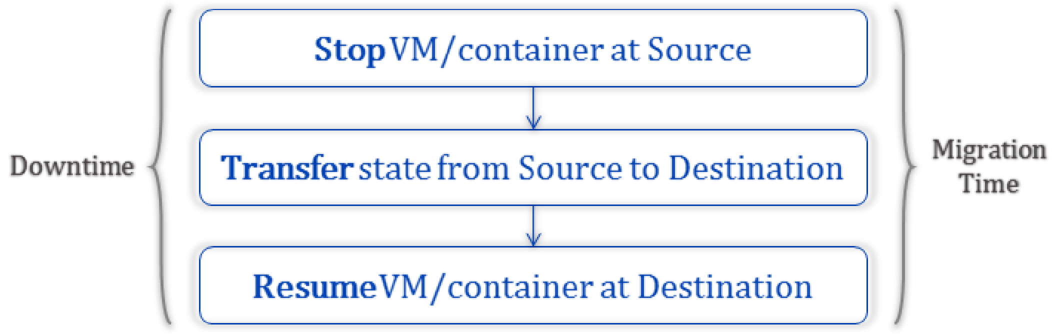
Stop VM/container at Source
Transfer state from Source to Destination
Resume VM/container at Destination
Downtime
Migration Time

Figure 2.2 Cold migration method

2.2.2.1 Cold Migration Method

A **cold migration** occurs when the VM/container execution is paused at the source. The data and contents of the VM/container's memory pages are transmitted to the target node that will host the VM/container. Service is then resumed at the destination by the host. During this period, applications running on a UE are unable to access the service until the VM/container resumes execution at the new location. As a result, the cold migration method involves a significant amount of **downtime**. Figure 2.2 shows that downtime and **total migration time** are the same for cold migration [62]. The total migration time is the amount of time needed to complete the migration process. Cold migration has the highest amount of downtime among migration methods [69].

2.2.2.2 Live Migration Methods

Live migration allows virtual machines and containers to remain operational for most of the migration process as opposed to cold migration, which causes significantly more downtime than live migration methods [70]. The live migration methods [70] include the following: pre-copy, post-copy, and hybrid migration. The rest of this section is an in-depth discussion of these methods.

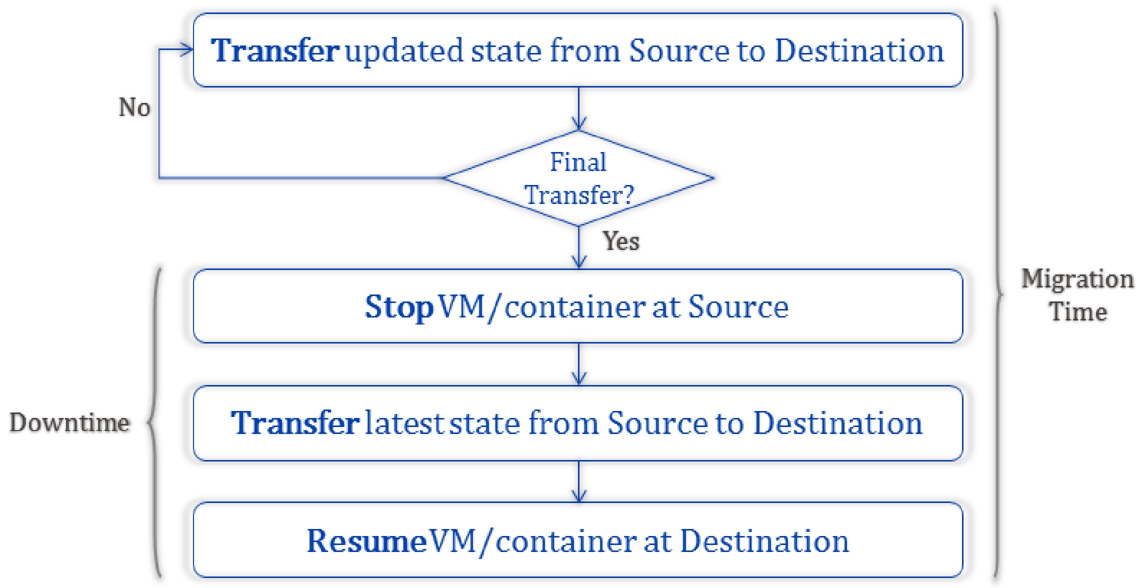
Transfer updated state from Source to Destination
No
Final Transfer?
Yes
Stop VM/container at Source
Downtime
Transfer latest state from Source to Destination
Resume VM/container at Destination
Migration Time

Figure 2.3 Pre-copy live migration method

2.2.2.2.1 Pre-Copy Live Migration Method

Pre-copy migration sends the entire VM/container state from the current node to the target node. An **iteration** is a round in which pre-copy waits for memory changes to be sent at the end of each round. The current node then resends **dirty pages**, which are updated memory pages from the previous iteration, over multiple iterations. Upon receiving the hand-off signal, the current node pauses the source VM/container execution to prevent memory and state modification and transfers the final dirty page and the latest changes in the runtime (execution) state, such as CPU and register updates, to the target edge node. Finally, the VM/container resumes operation on the target edge node [63]. The primary distinction between pre-copy and cold migration methods is the transmission phase. The transmission in the cold migration method includes the entire VM/container state (primarily related to the first iteration in the pre-copy method). This always involves memory pages as well as the runtime state of the VM/container being transferred.

On the other hand, in the pre-copy migration method, the source node sends the updated memory pages throughout the pre-copy stage, as well as any runtime state modifications. Therefore, since less data is transmitted while the VM/container is paused during hand-off, the pre-copy method downtime is usually less than the cold migration downtime [71]. The pre-copy method downtime is less deterministic since it is heavily influenced by the memory dirtying rate [69]. Finally, in contrast to the cold method, the pre-copy method typically transmits each memory page multiple times, which may have a negative impact on the **total amount of data transmitted** throughout the migration process and, consequently, the **total migration time** [62]. The steps of pre-copy migration are depicted in figure 2.3.

2.2.2.2.2 Post-Copy Live Migration Method

As presented in figure 2.4, before transferring the latest state from the source to the target, the **post-copy migration** method [72] pauses the VM/container execution to prevent runtime state changes. The state is then transferred to the target, along with the minimum memory and state required to resume the VM/container. The VM/container is then resumed at the target [73]. An access problem occurs when the VM attempts to access a page that the target has not received. A **page fault** occurs, and the source transmits the faulty page to the target [74]. When the VM/container is restarted at the target node during the post-copy process, any applications executing in the VM/container continue to run at the target. After sending all remaining pages, the page transfers to the target stops, and the VM/container post-copy migration is complete [75]. There are several distinct types of post-copy methods, each with its own distinct approach in the fourth stage, where page transfers may occur between the source and the target [72]. Each memory page is copied only once during the post-copy migration. As a result, it transmits a volume of data equal to that transferred during the cold migration phase but less than that transmitted during the pre-copy stage.

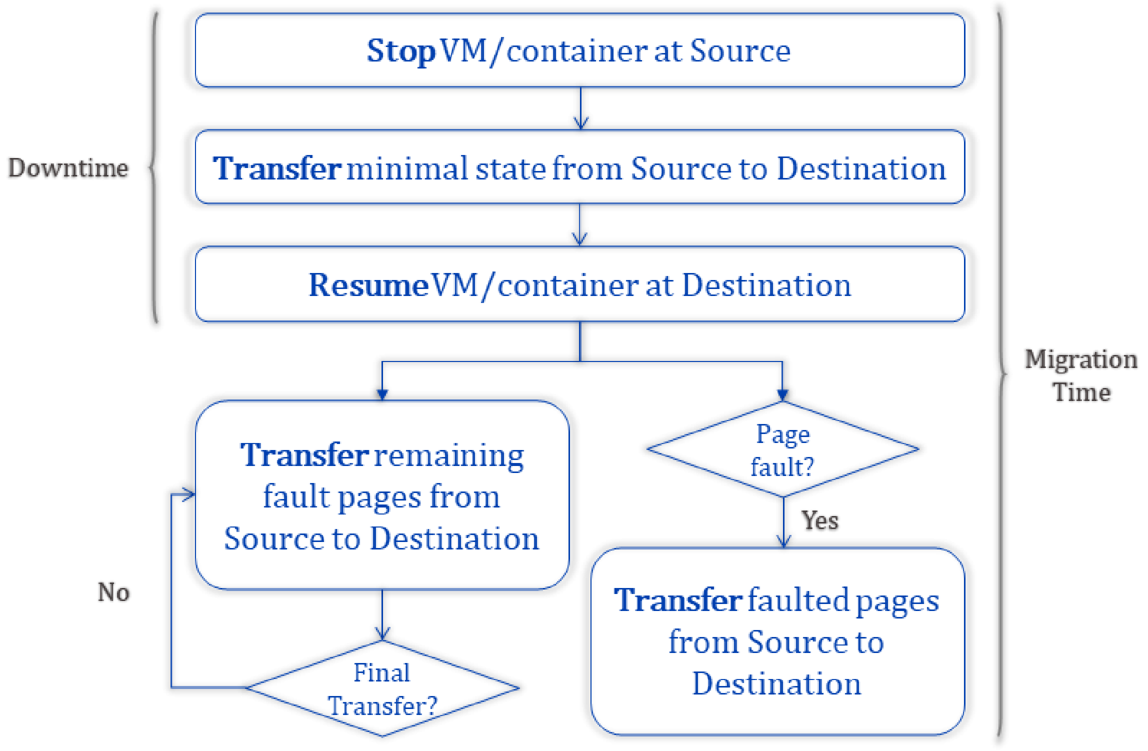
Stop VM/container at Source
Transfer minimal state from Source to Destination
Resume VM/container at Destination
Downtime
Migration Time
Transfer remaining fault pages from Source to Destination
Page fault?
Yes
Transfer faulted pages from Source to Destination
Final Transfer?
No

Figure 2.4 Post-copy live migration methods

Additionally, as with cold migration, the downtime incurred by the post-copy method is independent of the rate of memory dirtying generated by the service running in the VM/container and the total amount of data to be moved to the target. In comparison, cold or pre-copy migrations preserve the source node's current state throughout migration. As a result, if the target node fails during the migration, the most recent state of a transferred VM/container is likely to be lost since the service was updated at the target after hand-off [73]. One alternative for reducing the duration of residual dependencies on the source node is to "push" the VM/container's memory pages proactively from the origin to the destination edge node while the virtual machine or container is still running on the target. As a result, each memory page is delivered only once, and page transfers can be transmitted more quickly using the active pushing process rather than relying solely on demand paging [72].

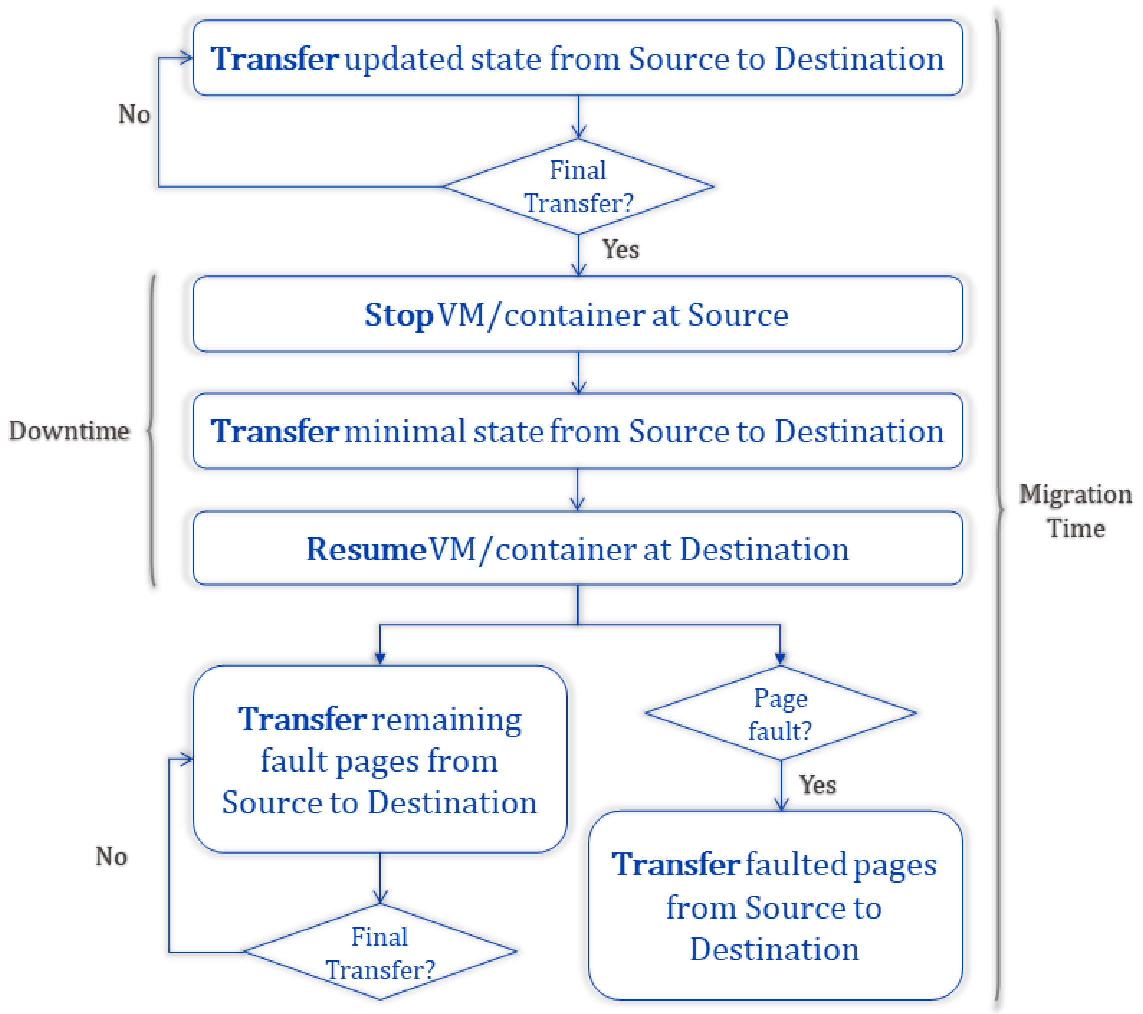
Transfer updated state from Source to Destination
No
Final Transfer?
Yes
Stop VM/container at Source
Downtime
Transfer minimal state from Source to Destination
Migration Time
Resume VM/container at Destination
Transfer remaining fault pages from Source to Destination
Page fault?
Yes
No
Transfer faulted pages from Source to Destination
Final Transfer?

Figure 2.5 Hybrid-copy live migration method

2.2.2.2.3 Hybrid-Copy Live Migration Method

Both pre-copy and post-copy methods, as previously discussed, have a number of drawbacks: (1) Non-deterministic downtime occurs during the pre-copy phase, and (2) service performance during the post-copy phase is affected by faulted pages [62]. Figure 2.5 depicts the **hybrid-copy migration** method [76] that integrates pre- and post-copy methods to overcome limitations and improve capabilities. The first four stages of hybrid migration are identical to those of pre-copy migration. The entire state and then dirty pages are sent to the destination while the virtual machine or container remains operational on the source node [63], [64]. The VM/container is then paused **after the hand-off is triggered,** and its state is transmitted. The VM/container can be restarted at the target when the memory and CPU state have been delivered. The most recent VM/container's execution state and memory pages are now present in the target node.

However, pages may have been dirtied throughout the pre-copy process. Accordingly, the final phase of the hybrid method is to transfer dirty pages to the target node using the post-copy method [77]. While the post-copy method transfers the entire memory and CPU states after the hand-off, the hybrid method in the pre-copy phase sends only the generated dirty pages after the hand-off. Other studies that have focused on other techniques have measured the performance of the hybrid method and compared them to other live migration techniques [78]–[81]. In addition to these studies, researchers have used the same hybrid migration concept, generally introduced the same technique, and measured the performance by comparing hybrid with pre- and post-copy methods [63], [64]. Still, other studies have found that the hybrid method outperforms both pre- and post-copy migrations regarding downtime and migration time [62], [63].

2.2.3 Limitations of Migration Methods

2.2.3.1 Pre-Copy and Post-Copy Migration Limitations

There are two variables that affect **pre-copy** migration that consequently affect the number of dirty pages transferred: the first is the memory dirtying rate of the service hosted on the container, that is, the rate at the service updates memory pages, and the second is the size of data transmitted throughout the pre-copy stage, since the more data transmitted during that stage, the longer the service requires to transmit dirty memory [62].

Post-copy migration has two disadvantages [62]. First, since a significant portion of the target container's state and memory is unavailable immediately following the hand-off, page faults deteriorate service performance by delaying access to memory pages at the target after the VM/container is started. Since memory pages must be accessed remotely, this remote access can be incompatible with the delay-sensitive services available in the context of edge computing [82]. The second disadvantage is that this strategy spreads the VM/container's total current state across both origin and target nodes during migration. When a post-copy method is in process, the source node is still transferring memory pages to the destination; however, the restarted VM/container may update some of these pages. This can increase delays since the data should be transferred from the remote node, the source.

2.2.3.2 Hybrid-Copy Migration Limitations

Since **hybrid migration** uses pre- and post-copy approaches, the amount of data to be sent and the memory dirtying rate in the source edge node continues to influence the performance of this method. The use of pre-copy before hand-off causes the hybrid method to transfer a large amount of data, which still correlates with the dirtying page rate of services running at the source. Nonetheless, since these parameters affect the faulty page numbers after the hand-off triggers, they should increase the delay and total migration time; however, this does not have a negative impact on the hybrid approach's downtime because it only transfers minimal state and memory from the source to the destination during hand-off [62]. Furthermore, the hybrid method has the same limitations as the post-copy approach stated in this subsection. IoT and 5G applications demand a novel design that synchronizes the source and destination more frequently than the current methods during migration.

2.3 Downtime and Migration Time Reduction

Researchers employ various strategies to reduce migration time. In recent years, the focus has been on various prediction-based approaches, e.g., [16], [32], [34], [40], [41], [65], [66], and data transfer approaches, e.g., [2], [29], [31], [33], [35], [67], [68] that are used to reduce migration time. All the stated references used in this chapter address the **fog/edge computing domain**. The fog/edge computing domain has been the primary focus of recent work. This is because earlier efforts focused on clouds and had to be modified for edge computing since edge computing requirements differ from cloud computing [1].

Migration within the fog/edge computing domain differs from a cloud data center since the network is typically a WAN [1]. The applications also differ, e.g., AR applications. The latency is higher than what would be found in a cloud network [1]. Furthermore, the path of a mobile device user is not always clear. The implication is that the mobile device may be out of range of any server that it communicates with. This may result in downtime that starts before the handover. Recent work on migration focuses on reducing the data transferred, predicting trajectories of mobile devices, and predicting memory dirty rates to reduce downtime and migration time.

2.3.1 Reducing Downtime and Migration Time by Reducing the Data Transfer Amount

Data transfer approaches focus on reducing the amount of data transferred during hand-off to fundamentally reduce downtime, resulting in significantly reduced latency and migration time and, as a result, improved user QoE. To reduce transfer size during hand-off, Ma et al. [29], [67] proposed an improved migration technique based on the hierarchical structure of the container file system. Transferring the basic image only at the beginning of the migration, followed by iterative memory difference, can help reduce transfer size during hand-off. In the best-case scenario, they had 2.7 seconds of downtime. Machen et al. [35] proposed a layered migration framework that supports container and virtual machine technologies. The framework breaks the application into various layers and transfers only the missing layers to the destination. This layered method can reduce downtime by transmitting less data from the source to the destination during hand-off. They assessed their performance using various applications, such as video streaming and gaming servers. They significantly lowered overall migration times but with a two-second average downtime for a small container. This time is still unacceptable for delay-sensitive applications, such as the stated head-mounted AR application.

Farris et al. [68] used the pre-copy method in stateless migration to achieve low latency by transferring as little data as possible during hand-off. Data is sent prior to hand-off to achieve lower latency using the pre-copy migration method. In the best-case scenario, their experiments had more than a second latency. To reduce downtime, Addad et al. [31] used memory, partial, and full migration strategies with predefined and non-predefined user paths, as well as different numbers of pre-copy iterations. This is accomplished and considers users' mobility patterns by storing the container files in a shared storage pool accessible to all edge nodes, which proactively sends data, resulting in less data to transfer during hand-off. Performance evaluation is done using video streaming and small-size containers, but they have more than one second of downtime in their best-case scenario.

Zhou et al. [33] propose a hardware accelerator that, instead of software, uses hardware implementation to expedite data transmission reduction computations and, as a result, service migration. In their best-case scenario, they achieve about 300 milliseconds (ms) of downtime. In another study, Puliafito et al. [2] stated that an AR application that uses a smart helmet should have a maximum end-to-end latency of 20 ms. This study used pre-copy and proactively sent data using compression before hand-off, resulting in less data transfer during hand-off. Despite their proposal, they still have 3.67 seconds of downtime in the best-case scenario.

Most papers on reducing downtime aim to reduce the transfer rate during hand-off by reducing data transfer during that period.

2.3.2 Reducing Downtime and Migration Time Using Optimization and Machine Learning Techniques

Prediction-based methods are used to reduce downtime that starts before the hand-off. Handover can be triggered when the value of a metric, such as the Received Signal Strength Indicator (RSSI), falls below a pre-set threshold value. The ideal threshold value would be sufficient for the migration process to occur with minimal service disruption. However, there are other factors to be considered, e.g., the load on the wireless links and the speed of the UE. This makes it difficult to determine a fixed threshold value. A poor threshold value results in the handover occurring too late or too early. Prediction-based approaches consider multiple metrics as well as a prediction of the UE's movement to determine the migration destination. For example, Ngo et al. [66] used the pre-copy method and proactively sent data prior to hand-off by memory checkpointing before and during the migration phase to determine the time to trigger hand-off and the destination edge node. In their study, the downtime period and migration start simultaneously, unlike other studies where the migration starts earlier than the downtime. The downtime period in this work may start during the handover time, which represents the time that the service is unavailable as the result of a weak RSSI signal. Although this re-definition influences results, in the best-case scenario, their experiments show about 7 seconds of total downtime and about 300 ms of end-to-end delay, even with this re-definition. This also makes it difficult to compare with most work where downtime corresponds to handover.

Yang et al. [34] developed a multi-tier MEC server deployment framework based on the pre-copy method to predict the next node based on the UE's position, direction, speed, and delay requirements. Their experiments demonstrate several seconds of downtime when using various prediction-based techniques with varying UE speeds. Majeed et al.

[65] use four regression models to predict offloading time in MEC using various runtime and offline metrics such as CPU and disk utilization, network bandwidth, and container image size to reduce end-to-end latency compared to their other evaluated ML approaches. In the best-case scenario, they achieved multi-seconds of delay by proactively transferring data prior to hand-off and employing the pre-copy method.

Pomalo et al. [32] used K-Nearest Neighbor, Logistic Regression, Random Forest, and XGBoost machine learning (ML) algorithms. They aim to predict the amount of time the migration service should start in advance of the new edge node in order to continue service without interruptions based on UE's route and speed. Their results were evaluated by comparing these ML approaches to determine which method best handled service continuity. This study requires the path type, e.g., main road, highway, and train, to predict the proper migration start time; otherwise, their approach produces long downtimes. Furthermore, they stated that their approach still has downtime, but did not specify its exact amount. The research mentioned above reduces downtime when compared to other ML approaches. The downtime is reduced because handover is triggered earlier by taking into consideration factors such as the load on the wireless links and the speed of the UE.

The authors of [40] propose an adaptive VM monitoring strategy for migrating a single VM using pre-copy and post-copy methods. They developed an autoregressive model to predict the dirty memory rate and use it to reduce migration downtime, migration time, and the data transfer amount. The model's output value is determined by a linear combination of a stochastic variable and the previous model's values. Tang et al. [16] use reinforcement learning with deep Q-learning container migration to propose power consumption, delay, and migration cost models. They compare their algorithm to other ML algorithms, including static threshold, median absolute deviation, and interquartile range regression. The authors of [41] use the pre-copy migration time, downtime, round-trip time, and energy consumption models to reduce delay and energy consumption in wireless connections with a bandwidth manager.

2.3.3 Limitations

The pre-copy migration method is mostly used for migration. The downtime for latency-sensitive and real-time applications is longer than expected when using the pre-copy method. As stated in this section, most research proposes a technique to overcome the lengthy pre-copy downtime. However, the limitation of the work that focuses on reducing downtime is that the assumption is that pre-copy is used. Most of the work does not focus on reducing downtime during the hand-off.

2.4 Multi-Service Migration in Edge Computing

Application services can be run on single or multiple VMs or containers. Regardless of whether the services are hosted on VMs or containers, application services must be migrated together. With the increasing use of multi-containerized applications [5] where there is dependency among tasks, it is crucial to ensure that all application containers finish their migration in a timely fashion to prevent waiting time and further delays in **latency-sensitive** and **real-time** applications. This is especially important in the presence of **dependent tasks** that can be found in more than 75% of multi-containerized applications [12].

Recent research includes migrating multiple services [13]–[15], [17]–[24]. Delays in multi-container migration can occur, as some application containers can complete their migration sooner than others, requiring them to wait for the migration completion of the remaining containers before continuing their tasks and remaining operational. Suppose a virtual tour guide with an Augmented Reality (AR) application on a mobile head-mounted device uses containerized microservices. As a basic containerized AR application, one microservice captures the environment from the device, and another microservice renders the AR data to the device. Each of the stated microservices can make use of single or multiple containers. This application needs ultra-low response time for smooth functionality. For such real-time applications, [3] recommends a maximum response time of 17 ms for an AR application to maintain end-user quality of experience at a desired level.

The rest of this section describes migration approaches that use optimization and ML techniques for migration.

2.4.1 Multiple Service Migration using Optimization and Machine Learning Techniques

This section describes migration approaches that employ optimization and ML techniques to migrate multiple services in parallel. The authors of [14] use Geometric Programming to assign transfer and compression rates to each VM in order to reduce the total migration time of multiple VMs. They consider parameters including VM size, memory dirtying rate, transfer rate, and compression ratio of VMs. They evaluate their experiments with up to seven VMs and nine pre-copy iterations. Maheshwari et al. [18] developed a cost model for multi-container migration that considers container size, number of containers, memory dirtying rate, bandwidth, and load at an edge node that supports mobility. The authors include migration time and computation time in their migration cost model and employ a Min-Max model to minimize the migration cost. Liu et al. [22] build a migration cost model by predicting the memory dirtying rate and employing parameters such as VM size and transfer rate. They use a cost model for multi-VM and employ adaptive bandwidth allocation to reduce migration costs.

In another study, Sun et al. [21] use an M/M/C/C queuing migration model to quantify performance metrics, e.g., the average waiting time of each migration request. The objective is to optimize migration time and minimize downtime for multiple-VM migration. Satpathy et al. [15] compared migration model performance for multiple VMs, including comparisons based on VM size, memory dirtying rate, and available bandwidth. Using a platform based on a software-defined network (SDN), He et al. [17] evaluated the performance of multiple VM migration models. They consider migration time and downtime to be two of their most important criteria. To balance server load, Zhang et al. [13] proposed a set of algorithms for optimizing load balancing and migrating multiple containers among cloud servers in order to balance server load. Their primary focus is load balancing; migration would occur as a result of server load balancing with the migration time model.

Similarly, Forsman et al. [23] present a load-balancing solution that reduces the migration cost of multiple VMs. Initially, they identify the overloaded and underutilized nodes and then estimate the loads by the algorithm described in [83] to balance the load. They consider migration time, downtime, and data transfer amount in their cost model. In another study, Satpathy et al. [19] propose a VM placement strategy for cloud servers while modeling multiple-VM migration with downtime and migration time. Considering power constraints, Elsaid et al. [20] examine the migration cost of multiple VMs using migration time and power consumption. Cerroni [24] investigates the cost of migrating multiple VMs based on downtime and migration time using the Markovian model. The network overhead and throughput degradation are also components of the migration cost model.

2.4.2 Service Migration Performance Analysis and Comparison

This section reviews migration analyses and comparisons of pre-copy, post-copy, and hybrid copy methods [37], [38], [42]. The papers in this section compare and discuss the various migration methods without proposing an ML technique. Mandal et al. [42] investigate the effects of bandwidth provisioning on pre-copy downtime and migration time for different bandwidth provisioning. They also investigate the effects of varying the number of pre-copy iterations on migration time. The authors of [37], [38] investigate how the input parameters of the pre-copy, post-copy, and hybrid-copy migration methods affect the output parameters. Input parameters include the number of pre-copy iterations, various transfer rates, and various memory dirtying rates, while output parameters include downtime, migration time, and the amount of data transferred during the migration process.

2.4.3 Limitations

Most studies have prioritized reducing migration time, including those discussed in this section that examined multi-service migration. **One limitation** of these methodologies is their exclusive focus on **minimizing migration times,** while **ignoring minimizing downtimes, of multiple containers**. The issue is that the bandwidth allocated to each container is assumed to be constant throughout the migration process, as in [14]. Each container's task may differ, resulting in a distinct volume of data that must be synchronized between the source and destination during the migration process. As a result, regardless of the volume of data to be transferred, the **bandwidth allocation would remain constant** throughout the downtime for the rest of the migration.

The main point is that each container generates a different amount of data that must be transferred during downtime. This phenomenon can cause **varying** lengths of **downtime and service unavailability**. As indicated in the introductory section of this chapter, containers of an application can be dependent on each other. Containers with longer downtimes can cause longer periods of service unavailability for users than those with shorter downtimes. As previously stated, real-time applications such as the AR application require a response time of no more than 17 ms [3]. However, most research has resulted in significantly **longer downtimes,** as mentioned in section 2.3. As a result, it is critical to consider the **reduction of downtimes** for multiple services as well as the reduction of migration times.

Bandwidth utilization of migration also can be improved when migrating multiple containers. As described in section 2.2, the entire memory of the container is transferred first, followed by the transfer of modified memories that occur during the migration process. We observed that in most research studies when a container is allocated a specified bandwidth during the migration, said container utilizes the assigned bandwidth for the duration of the migration process and remains unchanged thereafter. Because of the size of the container memory, the transfer of the initial migration stage can be time-consuming. However, studies have shown that dirty memory data is less than container memory data [29], [62]. As a result, in the intermediate stages of the migration, it is possible to **transfer dirty memory using less bandwidth**. Therefore, optimizing the utilization of available bandwidth can be accomplished by **modifying bandwidth allocation** to containers belonging to different users at the **initial, intermediate, and final stages of migration**. When the source is transmitting a large amount of data, particularly during the early stages of migration [29], [62], more bandwidth is required to allow for the efficient transfer of said data. According to existing literature, when the source transmits dirty memory during the intermediate stages of migration, the volume of data to be transferred is relatively small [29], [62]; consequently, we can use only a small amount of bandwidth. Ultimately, there may be a requirement for **increased bandwidth** towards the final stage of the migration process, particularly during **downtime**. As such, the utilization of higher bandwidth during the final stage of migration has the potential to mitigate the duration of such downtime.

Consequently, rather than maintaining uniform bandwidth allocation for each container, **modifying the bandwidth allocation** for each container during multiple container migrations can **potentially reduce both migration time and downtime**.

2.5 Migration Modeling in Edge Computing

Migration modeling is used by researchers in [13]–[24], [37]–[42] to understand future system behavior. The primary goal of the stated models is to determine the cost or outcome of migration by defining appropriate input parameters. Accurately characterizing and predicting the outputs is essential for a migration model [83]. For example, in this chapter, we will apply equation (2.6), which is the migration time model, to deduce that increasing the transfer rate reduces migration time. In this example, the model's output is migration time, and one of the input parameters is transfer rate.

Modeling the migration requires several input parameters, including the size of VM/container memory, memory dirtying rate, and transfer rate. The output parameters of the migration models are primarily migration time, downtime, and data transfer amount during migration, which can be defined as the model's cost or outcome to be predicted by models.

Service migration for **multi-container** applications is critical for mobile devices in edge computing [5], [11]. This section reviews migration models for multiple VMs/containers to support mobility. We start with migration models for a single VM/container since these serve as the foundation for multiple VM/container models, and then we review migration models for multiple VMs/containers.

Several studies on live migration modeling have been conducted over the last decade. Most of these migration models base their research on a single VM or container migration [16], [37]–[42]. These models primarily focus on downtime and migration time and compare live migration methods based on various input values, such as pre-copy iterations, memory dirtying rate, bandwidth, and VM/container size, using datasets, implementations, or their assumptions. A subset of these papers provides models and compares various parameters of live migration methods [16], [37], [38], [40], while others employ estimation and optimization techniques to reduce migration costs, such as downtime and migration time [39], [41], [42].

Despite extensive research on modeling the migration of a single VM/container, few authors focus on modeling multiple-VM/multiple-container migration [13]–[15], [17]–[24]. Some of these studies focus on the number of VMs/containers and provisioned bandwidth in addition to the stated input values. Most of these studies focus on modeling **multiple-VM migration** [14], [15], [17], [19]–[24] in order to optimize the migration performance of multiple VMs, while authors in [13], [18] focus on modeling **multiple-container migration**.

Table 2.1: Symbols and Definitions

Parameter	Description
V_j^{Pre}	Dirty memory generated during round i for single $VM/container$ using pre-copy method, $\forall\ i \in \{1, \dots, m\}$
V_s^{Pre}	Memory amount in the stop-and-copy phase for a single $VM/container$ using pre-copy migration method
M	Memory size of a $VM/container$ (single $VM/container$ migration)
$\bar{r}$	Average transfer rate (bandwidth) for $VM/container$ in pre-copy migration method (single $VM/container$ migration)
$\bar{d}$	Average memory dirtying rate for $VM/container$ in pre-copy migration method (single $VM/container$ migration)
t_i^{Pre}	Time to transfer V_i^{Pre} in pre-copy migration, $\forall\ i \in \{1, \dots, m\}$
λ	$\bar{d}/\bar{r}$, for single $VM/container$ migration
M_j	Memory size of any $VM_j/Container_j$, $\forall\ j \in \{1, \dots, p\}$
$\bar{r}_j$	Average transfer rate (average bandwidth) available for any $VM_j/Container_j$ in pre-copy migration method
$V_{i,j}^{Pre}$	Dirty memory generated during round i for any $VM_j/Container_j$ in pre-copy, $\forall\ j \in \{1, \dots, p\}$ and $\forall\ i \in \{1, \dots, m\}$
$V_{s,j}^{Pre}$	Memory amount in the stop-and-copy phase for any $VM_j/Container_j$ in pre-copy migration method
$\bar{d}_j$	Average memory dirtying rate for any $VM_j/Container_j$ in pre-copy migration method
$t_{i,j}^{Pre}$	Time to transfer $V_{i,j}^{Pre}$ in pre-copy migration, $\forall\ j \in \{1, \dots, p\}$ and $\forall\ i \in \{1, \dots, m\}$
$TM_j^{PreCopy}$	Total migration time for any $VM_j/Container_j$ for the pre-copy migration method
$TD_j^{PreCopy}$	Downtime for any $VM_j/Container_j$ for both pre-copy migration method
λ_j	$\bar{d}_j/\bar{r}_j$, $\forall\ j \in \{1, \dots, p\}$
B	Total maximum bandwidth reserved for the entire migration between two edge nodes
$TA_j^{PreCopy}$	Amount of data to be migrated during migration, for any $VM_j/Container_j$ in pre-copy method
τ	The inter-iteration delay in pre-copy migration method
$TD_j^{HybridCopy}$	Downtime for $VM_j/Container_j$ for hybrid-copy method
$V_{c,j}^{HybridCopy}$	CPU state size of $VM_j/Container_j$ which is part of the memory size for hybrid-copy method
$T_j^{PageFault}$	Time required to process and copy a single memory page
PS_j^{Mem}	Memory page size
T_j^{fp}	Page fault processing time to locate the page fault and request the page from the source
PF_j	Probability of page faults
$TM_j^{HybridCopy}$	Total migration time for $VM_j/Container_j$ for hybrid-copy method
w_j	Number of memory pages (window size) will be transferred when a page fault occurs for $VM_j/Container_j$
$TA_j^{HybridCopy}$	Total data amount to be transmitted during hybrid-copy migration for $VM_j/Container_j$
$TM_j^{Post-NPF-HC}$	Part of migration time for $VM_j/Container_j$ if there is No Page Fault during the post-copy phase of hybrid-copy migration
$TM_j^{Post-PF-HC}$	Part of migration time for $VM_j/Container_j$ if Page Fault occurs in the post-copy phase of hybrid-copy migration

This section describes existing **pre-copy**, **post-copy**, and **hybrid-copy migration models** that use average parameter values derived from [14]–[18], [21], [22], [37]–[42]. Although these models are not completely identical, the pre-copy migration for multiple VMs/containers will be modeled using the same concept as [14]–[18], [21], [22], [37]–[42] in this chapter. Therefore, **all equations in this chapter are representative of** [14]–[18], [21], [22], [37]–[42]. Downtime and migration time are the two primary parameters for migration modeling analysis [14]. Downtime is an important performance metric for end-users, which must be as low as possible to avoid service interruptions [14]. The total migration time must be as short as possible because it consumes computational and network resources from both the origin and the target edge nodes [14]. The amount of data that must be transmitted during the migration process of multiple VMs/containers is also considered in this document. We will examine the **downtime**, **migration time**, and **transferred data during migration** for the pre-copy, post-copy, and hybrid-copy migration models, which utilize a single VM/container and, the pre-copy model, which uses multiple VMs/containers.

2.5.1 Single VM/container Migration Models

2.5.1.1 Pre-Copy Migration Model

This section describes existing pre-copy, post-copy, and hybrid-copy migration models derived from references [14]–[18], [21], [22], [37]–[42] that use single VMs/containers. Table 1 defines parameters and their notations for the models described in this paper. In the table, the M, $\bar{d}$, and $\bar{r}$ represent the VM/container memory size, average memory dirtying rate, and average transfer rate (average bandwidth) available during migration for any VM/container, respectively. The parameters specified affect migration time ($TM^{PreCopy}$) and downtime ($TD^{PreCopy}$). Higher M and $\bar{d}$ levels increase migration time and downtime, while higher $\bar{r}$ levels decrease migration time and downtime.

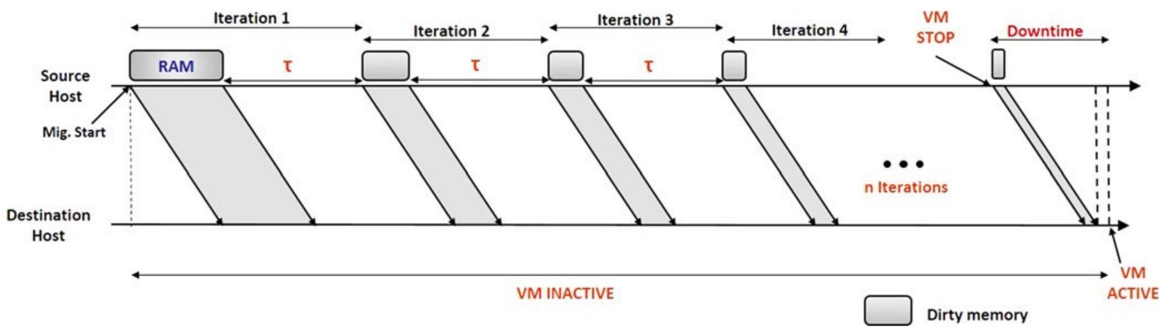

Figure 2.6 Pre-copy iterations (rounds) [42]

During the first iteration of the pre-copy method, the entire VM/container memory is transferred from the origin edge node to the target. Thus, the data transmitted throughout the first round, i.e., V_1^{Pre}, may be calculated as follows:

$$V_1^{Pre} = M \tag{2.1}$$

During pre-copy migration, the $VM/container$ remains active at its origin, making the memory dirty during the transfer process. The pre-copy iterations then transfer only the dirty memory of the previous round. The amount of data transmitted during round i for a $VM/container$ may be calculated using the subsequent equation:

$$V_i^{Pre} = \bar{d}.t_{i-1}^{Pre} \tag{2.2}$$

Once i reaches m, the final iteration, known as stop-and-copy, commences. Additionally, the required time for the data transfer of iteration i for a $VM/container$, i.e., t_i^{Pre}, may be iteratively determined using equations (2.1) and (2.2) as follows:

$$t_1^{Pre} = \frac{V_1^{Pre}}{\bar{r}} + \tau = \frac{M}{\bar{r}} + \tau \tag{2.3}$$

here τ is the inter-iteration delay shown in figure 6.

$$t_2^{Pre} = \frac{V_2^{Pre}}{\bar{r}} + \tau = \frac{\bar{d}t_1^{Pre}}{\bar{r}} + \tau = \lambda t_1^{Pre} + \tau = \lambda\left(\frac{M}{\bar{r}} + \tau\right) + \tau \tag{2.4}$$

here λ is the average memory dirtying rate divided by the average transfer rate, $\bar{d}/\bar{r}$, for a VM/container. Thus, t_i^{Pre} can be calculated as follows:

$$t_i^{Pre} = \frac{V_i^{Pre}}{\bar{r}} + \tau = \lambda t_{i-1}^{Pre} + \tau = \frac{M}{\bar{r}}\lambda^{i-1} + \tau\left(\frac{1-\lambda^i}{1-\lambda}\right) \tag{2.5}$$

Therefore, the migration **downtime** for a VM/container, i.e., $TD^{PreCopy}$, may be calculated as:

$$TD^{PreCopy} = \frac{M}{\bar{r}}\lambda^m + \lambda\tau\left(\frac{1-\lambda^m}{1-\lambda}\right) \tag{2.6}$$

By applying the downtime model, it can be inferred that reducing the memory size of a VM/container, M, will result in a decrease in downtime. Similar outcomes can be deduced upon decreasing the value of λ. Furthermore, increasing the transfer rate would result in a reduction in downtime.

The total **migration time** for a VM/container, i.e., $TM^{PreCopy}$ with m number of pre-copy transfer rounds followed by a final stop-and-copy round, is given by:

$$TM^{PreCopy} = \frac{M}{\bar{r}}\frac{1-\lambda^m}{1-\lambda} + \tau\frac{m(1-\lambda)-\lambda(1-\lambda^{m+1})}{\left(1-\lambda_j\right)^2} + \frac{M_j}{\bar{r}}\lambda^m + \lambda\tau\left(\frac{1-\lambda^m}{1-\lambda}\right) \tag{2.7}$$

By utilizing the migration time model, it can be inferred that a decrease in VM/container memory size and an increase in transfer rate result in a reduction in migration time. A decrease in the value of λ correlates with a reduction in migration time. Given that λ is defined as the memory dirtying rate divided by transfer rate, it is reasonable to anticipate reduced migration time when the memory dirtying rate is decreased.

The **total amount of data** to be sent during the pre-copy migration for a VM/container, i.e., $TA^{PreCopy}$, is given by:

$$\begin{aligned} TA^{PreCopy} &= \left(\sum_{i=1}^{m} V_i^{Pre}\right) + V_s^{Pre} = M + \left(\sum_{i=2}^{m} \bar{d}t_{i-1}^{Pre}\right) + \bar{d}t_m^{Pre} \\ &= M + M\lambda\frac{1-\lambda^m}{1-\lambda} + \tau\bar{d}\frac{m(1-\lambda)-\lambda(1-\lambda^{m+1})}{(1-\lambda)^2} + M\lambda^m \\ &+ \bar{d}\tau\left(\frac{1-\lambda^m}{1-\lambda}\right) \end{aligned} \tag{2.8}$$

where V_i^{Pre} is the dirty memory generated throughout iteration i, V_s^{Pre} is the memory amount in the stop-and-copy phase, t_{i-1}^{Pre} is the time to transfer V_i^{Pre} during iteration i, and $\bar{d}$ is the average memory dirtying rate, for a $VM/container$ in pre-copy migration method.

Equations (2.6, 2.7, 2.8), subject to:

$$\sum_{j=1}^{p} \bar{r} \leq B \ and \ \lambda < 1 \tag{2.9}$$

By applying the data transfer amount model, a reduction in the total amount of transferred data can be observed by reducing both M and λ. See section 4.3 for further information.

2.5.1.2 Post-Copy Migration Model

The migration **downtime** for a VM/container, i.e., $TD^{PostCopy}$ is the time to transfer the CPU states from the source to the destination and can be calculated as:

$$TD^{PostCopy} = \frac{V_c^{PostCopy}}{\bar{r}} \tag{2.10}$$

where $V_c^{PostCopy}$ is the CPU state size of a VM/container and part of the memory size. The previous equation implies that the duration of the downtime associated with the post-copy method ought to be very small.

The total **migration time** for a VM/container, i.e., $TM^{PostCopy}$, is given by:

$$\begin{aligned} TM^{PostCopy} = (1 - w.PF)\frac{\left(M - V_c^{PostCopy}\right)}{\bar{r}} \\ + PF\left(T^{PageFault} + \frac{PS^{Mem}(w-1)}{\bar{r}}\right)\frac{\left(M - V_c^{PostCopy}\right)}{PS^{Mem}} \end{aligned} \tag{2.11}$$

where PF is the probability of page faults of a VM/container, and w is the number of memory pages that will be transferred when a page fault occurs for a VM/container during post-copy migration method. The page fault time, i.e., $T^{PageFault}$, is the amount of time required to transfer and process a single memory page of a VM/container after hand-off in the post-copy method and is given by:

$$T^{PageFault} = \frac{PS^{Mem}}{\bar{r}} + T^{fp} \tag{2.12}$$

where PS^{Mem} is the memory page size of a VM/container and T^{fp} is the page fault processing time to locate the page fault and to request the page from the source during the post-copy migration method.

Thus, the **total amount of data** to be sent during the post-copy migration for a VM/container, i.e., $TA^{PostCopy}$, is given by:

$$TA^{PostCopy} = M \tag{2.13}$$

The post-copy data transfer model demonstrates a reduction in the amount of data that needs to be transmitted compared to the pre-copy data transfer.

2.5.1.3 Hybrid-Copy Migration Model

The migration **downtime** for a VM/container, i.e., $TD^{HybridCopy}$ is the time to transfer the CPU states from the source to the destination and can be calculated as:

$$TD^{HybridCopy} = \frac{V_c^{HybridCopy}}{\bar{r}} \tag{2.14}$$

where $V_c^{HybridCopy}$ is the CPU state size of a VM/container and part of the memory size.

The hybrid-copy migration method is divided into the pre-copy phase (before hand-off) and the post-copy phase (after hand-off).

The page fault time, i.e., $T^{PageFault}$, is the amount of time required to transfer and process a single memory page of a VM/container after hand-off in post-copy phase of hybrid-copy migration and is given by:

$$T^{PageFault} = \frac{PS^{Mem}}{\bar{r}} + T^{fp} \tag{2.15}$$

where PS^{Mem} is the memory page size of a VM/container and T^{fp} is the page fault processing time to locate the page fault and to request the page from the source during the post-copy phase of hybrid-copy method.

Further, the total hybrid-copy **migration time** for a VM/container, i.e., $TM^{HybridCopy}$, with m number of pre-copy phase transfer rounds, followed by a stop-and-copy round, and a post-copy phase, is given by:

$$TM^{HybridCopy} = TM^{PrePhase} + TM^{PostPhase} + TD^{HybridCopy} \tag{2.16}$$

The $TM^{PrePhase}$ is calculated similarly to the pre-copy method for a VM/container given by:

$$TM^{PrePhase} = \frac{M}{\bar{r}}\frac{1-\lambda^{m}}{1-\lambda} + \tau\frac{m(1-\lambda)-\lambda(1-\lambda^{m+1})}{\left(1-\lambda_j\right)^2} \tag{2.17}$$

and $TM^{PostPhase}$ for a VM/container is given by:

$$TM^{PostPhase} = TM^{PostPhase-NPF-HC} + TM^{PostPhase-PF-HC} \tag{2.18}$$

where $TM^{PostPhase-NPF-HC}$ is the part of migration time for a VM/container if no page fault occurs during the post-copy phase. The $TM^{PostPhase-PF-HC}$ is a part of migration time for a VM/container if a page fault occurs during the post-copy phase of the hybrid-copy migration method.

$$TM^{PostPhase-NPF-HC} = (1 - w.PF)\frac{\left(V_s^{Pre} - V_c^{HybridCopy}\right)}{\bar{r}} \tag{2.19}$$

where PF is the probability of page faults of a VM/container, and w is the number of memory pages which will be transferred when a page fault occurs for a VM/container during post-copy phase of hybrid-copy migration. Correspondingly, V_s^{Pre} is the amount of data generated in the last round of the pre-copy phase and can be calculated for a VM/container as follows:

$$V_s^{Pre} = M\lambda^m + \bar{d}\tau\left(\frac{1-\lambda^m}{1-\lambda}\right) \tag{2.20}$$

And:

$$TM^{PostPhase-PF-HC} = PF\left(T^{PageFault} + \frac{PS^{Mem}(w-1)}{\bar{r}}\right)\frac{\left(V_s^{Pre} - V_c^{HybridCopy}\right)}{PS^{Mem}} \tag{2.21}$$

Then:

$$\begin{aligned} TM^{HybridCopy} &= \frac{M}{\bar{r}}\frac{1-\lambda^m}{1-\lambda} + \tau\frac{m(1-\lambda) - \lambda(1-\lambda^{m+1})}{\left(1-\lambda_j\right)^2} \\ &+ (1 - w.PF)\frac{\left(V_s^{Pre} - V_c^{HybridCopy}\right)}{\bar{r}} \\ &+ PF\left(T^{PageFault} + \frac{PS^{Mem}(w-1)}{\bar{r}}\right)\frac{\left(V_s^{Pre} - V_c^{HybridCopy}\right)}{PS^{Mem}} \\ &+ \frac{V_c^{HybridCopy}}{\bar{r}} \end{aligned} \tag{2.22}$$

The **total amount of data** to be sent during the hybrid-copy migration for a VM/container, i.e., $TA^{HybridCopy}$, is given by:

$$TA^{HybridCopy} = M + M\lambda\frac{1-\lambda^m}{1-\lambda} + \tau\bar{d}\frac{m(1-\lambda) - \lambda(1-\lambda^{m+1})}{(1-\lambda)^2} + M\lambda^m + \bar{d}\tau\left(\frac{1-\lambda^m}{1-\lambda}\right) \tag{2.23}$$

By comparing the models of the hybrid-copy method with pre-copy and post-copy methods, it is evident that the utilization of the hybrid-copy method results in reduced post-copy downtime and approximately expedited pre-copy migration time at the same time. As a result, the hybrid-copy method combines the benefits of both pre-copy and post-copy methods.

2.5.2 Multiple VM/containers Migration Model

This section describes existing pre-copy migration models derived from references [14]–[18], [21], [22], [37]–[42] that use multiple VMs/containers. The M_j is the memory size of any $VM_j/container_j$. The migration **downtime** for any $VM_j/container_j$, i.e., TD_j^{Pre}, using equations (2.1) to (2.6), may be calculated as:

$$TD_j^{Pre} = \frac{M_j}{\bar{r}_j}{\lambda_j}^m + \lambda_j\tau\left(\frac{1-{\lambda_j}^m}{1-\lambda_j}\right) \tag{2.24}$$

where λ_j is the average memory dirtying rate divided by the average transfer rate, $\bar{d}_j/\bar{r}_j$, for any $VM_j/container_j$, and τ is the inter-iteration delay shown in figure 2.6. We use a maximum value here since VMs/containers are dependent on, and interact with, one another, and some must wait for others to respond to each user. The maximum amount of downtime during pre-copy migration is expressed as follows:

$$T_{Downtime}^{Pre} = max\{TD_1^{Pre}, \quad TD_2^{Pre}, TD_3^{Pre}, \quad \dots, \quad TD_p^{Pre}\} \tag{2.25}$$

where p is the number of VMs/containers.

The **migration time** for every $VM_j/Container_j$, i.e., TM_j^{Pre} with m number of pre-copy transfer rounds followed by a final stop-and-copy round, is given by:

$$TM_j^{Pre} = \frac{M_j}{\bar{r}_j}\frac{1-\lambda_j{}^m}{1-\lambda_j} + \tau\frac{m\left(1-\lambda_j\right)-\lambda_j\left(1-\lambda_j{}^{m+1}\right)}{\left(1-\lambda_j\right)^2} + \frac{M_j}{\bar{r}_j}\lambda_j{}^m + \lambda_j\tau\left(\frac{1-\lambda_j{}^m}{1-\lambda_j}\right) \tag{2.26}$$

The maximum migration time when the $\bar{r}_j$ network transfer rate is assigned for each $VM_j/container_j$ in pre-copy migration can be expressed as:

$$T_{migration}^{Pre} = max\{TM_1^{Pre},\ TM_2^{Pre}, TM_3^{Pre}, \dots,\ TM_p^{Pre}\} \tag{2.27}$$

where p is the number of VMs/containers.

Accordingly, the **total data transfer amount** during **migration** for any $VM_j/container_j$, i.e., TA_j^{Pre}, can be calculated as:

$$\begin{aligned} TA_j^{Pre} &= \left(\sum_{i=1}^{m} V_{i,j}^{Pre}\right) + V_{s,j}^{Pre} = M_j + \left(\sum_{i=2}^{m}\bar{d}_j t_{i-1,j}^{Pre}\right) + \bar{d}_j t_{m,j}^{Pre} \\ &= M_j + \left(\bar{d}_j\sum_{i=2}^{m} t_{i-1,j}^{Pre}\right) + \bar{d}_j t_{m,j}^{Pre} \\ &= M_j + M_j\lambda_j\frac{1-\lambda_j{}^m}{1-\lambda_j} + \tau\bar{d}_j\frac{m\left(1-\lambda_j\right)-\lambda_j\left(1-\lambda_j{}^{m+1}\right)}{\left(1-\lambda_j\right)^2} \\ &\quad + M_j\lambda_j{}^m + \bar{d}_j\tau\left(\frac{1-\lambda_j{}^m}{1-\lambda_j}\right) \end{aligned} \tag{2.28}$$

where $V_{i,j}^{Pre}$ is the dirty memory generated throughout iteration i, $V_{s,j}^{Pre}$ is the memory amount in the stop-and-copy phase, $t_{i-1,j}^{Pre}$ is the time to transfer $V_{i,j}^{Pre}$ during iteration i, and $\bar{d}_j$ is the average memory dirtying rate, for any $VM_j/container_j$ in pre-copy migration method.

The total transferred data for all VMs/containers throughout the migration process is represented by the following equation:

$$Data_{migration}^{Pre} = \{TA_1^{Pre} + TA_2^{Pre} + TA_3^{Pre} + \cdots + TA_p^{Pre}\} \tag{2.29}$$

Equations (2.24) to (2.28) subject to:

$$\sum_{j=1}^{p} \bar{r}_j \leq B \ and \ \lambda_j < 1 \tag{2.30}$$

2.5.3 Limitations

The models provided need to be more accurate to represent real-world scenarios.

Most research uses average input parameter values to analyze the migration of multiple services, as found in [13]–[24], [37]–[42]. This research uses the **pre-copy** live migration method [56]. The input parameters, including **memory dirtying rate**, **transfer rate**, and **container size**, can change continuously throughout the migration process for every single container of an application, especially when **mobility** is considered. However, memory dirtying rate and container size **fluctuate** over time due to the task of each container. Figure 2.7 depicts the various memory dirtying rates during a migration process derived from the CSAP dataset [84]. As the end user's device that runs applications moves, the transfer rate can also **fluctuate**, resulting in different distances and signal strength to the base station and varying available bandwidth [85].

The problem is that while the stated input parameters can change continuously throughout the migration process, most migration models for multiple services **assume** that the **parameter values remain constant** in contrast to what we see in figure 2.7.

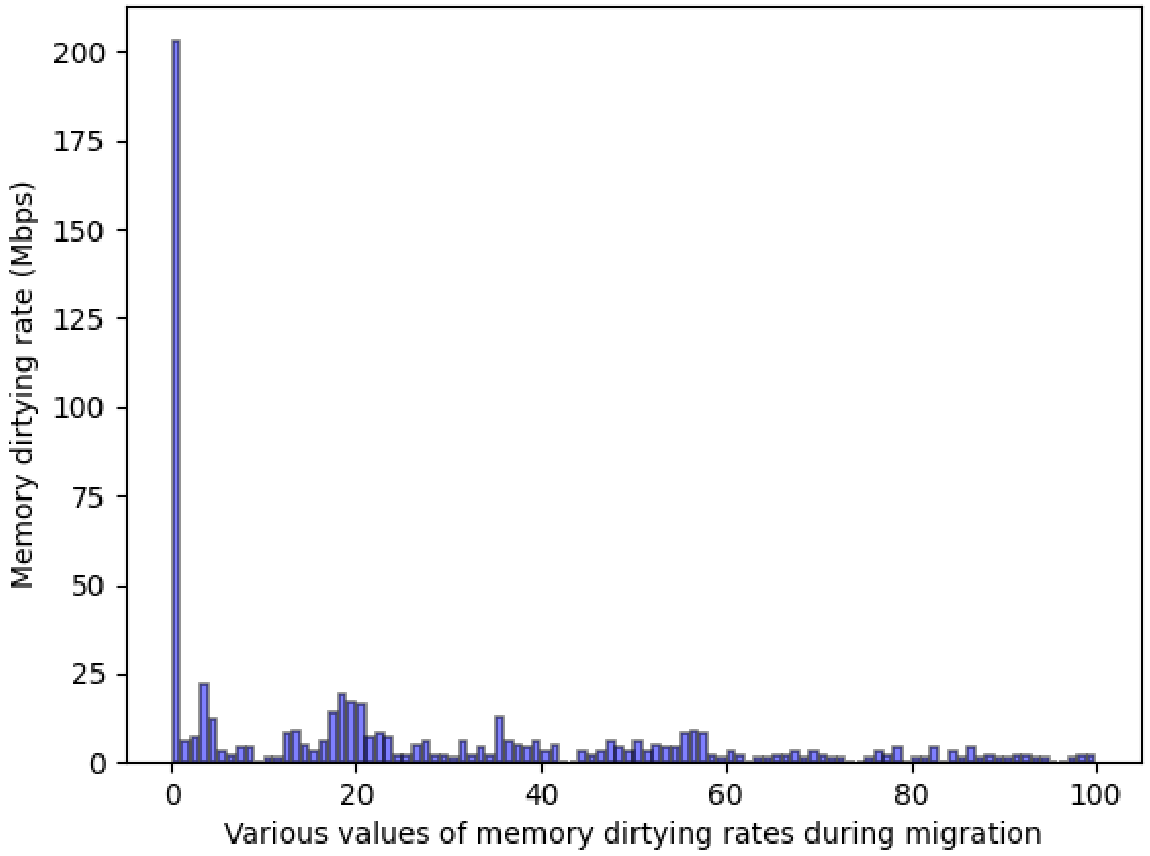

Figure 2.7 Various values of memory dirtying rates during migration

As a result, migrating mission-critical multi-containerized applications of mobile IoT devices that use only average parameter values can result in outputs that deviate from their realistic outputs. The issue with depending exclusively on average parameter values is that the results for output parameters, downtime, migration time, and data transfer amount are similar, using the same average values for input parameters. In contrast, these **results can vary for non-average parameter values** with the same average values for input parameters.

To illustrate the previously discussed **limitations of the current migration models**, consider, for instance, two migration processes with the same average parameter values (e.g., transfer rate and memory dirtying rate) but with varying values throughout the migration. In this example, if the memory dirtying rate increases at the end of the migration, it can significantly impact downtime since downtime occurs at the end of the migration [6], and this increased memory dirtying rate requires transferring a higher-than-average dirty memory rate. The same holds true if the transfer rate drops at the end of the migration and the dirtied memory data must transfer at a lower-than-average transfer rate. In both cases above, using average parameter values yields the same results, whereas using actual (non-average) parameter values yields different results. In this case, data must be transferred at a slower-than-average rate.

The same situation as described above occurs due to using average parameter values in current models, while the actual (non-average) transfer rate was lower at the beginning in this example. As a result, if input parameters are higher or lower than their average value at crucial points during the migration, they can severely affect downtime and migration time. These situations deteriorate when critical parameters are extremely higher or lower than average, and in these cases, they significantly affect downtime and migration time.

However, **these critical conditions are hidden** when constructing the edge computing environment using only average input parameter values. As a result, **new migration models in edge computing are required** to achieve more accurate and precise results for multi-container mission-critical IoT applications that consider user mobility.

Another **problem** with using non-average parameter values for the **pre-copy** migration method is that pre-copy typically uses a limited number of rounds (iterations), which is **incompatible** with constantly changing parameter values, namely memory dirtying rate. The number of iterations is typically 10-30 rounds in previously stated migration models. Since input parameters such as memory dirtying and transfer rates can fluctuate in short intervals, these parameters can **change** multiple times in every pre-copy round. Therefore, there is no single value for each input parameter in each iteration of the pre-copy migration method. Using non-average parameter values would be unreasonable under these conditions, as there is no way to use the pre-copy method to utilize every single value of input parameters throughout the migration.

Therefore, another migration method is needed with actual (non-average) input parameter values in migration models to provide a refined, rational, migration analysis, which takes into account low response time and mobility, in multi-containerized edge computing environments.

2.6 Research Gaps

With the advent of modern applications, the demand for real-time responses is becoming increasingly unavoidable. The mobility of IoT devices increases this demand. According to existing research, current migration methods are incapable of meeting the latency requirements of mission-critical applications.

To satisfy the demand for rapid response, availability, and continuity of services for users, edge computing migration requires additional research. Current migration methods require new methods or modifications. Since they are **fundamentally** designed for cloud applications, these methods **cannot meet** the **latency** and **availability** requirements for mobile IoT devices. This research will focus on **modifying and developing new migration methods** that can use **actual parameter values** rather than only averages. We developed an alternative method to the pre-copy method to overcome its limitations.

Using actual parameter values can lead to new innovative strategies for overcoming additional challenges, such as **improved bandwidth utilization**. This strategy represents an additional perspective that we are examining in the context of our research.

Moreover, all migration models are based on **average parameter values**, although these parameters, such as **memory dirtying rate** and **transfer rate**, constantly **change during the migration process**. These average values can be detrimental for comprehending the migration process and for designing practical components. In addition, the pre-copy method, which is widely applied in the literature, cannot work with parameters other than average parameters due to its reliance on a limited number of rounds. Since parameters such as memory dirtying rate and transfer rate can **fluctuate** multiple times per round, these limited rounds **prevent accurate results** and comprehension.

New methods should be incorporated into existing techniques, such as **ML techniques**, to reduce downtime and migration time further and provide more **service availability** and **continuity** for users.

Chapter 3

MiGrror: Mitigating Downtime in Mobile Edge Computing, An Extension to Live Migration

User devices, called User-Equipment (UE), capable of working with cloud computing, have grown exponentially in recent years, leading to a significant increase in data production. Moreover, upcoming Internet-of-Things (IoT) applications such as virtual and augmented reality and intelligent transportation will require low latency, communications, and processing. Edge computing *is a revolutionary criterion in which dispersed edge nodes supply resources near end devices because of the limited resources available on UEs. Rather than transmitting massive amounts of data to the cloud, edge nodes could filter, analyze, and process the data they receive using local resources. Mobile Edge Computing (MEC), in particular, has the potential to significantly reduce*

This chapter is derived from:

- **Arshin Rezazadeh**, Davoud Abednejad, Hanan Lutfiyya "MiGrror: Mitigating Downtime in Mobile Edge Computing, An Extension to Live Migration, "Procedia Computer Science, Vol. 203, pp. 41-50, 2022 [BEST PAPER AWARD]
- **Arshin Rezazadeh**, Davoud Abednejad, Hanan Lutfiyya "Hybrid-MiGrror: An Extension to the Hybrid Live Migration to Support Mobility in Edge Computing," Journal of Ubiquitous Systems & Pervasive Networks, Vol. 18, No. 1, pp. 39-48, January 2023 [INVITED PAPER]

processing delays and network traffic between UEs and servers when user mobility is considered. Mobility is essential since it affects user applications' response time. This Chapter represents a novel technique for migration that minimizes delay and downtime by utilizing edge computing. Our proposed method syncs more frequently than the pre-copy method, which is the most used migration method that synchronizes (syncs) the source and destination only based on multiple rounds. When compared to established migration methodologies, our results indicate that our mechanism has less latency, downtime, migration time, and packet loss. These results allow delay-sensitive applications that require ultra-low latency to function smoothly during migration.

3.1 Introduction

Cloud computing provides storage and computing power to be used by third-party services. Currently, many applications (apps) use cloud computing [86]. Cloud computing infrastructures are scalable and accessible via flexible pay-as-you-go models. There are limitations to relying solely on cloud resources [87]. One challenge is the long distance between cloud computing resources and data sources/receivers. Consider Internet-of-Things (IoT) applications that monitor the sensors of a physical object. Data must be processed in real-time or near real-time if the application has strict Quality of Service (QoS) and Quality of Experience (QoE) requirements. Accessing cloud resources may require multiple hops. The latency in communicating data from data sources to the cloud may not be timely enough for a delay-sensitive application that requires real-time responses [26], [88], [89].

Fog computing was proposed with the goal to distribute computing, storage, control and networking services at the network edge, where data is generated [57]. This reduces the latency since fewer hops are required to transfer data. However, complications occur if a mobile User Equipment (UE) moves, and hence it may move away from the node that is hosting a service with which it is communicating with. Real-time video streaming and conferencing, face recognition, online gaming, augmented reality (AR), and virtual reality (VR) are examples of applications [29] impacted by mobility. Mobile Edge Computing (MEC) envisions that fog computing resources are provided at the edge of the network [90]. Services can be deployed as a virtual machine (VM) or as a container that is placed on a fog node. Migrating VM/containers that encapsulate a service between edge nodes can be used to deal with UE mobility.

Migration in MEC typically refers to the process of moving a running virtual machine (VM) or a container from the current edge node to either an edge node or cloud without disconnecting applications [59]. Hand-off is a component of migration [29] and is triggered when a device disconnects from an edge node's access point (AP) and connects to the AP of another edge node. VM/container migration typically results in downtime ranging from a few seconds to a few minutes [29]–[36]. This downtime accounts for a significant portion of the delay [60]–[62]. Furthermore, the UE is unable to access services and data during hand-off because it must relocate from the previous connection point to the next. Their downtimes are typically greater than one second, which negatively affects latency, QoS, and QoE requirements, while low latency and real-time apps, such as some AR applications, are in demand. An AR application in a head-mounted device, for example, requires less than 17 millisecond (ms) of end-to-end delay to function smoothly [3], whereas most research delays and downtimes exceed this number and cannot support real-time responses. These findings highlight the need for a new approach to further reduce delay and downtime for modern applications, which could also be integrated into most existing techniques.

Efforts have been made to reduce VM/container migration downtime when a UE hands off from one edge node to another. Data transfers are required when migrating VMs or containers. Recent research has focused on reducing the amount of data transferred during hand-off using different metrics such as runtime and offline characteristics. Their research focuses on predicting the best time to trigger a hand-off and improving the selection of edge nodes to allow for shorter processing times [32], [34], [65], [66]. These approaches mostly use live migration approaches borrowed from cloud computing, which uses the pre-copy migration technique [56], to reduce migration downtime [2], [29], [31]–[35], [65]–[68].

During migration, the pre-copy method transfers data from the source to the destination in predefined rounds[62], whereas our proposed approach synchronizes data between source and destination more frequently as data at the source changes. The main advantage of our proposed technique is that more frequent synchronization results in less data transfer during hand-off.

Most research uses the pre-copy method, while the work described in this Chapter focuses on an alternative to pre-copy that improves the performance of the migration method by reducing downtime when handover is triggered.

The rest of this Chapter is organized as follows: Section 3.2 provides background information on traditional migration techniques and related work. Section 3.3 discusses the design of our approach to edge infrastructure. Section 3.4 evaluates the proposed method, and Section 3.5 summarizes this Chapter.

3.2 Background and Related Work

3.2.1 Background: Fundamental Migration Techniques in Mobile Edge Computing

Stateless migration occurs when the state (which may include CPU, register, signal, and memory states) of the services of the users is not saved; otherwise, it is stateful migration [62], [65]. The primary focus of this paper is on stateful migration techniques. In this subsection, we provide a sketch of the most fundamental approaches to stateful migration. We will use these methods for comparison and evaluation in section IV.

Cold migration: The VM/container execution is halted at the source. The VM/container is then transferred to the destination and resumes as soon as the

VM/container becomes available at the destination. Apps hosted on the UE cannot access their service during this time until the VM/container restarts execution at the new location. Cold migration produces a lengthy downtime compared to live migration methods [62] since live migration methods enable VMs/containers to continue running during most of the migration process [62], [70]. These techniques are described below.

Post-copy migration: This live migration technique first freezes the VM/container to stop run-time state modification and then transfers the latest state to the destination. The VM/container continues operating at the target while the remainder of the latest state is being transferred. The VM/container is connected to the destination while still reading data from the source until the state transfer is completed [62].

Pre-copy migration: With this live migration technique, the entire VM/container state from the source to the destination is transferred. It then transmits modified memory pages, called dirty pages, over several iterations. It later stops the VM/container execution at the source node to copy the last dirty page to the destination. Finally, the VM/container continues execution at the destination [62]. Pre-copy requires more data transfer than post-copy during migration since dirty pages must be periodically sent before hand-off. However, post-copy causes longer delays because it still has some data in the source that must be read from the destination until the state transfer is completed [62].

Hybrid-copy migration: Both pre-copy and post-copy methods, as previously discussed, have drawbacks: (i) Non-deterministic downtime occurs during the pre-copy phase; (ii) service performance during the post-copy stage is affected by faulted pages [62]. The first stage of hybrid-copy migration is identical to pre-copy migration, sending the entire state and then dirty pages to the destination while the virtual machine or container remains operational on the origin [63], [64]. The VM/container is then paused after the hand-off is triggered, and its state is transmitted. The VM/container can be restarted at the target when the state and memory have been delivered. The most recent VM/container's execution state and memory pages are now present in the target.

However, pages may have been dirtied throughout the pre-copy process. Accordingly, the final phase of the hybrid method is to transfer dirty pages to the target using the post-copy method [77]. Since the hybrid technique sends only the dirty pages after the hand-off, it typically transfers fewer memory pages than post-copy. According to the abovementioned research findings, the hybrid technique outperforms both pre- and post-copy migrations.

3.2.2 Migration Strategies in Mobile Edge Computing

Researchers employ various strategies to reduce migration time. In recent years the focus has been on various prediction-based, e.g., [32], [34], [65], [66] and data transfer approaches, e.g., [2], [31], [33], [35], [57], [67], [68] that are used to reduce migration time. The data transfer approach papers focus on reducing the amount of data transferred during hand-off to fundamentally reduce downtime, resulting in significantly reduced latency and migration time and, as a result, user QoE. While prediction-based methods reduce downtime, latency, and migration time when compared to other ML techniques, they do not fundamentally reduce them. In the following, we will provide a brief description of each paper.

Data Transfer: To reduce transfer size during hand-off, Ma et al. [29], [67] proposed an improved migration technique based on the hierarchical structure of the container file system. Transferring the basic image only at the beginning of the migration, followed by iterative memory difference, can help reduce transfer size during hand-off. In the best-case scenario, they had 2.7 seconds of downtime. Machen et al. [35] proposed a layered migration framework that supports container and virtual machine technologies. The framework breaks the application down into various layers and transfers only the missing layers to the destination. This layered method can reduce downtime by transmitting less data from the source to the destination during hand-off. They assessed their performance using various applications, such as video streaming and gaming servers. They significantly lowered overall migration times but with a 2-second average downtime for a blank container. This time is still unacceptable for delay-sensitive applications, such as the stated head-mounted AR application. Farris et al. [68] used the pre-copy technique in stateless migration to achieve low latency by transferring as little data as possible during hand-off. They send data ahead of time before handing off to achieve lower latency. In the best-case scenario, their experiments had more than a second latency. To reduce downtime and migration time, Addad et al. [31] use memory, partial, and full migration strategies with predefined and non-predefined paths, as well as different numbers of pre-copy iterations. They accomplish this by storing the container files in a shared storage pool accessible to all edge nodes and proactively sending data, resulting in less data to transfer during hand-off. They evaluate their performance using video streaming and blank containers, but they have more than one second of downtime in their best-case scenario. Most papers on reducing downtime and delays aim to reduce transfer time during hand-off by reducing data transfer during that period. On the other hand, Zhou et al. [33] propose a hardware accelerator concept to expedite data transmission reduction computations and, as a result, service migration. They achieve about 300 ms of downtime in their best-case scenario. Puliafito et al. [2] stated that an AR application that uses a smart helmet should have a maximum end-to-end latency of 20 ms. This study used pre-copy and proactively sent data using compression before hand-off, resulting in less data transfer during hand-off. Despite their proposal, they still have 3.67 seconds of downtime in the best-case scenario.

Prediction-Based Approaches: Handover can be triggered when the value of a metric, such as the Received Signal Strength Indicator (RSSI) falls below a preset threshold value. The ideal threshold value would provide sufficiently for the migration process to occur with minimal disruption of service. However, there are other factors to be considered, e.g., the load on the wireless links and the speed of the UE. This makes it difficult to determine a fixed threshold value. A poor threshold value results in the handover being done too late or too early. Prediction-based approaches consider multiple metrics as well as a prediction of the UE's movement to determine the migration destination. For example, Ngo et al. [66] used the pre-copy method and proactively sent data prior to hand-off by memory checkpointing before and during the migration phase in order to determine the time to trigger hand-off and the destination edge node. The downtime period and migration start simultaneously in their work, unlike other work where the migration starts earlier than the downtime. The downtown period in this work covers the handover time, which represents the time that the service is unavailable, but it also includes a period of time when the service is up. Although this re-definition influences results, in the best-case scenario, their experiments show about 7 seconds of total downtime and about 300 ms of end-to-end delay, even with this re-definition. This also makes it difficult to compare with most work where downtime corresponds to handover.

Yang et al. [34] developed a multi-tier MEC server deployment framework based on the pre-copy method in order to predict the next node based on the UE's position, direction, speed, and delay requirements. Their experiments demonstrate several seconds of downtime when using various prediction-based techniques with varying UE speeds. Majeed et al. [65] use four regression models to predict offloading time in MEC using various runtime and offline metrics such as CPU and disk utilization, network bandwidth, and container image size to reduce end-to-end latency compared to their other evaluated ML approaches. In the best-case scenario, they achieved 1.4 seconds of delay by proactively transferring data prior to hand-off and employing the pre-copy technique.

Pomalo et al. [32] used K-Nearest Neighbor, Logistic Regression, Random Forest, and XGBoost machine learning (ML) algorithms to predict how much time the migration service should start in advance to the new edge node in order to continue service without interruptions. Their results were evaluated by comparing these ML approaches to determine which method best-handled service continuity. This study requires the path type, e.g., main road, highway, and train, to predict the proper migration start time; otherwise, it produces long downtimes. Furthermore, they stated that their approach still has downtime but did not specify its exact amount. The research mentioned above reduces downtime when compared to other ML algorithms. The downtime is reduced because handover is triggered earlier by taking into consideration factors such as the load on the wireless links and the speed of the UE. Our work is able to reduce downtime regardless of when the handover is triggered. As a result, in most cases, the two method groups mentioned above, prediction-based and data-transfer, can be combined to achieve better performance results.

Zhang and Zheng suggest a deep Q-network (DQN) for task mobility in [91]. They envision MEC servers with a single user traveling between places. The reward function incorporates the cost of migration and the QoS. This method is compared to both a dynamic programming method and a method without migration, and it outperforms both in DQN simulations.

Moon et al. [92] optimize for server load balancing and migration cost while achieving delay requirements in a MEC-based vehicular network using a deep Q-learning (DQL) technique. The DQL strategy performs more tasks and works better at load balancing than the other three approaches. This method takes into account many jobs, yet they are independent of one another.

Wang et al. [93] present an RL technique to microservice task coordination. Their migration strategy is pre-copy migration. They use a Markov Decision Process (MDP) framework to represent the issue, and their RL solution is based on Q-learning. When it comes to delays, they decide that downtime is more important than total migration. Finally, the authors prioritize "microservice coordination" delay above downtime and overall migration time.

3.2.3 Contribution

The pre-copy migration method is used in most of the studies mentioned. Most of the included work described in this section still results in significant downtime, delay, and migration time. The issue with the previously mentioned studies is that they continue to use pre-copy as the primary transfer method [2], [29], [31]–[35], [65]–[68] while reducing transfer size during hand-off [2], [31], [33], [35], [57], [67], [68]. Consider an AR application in a head-mounted device. This app requires a latency of less than 17 ms to operate appropriately [3], whereas the above-mentioned research latencies and even solely downtimes outputs exceed 17 ms. As a result of such a long delay, the user's QoE may suffer, and there is a demand for a new approach to smoother migration that replaces the pre-copy migration technique.

MiGrror is a solution to the problem mentioned above; it is intended for applications that require a low-latency or real-time response. Using MiGrror rather than pre-copy may result in less downtime, delay, and migration time. The pre-copy technique is based on rounds, but memory contents may change more than once during a round, and the source must wait for a certain amount of time before sending dirty pages from the source to the destination. As a result, the source may send dirty pages at a low rate until the hand-off triggers. To improve the efficiency of future IoT and 5G applications, a new design is required that synchronizes more frequent memory differences from the source to destination during migration, reducing the transfer size during hand-off and, thus, downtime and delay. In the remainder of this Chapter, we will present our approach as a complement to current research, i.e., prediction-based methods and other techniques, since it can be used instead of pre-copy, resulting in less downtime and delays than pre-copy.

3.3 MiGrror: Mirroring Service on Fog/Edge Migration

This section presents a new migration algorithm and workflow based on VM/container mirroring that we refer to as MiGrror (a combination of terms migration and mirror). We use mirroring in our approach to synchronize the source and destination VM/containers more quickly during hand-off.

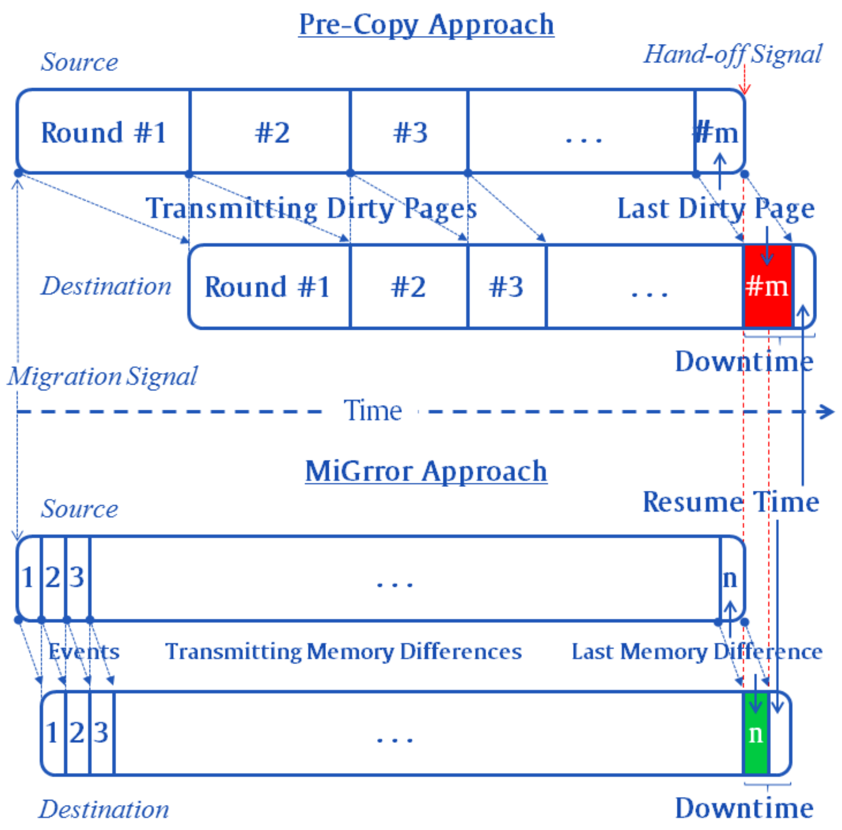

Figure 3.1 Distinction between MiGrror and Pre-Copy.

3.3.1 Description of Pre-copy

This subsection presents the pre-copy method. With pre-copy, a node is selected for migration. A VM/container is initialized on the target node. This is followed by transferring the memory state to the target node even as the VM/container executes on the source node. This transfer is round 1. For round i, where $i > 1$, a page is transmitted if it has been written to (dirtied). The rounds typically end based on a predefined number. At this point, the VM/container on the source node is terminated, and devices using it are now expected to use the VM/container on the target node. It is critically important to minimize the downtime when the VM/container stops running. During this time, the VM/container stops running, and the service/data are inaccessible. Since downtime contributes to performance loss, this should be kept as low as possible, if not zero. We define downtime as follows:

$$t_{downtime} = t_{prepare} + t_{transfer} + t_{resume} \tag{3.1}$$

where $t_{prepare}$ is the time required to stop the VM/container and calculate the latest dirty memory, and $t_{transfer}$ is the time required to transfer the last dirty memory of the VM/container, which is calculated as follows:

$$t_{transfer} = Amount\ of\ the\ final\ dirty\ memory\ data\ (bits)\ /\ bits\ per\ second \tag{3.2}$$

This is defined as the time necessary to disconnect from the current AP (the AP of the source edge node) and connect to the next AP (AP of the destination edge node). We define t_{resume} as the time needed to resume a VM/container at the destination based on the received data after hand-off. Since $t_{transfer}$ has the largest impact on downtime and migration time when compared to other stated parameters [62], we focused on $t_{transfer}$ to reduce downtime.

3.3.2 Description of MiGrror

Fig. 3.1 depicts the distinction between the MiGrror and pre-copy approaches. Assume a UE is moving from one node to another. As illustrated in Fig. 3.1, pre-copy transmits dirty memory at the end of a round, representing a predefined amount of time. With MiGrror, the goal is to reduce the amount of data that must be transferred during downtime in order to achieve higher performance; therefore, we focused on reducing $t_{transfer}$. The distinction between the two methods is that MiGrror, as shown at the bottom of Fig. 3.1, uses events to synchronize (sync) the source and destination as events occur, rather than waiting for the end of a round as pre-copy does. Each memory change at the source causes an event to be generated, indicating that the source and destination must be synced. MiGrror does not need to wait for a period of time to elapse. Instead, MiGrror allows the possibility of multiple synchronizations of the source and destination during the period of time that corresponds to the pre-copy's round in order to mirror the current VM/container available at the destination. These n MiGrror sync events and m rounds of pre-copy are depicted in Fig. 3.1. In most cases, n is expected to be larger than m since MiGrror syncs as soon as a memory change occurs and sends memory differences as soon as they become available. A small amount of dirty memory remains when hand-off is triggered. After the hand-off trigger, this data is the final memory difference that the source sends to the destination for synchronization. This is less memory than the last round in pre-copy since other memory differences have already been transmitted. As a result, $t_{transfer}$ is reduced in MiGrror when compared to pre-copy.

Consequently, as shown in Fig. 3.1, downtime is reduced compared to pre-copy. The diagram's right-most section represents the resumption time required to restart the VM/container at the destination. The remainder of this section will discuss MiGrror.

3.3.3 Synchronized Proposed Algorithms

Algorithm 3.1 describes the synchronized migration pseudo-code for the source edge node, from which migration starts at. Algorithm 3.2 depicts the synchronized migration pseudo-code for the destination edge node, which is where migration completes.

Fig. 3.2 shows the design details of our entire workflow. The source edge node is the node that is currently providing services to end-user applications. After these steps are completed, the destination will provide the migrated service. The specifics of these steps are described in the rest of this section.

S1: Service Running on the Source Edge Node

When a UE is in range, the source edge node, or cloud, transmits the base image to the subsequent potential edge node (see algorithm 3.1, line 1). This action clones the base image to be received by the destination (see algorithm 3.2, lines 1 and 2). Consequently, the source does not need to transfer the base image during migration since it has already been transmitted.

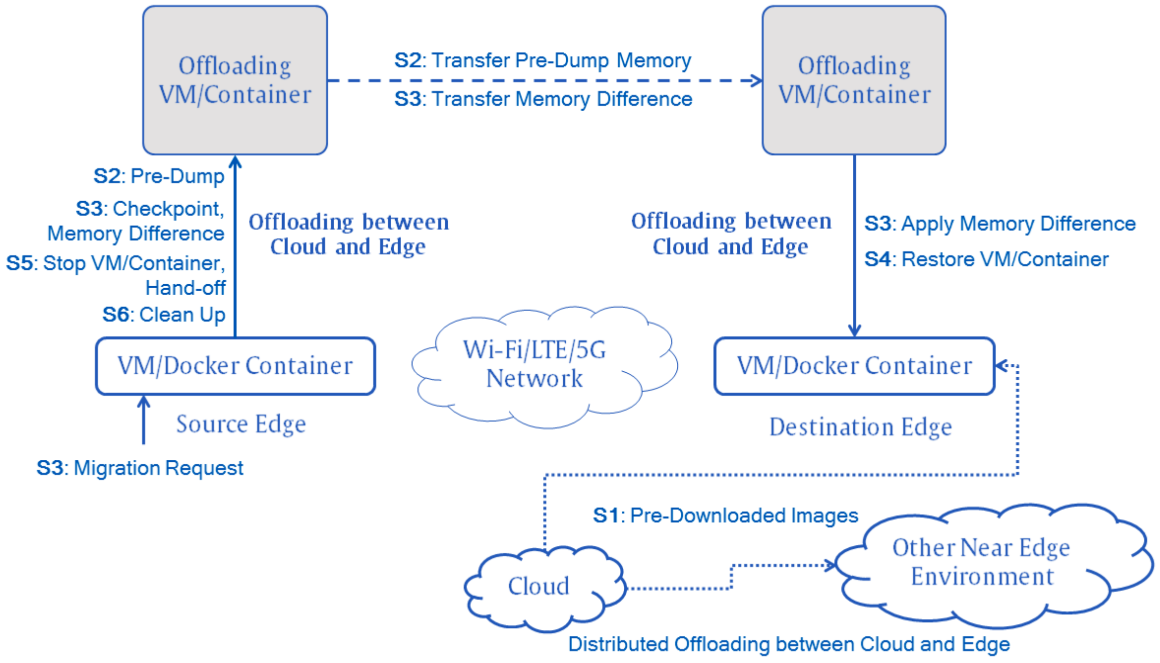
Offloading VM/Container
S2: Transfer Pre-Dump Memory
S3: Transfer Memory Difference
Offloading VM/Container
S2: Pre-Dump
S3: Checkpoint, Memory Difference
S5: Stop VM/Container, Hand-off
S6: Clean Up
Offloading between Cloud and Edge
Offloading between Cloud and Edge
S3: Apply Memory Difference
S4: Restore VM/Container
VM/Docker Container
Wi-Fi/LTE/5G Network
VM/Docker Container
Source Edge
Destination Edge
S3: Migration Request
S1: Pre-Downloaded Images
Other Near Edge Environment
Cloud
Distributed Offloading between Cloud and Edge

Figure 3.2 Workflow of offloading service migration in mobile edge computing

S2: Send the Pre-Dump Memory

Before receiving the migration request, the source edge node synchronizes a memory snapshot (algorithm 3.1, lines 2 and 3) to the target (algorithm 3.2, lines 3 to 5). To reduce the size of the transferred memory image during hand-off, MiGrror checkpoints the source container first, then dumps (stores content of memory) a snapshot of VM/container memory in step S2 without interrupting the VM/container service at the current edge node. This procedure may be repeated as the memory of the VM/container in the current node hosting the service changes. In order to reduce the total time, it is possible to run steps S1 and S2 concurrently.

Algorithm 3.1: MiGrror (at the Source)

```
1: send base image to Destination (or send from the cloud);
2: create pre-dump memory snapshots;
3: send pre-dump to Destination;
4: wait for migrationRequest;
5: handoffSignalReceived = False;
6: while handoffSignalReceived == False
7:     switch (Event) /* memory change OR handoffRequest */
8:     case Memory change
9:         checkpoint();
10:        calculate_memory_difference();
11:        send_sync_event();
12:    case handoffRequest
13:        handoffSignalReceived = True
14:        do hand-off();
15: stop VM/container;
16: wait for T seconds then release VM/container
```

Algorithm 3.2: MiGrror (at the Destination)

```
1: receive base image;
2: start VM/container;
3: receive pre-dump from Source;
4: apply pre-dump;
5: restore VM/container;
6: wait for migrationRequest;
7: handoffSignalReceived = False;
8: while handoffSignalReceived == False
9:     receive memoryDifference;
10:    apply memoryDifference;
11:    restore VM/container;
12:    case handoffRequest
13:        handoffSignalReceived = True
14:        do hand-off();
15: communicate from this edgeNode to UE;
```

S3: Migration Request, Checkpoint, Memory Difference Transmission, and Apply Memory Difference

The source edge node synchronizes a memory snapshot to the target after receiving the migration request that activates the migration signal (algorithm 3.1, line 4 at source, and algorithm 3.2, line 6 at destination). The cloud-edge control mechanism initiates the migration request (step S3 in Fig. 2). Upon activation of the migration signal, MiGrror synchronizes the source and destination. The source node waits for memory changes and signal to start handover events. A memory change event (algorithm 3.1, line 8) triggers a checkpoint (algorithm 3.1, line 9) of the VM/container on the source edge node to obtain a memory snapshot. It then compares two consecutive snapshots (algorithm 3.1, line 10) to determine memory differences in order to determine the dirty memory. The source then sends the memory difference to the destination edge node (algorithm 3.1, line 11 at the source, and algorithm 3.2, lines 9 and 10 at the destination), where it is re-assembled. MiGrror transfers memory differences as memory changes at the source edge node without interrupting the VM/container service (synchronous mirroring). Depending on the policy, it may sync to one or more nodes. A mirror of the source's VM/container resumes running at the destination as a result of mirroring the source and destination. Consequently, we can eliminate much of the usual VM/container resumption time at the destination when the migration process is finished. Furthermore, mirroring could start prior to the migration request, depending on the application requirements.

S4: Restore VM/Container in the Destination Edge Node

The memory difference from the source edge node is received by the destination, which then restores the VM/container with the received dirty memory (algorithm 3.2, line 11).

S5: Stop Source VM/Container and Hand-off, Service Running on the Target Edge

The source sends the most recent state to the destination after receiving the hand-off signal. The destination now has all of the needed data. This data was acquired as a result of cloning in S1, mirroring in S2, S3, and restoring in S4. Control is handed off to the next edge node after the hand-off triggers (algorithm 3.2, line 14), and the VM/container on the source node is stopped (algorithm 3.1, line 15). Simultaneously, the destination edge node will proceed to step S1 in order to prepare for the future possible migration.

S6: Clean Up the Source Edge Node and Failure Recovery

Removing the footprints of the existing container will result in a cleaned source edge node. Algorithm 3.1, line 16 represents the waiting time before the migrated VM/container is released. The benefit of the waiting period is that if the end-user moves back or it may serve as a back-up if the next node’s connection is lost.

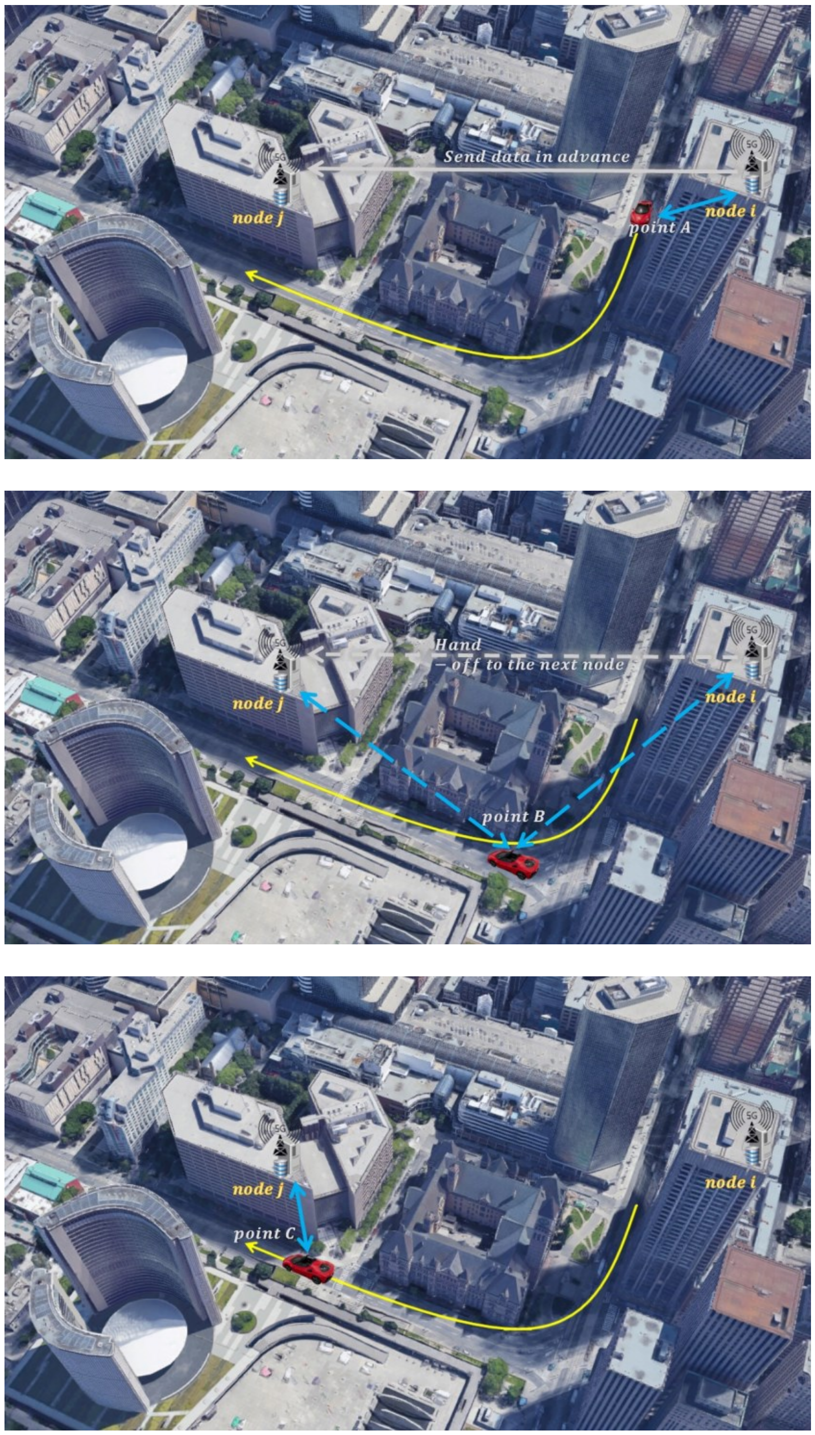

Figure 3.3 A smart city scenario. (a) top; (b) middle; (c) bottom. (Image location: Nathan Phillips Square, Toronto, ON Canada, from maps.google.ca. Red car from ferrari.com, both accessed Dec. 2020).

3.3.4 A Smart City Scenario

Consider a road on which a mobile UE is travelling. The mobile UE could run various applications, such as an AR app in a head-mounted device. As illustrated in Fig. 3.3, a UE will leave the node i range and enter the node j range. Node i is the source in this diagram, and node j is the destination.

Fig. 3.3 depicts a mobility model, edge nodes, migration, and hand-off. As shown in Fig. 3.3, the end-user moves from point A to point C while maintaining the connection for apps and services. Fig. 3.3a depicts the user at point A, where the migration starts. When the end-user is within the node i range, mirroring starts. As a result, a live copy of the source node's data resides at the destination is always ready to use.

As the end-user device advances, it reaches point B in Fig. 3.3b. The hand-off that occurs on point B helps keep apps alive and connected throughout the migration. It connects the current edge node (node i) to the new one (node j). At this point, apps data should have completed transferring from node i to j, allowing apps to continue providing services when the end-user reaches point C, as shown in Fig. 3.3c.

When the user arrives at point C, the previous node has completed the VM/container migration (source node i). As a result, apps receive service from the new edge node (destination node j). This fashion occurs every time a user moves from one location to another. Since data is quickly mirrored to destinations, the hand-off at point B will be as seamless as possible. Our proposed method also assists apps in handling their services while the end-user moves from one edge node range to another. This performance comes at a cost, which could be a loss of network bandwidth during mirroring.

3.4 Results and Discussions

This section describes the simulation setup and reveals insights from the simulation results of the proposed approach.

3.4.1 Simulation Setup

We use MobFogSim [27] for our simulations and extend it to provide results in this section. MobFogSim is an extension of iFogSim [94] that supports device mobility and VM/container migration. iFogSim is an extension of CloudSim [95]. Moreover, we use real-time mobility patterns of UEs. These patterns are embedded in MobFogSim using the Simulation for Urban Mobility (SUMO) [96] tool and Luxembourg SUMO Traffic (LuST) [97]. The latter is a dataset that studied vehicles in Luxembourg over a 24-hour period, covering an area of 156 km^2 and 932 km of road captured in 2015. Our simulation used this dataset to determine the vehicle's position, speed, time, and movement direction.

In this study, to analyze our proposed method's performance, we performed an experimental evaluation using a Virtual Machine running on an Ubuntu 18.04.5 LTS with a 2.2 GHz quad-core processor, 64 GBs of storage, and 32 GBs of memory. We assumed a mobile edge computing infrastructure with 2 $Mbps$ bandwidth for each UE and 100 $Mbps$ bandwidth for each AP. Other features, such as a processing limit of 281 $MIPS$, can also be used to describe UE configuration. Processing power estimated by Million Instructions Per Second ($MIPS$).

We used a variety of input values as part of the simulation environment. We considered a square 10 x 10 km area with 144 uniformly distributed edge nodes as the environment. Each edge node is linked to a single AP with a 1000-meter signal range. A simulation ends once the user finishes the migration processes. The migration process starts once the user reaches the migration point, defined at 40 meters from its connected

AP (source node's AP) coverage boundary. Our migration strategy selects the one with the lowest distance (Euclidean distance) between the user and the access point of the candidate edge nodes present in the user's path. The stated dataset determines the user speed, and the network bandwidth between edge nodes is set to 100 $Mbps$ with a latency of 1 ms.

3.4.2 Performance Characteristics

We evaluate methods using the following five metrics:

Downtime is the duration of transferring a stopped VM or container from one edge node to another and restarting it. The VM/container ceases to function during this period, and the associated services/data become unavailable.

Network usage is the amount of data transferred between the source and destination nodes when migration has been triggered.

Data loss refers to the amount of data that a UE needs to transmit to/receive from an edge node during downtime but will not be able to do so because there is no connectivity between the UE and the edge node. This metric represents a disruption of service.

Average delay is the latency between a mobile UE that runs an application and the edge node that runs the corresponding VM/container. Delay does not include packet loss. The shorter the value, the less time that users wait to receive data from the migrated service hosted on the destination edge node and hence a higher performance for their running apps.

Average migration time is referred to as the time needed to prepare, perform various transfers, and restart the VM/container at the destination edge node. It is then completed when the VM/container restarts for the final time with all required data at the destination edge node.

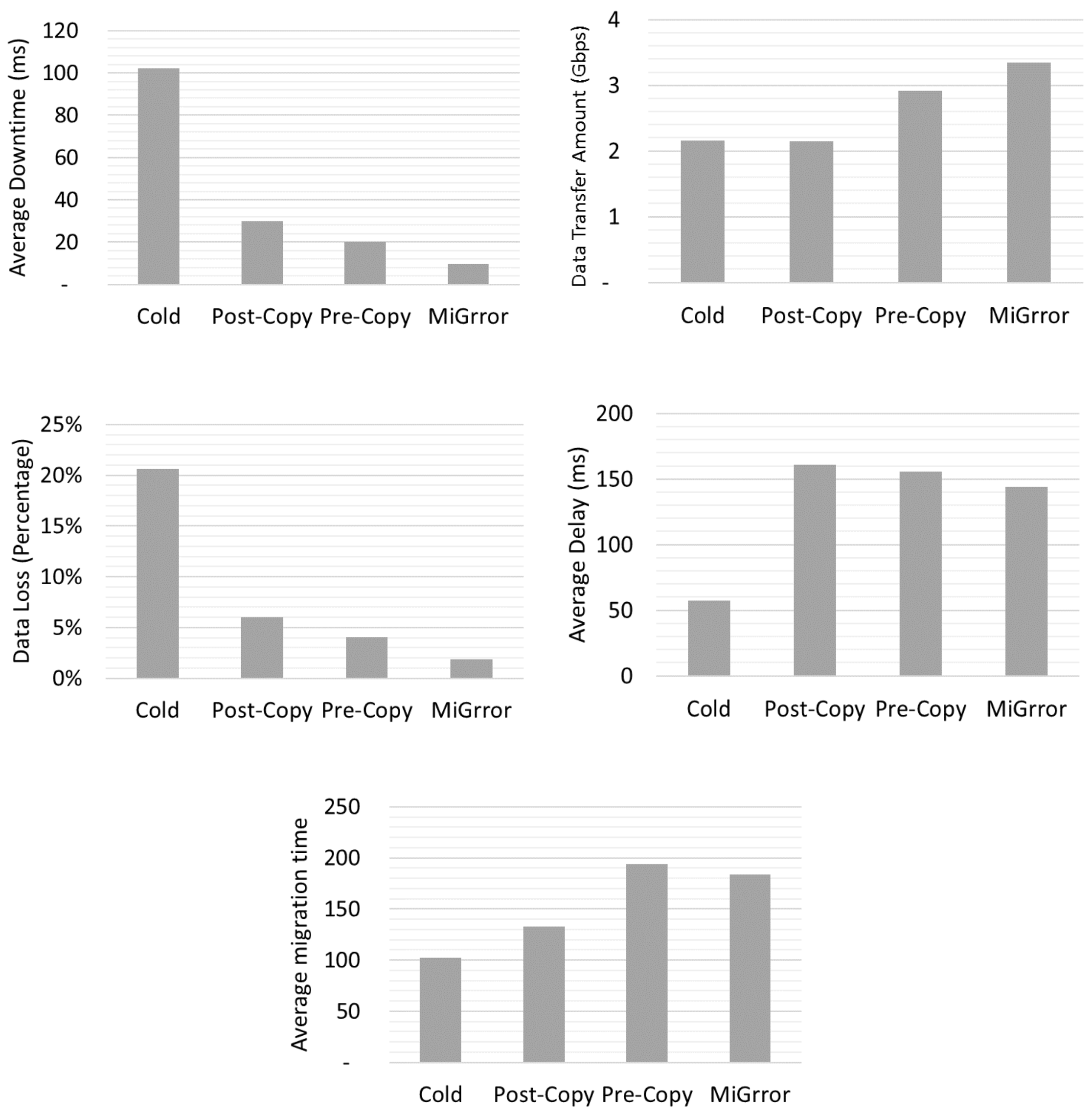

Figure 3.4 Results: top left: (a) Average downtime; top right: (b) Total network usage; middle left: (c) Data loss; middle right: (d) Average delay; bottom: (e) Average migration time.

3.4.3 Simulation Results

The simulation results were then compared in terms of (i) downtime, (ii) network usage, (iii) packet loss, (iv) average delay, and (v) average migration time. We ran 30 simulations with the same configurations. It is worth noting that we used post-copy via pseudo-paging in our experiments.

Fig. 3.4a demonstrates that the cold migration downtime is the longest, lasting more than 100ms. The downtime for post-copy migration is less than that for cold migration [62], which is to be expected. Pre-copy has less downtime than post-copy, slightly over 10ms; nonetheless, this downtime may be unacceptable for applications with stringent real-time requirements [98]. MiGrror has the shortest downtime of any of the four evaluated approaches, at roughly 9.5 ms. This downtime should be acceptable for almost all applications, especially for some 5G apps that need a very short delay, mostly less than a 10 ms end-to-end latency [98].

Fig. 3.4b illustrates the amount of data transferred for migration. Cold and post-copy migration methods transfer the least amount of data since they only send memory data to the destination once. However, MiGrror and pre-copy migration methods send memory pages many times, as shown in the graph. MiGrror, in particular, has the most transferred data because it sends more memory differences than the other migration approaches. An approach that uses less bandwidth is more acceptable. This volume of data transfer may be the MiGrror approach's disadvantage when used. However, the bandwidth of 5G networks is more than ten times that of 4G networks, ranging from 1 to 10 $Gbps$ [99]. To provide reasonable downtime for future highly delay-sensitive applications requiring a low-latency response, edge nodes using our migration approach could communicate with each other more than traditional methods and thus consume more bandwidth. It is, yet, dependent on the application requirements and use-cases. An app may require ultra-low latency or may have bandwidth constraints. Different characteristics affect the use of the mentioned migration methods and should be selected deliberately.

As demonstrated in Fig. 3.4c, cold migration has the highest data loss during downtime. Since it produces longer downtime than other methods, it affects its data loss. At the same time, data loss for the post-copy and pre-copy migration is around 6% and 4%, respectively. MiGrror has the lowest amount, approximately 2 percent data loss during downtime. This value is expected because MiGrror has the lowest downtime among the assessed methods, revealing another strength of the proposed migration approach.

The average delay is presented in Fig. 3.4d. The average delay does not consider dropped packets and only considers packets that arrive at their destination. Cold migration provides the shortest latency; however, we should consider data loss because the simulation does not account for it in measurements. Nonetheless, when compared to other approaches, MiGrror has the shortest average latency, which is 10% less than the post-copy method and 9% less than the pre-copy method. These results demonstrate there is less delay in being able to use the migrated service. This supports the smooth operation of latency-sensitive applications in the edge environment.

The average migration time for each approach is depicted in Fig. 3.4e. Cold and post-copy migration approaches require shorter migration time than other methods since data is not sent several times before hand-off. The essential contrast is between the pre-copy and MiGrror because they both send data prior to hand-off. As shown in Fig. 3.4e, MiGrror's migration time is less than that of the pre-copy.

3.5 Summary

This Chapter discussed the importance of VM/container migration in mobile edge computing. Migration can take into account the edge node's geographical location in relation to the user, user direction and speed, and network characteristics of the user and the edge node. Four mechanisms for the VM/container migration have been sketched and then simulated. We consider the user's mobility and wireless connectivity in the simulation. Our results reveal that MiGrror provides promising results when used to support MEC applications with mobile users, particularly when the latency is critical. This performance may come at the expense of increased bandwidth usage for some applications.

While MiGrror is a promising migration method in edge computing, understanding its performance and the future behavior of a system using MiGrror is vital. Measuring the performance is essential when comparing MiGrror to other migration methods. In order to fully understand the performance of the MiGrror, its model must be developed. The model must incorporate crucial factors, including downtime and migration time. Furthermore, unlike other research that relied solely on average values, the MiGrror migration model requires non-average values due to the continuous transfer of dirty memories from the source to the destination. The subsequent Chapter will discuss the MiGrror migration method's models.

Chapter 4

A Non-average Multi-microservice Migration Modelling Approach

This Chapter analyzes the performance of the MiGrror migration method and the pre-copy live migration method when the migration of heterogeneous multiple VMs/containers is considered. Researchers use mathematical modeling to comprehend the behavior of a future system. This Chapter presents mathematical models for the stated methods and provides migration guidelines and comparisons for services to be implemented as multiple containers, as in microservice-based environments. Experiments demonstrate that MiGrror outperforms the pre-copy technique and, unlike conventional live migrations, can maintain less than 10 milliseconds of downtime and reduce migration time with a minimal bandwidth overhead. The results show that MiGrror can improve service continuity and availability for users. Most significant is that the model can use average and non-average values for different parameters during migration to achieve improved and more accurate results, while other research typically only uses

---

This chapter is derived from:

- **Arshin Rezazadeh**, Hanan Lutfiyya "Multi-microservice Migration Modelling, Comparison, and Potential in 5G/6G Mobile Edge Computing: A Non-average Parameter Values Approach," IEEE Access –accepted *average values. This Chapter shows that using only average parameter values in migration can lead to inaccurate results.*

4.1 Introduction

New cloud-based applications (apps) have emerged due to the cloud's virtually infinite accessible resources and extensive service offerings [86], [100], [101]. Moreover, **microservices** are gaining increasing interest as a potent architectural practice for delivering software services. In this approach, applications are designed as a set of modules known as microservices, with each module focused on one component of the entire application [102]. Currently, microservices are delivered utilizing container frameworks rather than virtual machines (VMs). Although the microservices concept was originally built for the cloud context, it is gaining traction as a viable solution for **edge computing** environments [103]. However, these advancements have been followed by challenges for delay-sensitive applications with strict delay requirements [26]. **Mobility support** and **low latency** cannot be accommodated by the present cloud computing paradigm [87]. In order to solve these issues, the fog [57] and edge computing [4] paradigms have been proposed that seek to expand cloud resources and services and bring them closer to the network's edge where data is generated. Consequently, end-to-end latency is lower since the data is transmitted across fewer hops.

Multi-access/Mobile edge computing (MEC) was recently introduced as a key enabler of future 5G and 6G networks, shifting services from large remote cloud servers to an ubiquitous architecture of micro servers close to access networks and base stations [104]–[106]. This proximity can help MEC provide its main characteristics: **mobility support**, **real-time response**, and **high bandwidth** [107], which is especially important for mobile Internet-of-Things (IoT) devices. These characteristics are vital for demanding applications such as autonomous vehicles, healthcare, virtual reality, augmented reality, and online gaming [4], [29], [108], [109], particularly when migration is involved. With more users shifting to edge computing and microservices, managing resources is becoming more challenging. Mobile devices at the network's edge may be repositioned between various MEC nodes. When this movement occurs, corresponding microservices may **require migration** between MEC nodes to keep proximity to the device [6]. Furthermore, some MEC nodes may become overloaded due to changing workloads, while others may stay underutilized on the same network infrastructure [14].

In some modern applications, multiple cooperating services are required in microservices-based environments to provide certain services; thus, we may need to consider migrating multiple microservices [21] for those containers that need to be in proximity to the device. Each containerized application may make use of multiple containers. Furthermore, each mobile IoT device may run multiple applications; in this environment, **multiple** container migration is inevitable [5], [11]. Therefore, we need to investigate the simultaneous migration of multiple containers. Assume a smart city in which tourists are traversing with their mobile IoT devices. The mobile IoT devices are running applications such as augmented reality (AR) and virtual reality (VR) for a virtual tour guide in the context of the metaverse. The mobile IoT devices are connected to the edge to reduce application latency (turn-around time) and to provide more bandwidth. As a basic example of a containerized metaverse application, one microservice captures the environment from the device, and another microservice renders the AR data to the device. Each microservice can use single or multiple containers in its tasks. For the VR component of the application, another service deploys virtual reality components to the mobile device. This application needs ultra-low response time for smooth functionality. For such real-time applications, Salman et al. [3] suggests an end-to-end response threshold of 17 milliseconds (ms); otherwise, it cannot meet real-time latency requirements. While tourists traverse the city, mobile IoT devices require the migration of some containers in order to keep their connections alive. The remaining containers, e.g., containers that render the environment, can replicate without migration to reduce response time. Since the applications require high bandwidth and ultra-low latency, migrations and hand-offs must occur fast enough to keep the applications' response time as minimal as possible.

The hand-off is a migration component [29] that is triggered when a device disconnects from the access point (AP) of an edge node and connects to another node's AP on the same network infrastructure. **Downtime** occurs when a VM or container is unavailable during migration while a device is handed off from one edge node to the next [28]. Downtime caused by VM/container migrations lies in the range of seconds to minutes [29]–[36]. The **delay** is strongly affected by the amount of downtime and page faults [60], [61]. Moreover, since the mobile IoT device must migrate from the old connection point to the new one throughout the procedure, it cannot access services or data during hand-off. There has been considerable work focused on reducing downtime [29], [31]–[35], [39]. Live migration techniques could facilitate downtime issues by sending and receiving data while the VM or container is still operating at the source or destination.

The pre-copy live migration proposed by Clark et al. [56], mostly used in literature, moves data from the source to the destination in pre-determined rounds that regularly transfer changes from the source to the destination. Despite pre-copy lowering the downtime compared to the non-live migration method, the VM/containers are not synchronized (sync) immediately following a change in the memory from the source [28]. This late synchronization causes more data to be required to transfer after hand-off and, consequently, high downtime and migration time for delay-sensitive applications [28]. Some or all application components of intelligent transportation, virtual reality, healthcare, augmented reality, and online gaming requires ultra-low latency for data processing and communication [9].

The stated end-to-end response is difficult to achieve with the pre-copy method which led Rezazadeh et al. [28] to propose the **MiGrror migration technique** for faster synchronization between source and destination, which results in less data transmitted during hand-off and, consequently, less migration time and downtime compared to the pre-copy method. The MiGrror technique mirrors memory from the source to the destination in the same way that mirroring is used in wide-area network servers. In this analysis, the pre-copy method ends and hands off when the number of rounds reaches a pre-defined threshold, e.g., 10-30 rounds in most research. To ensure fairness, we initiate the hand-off for both methods at the same time: pre-copy and MiGrror.

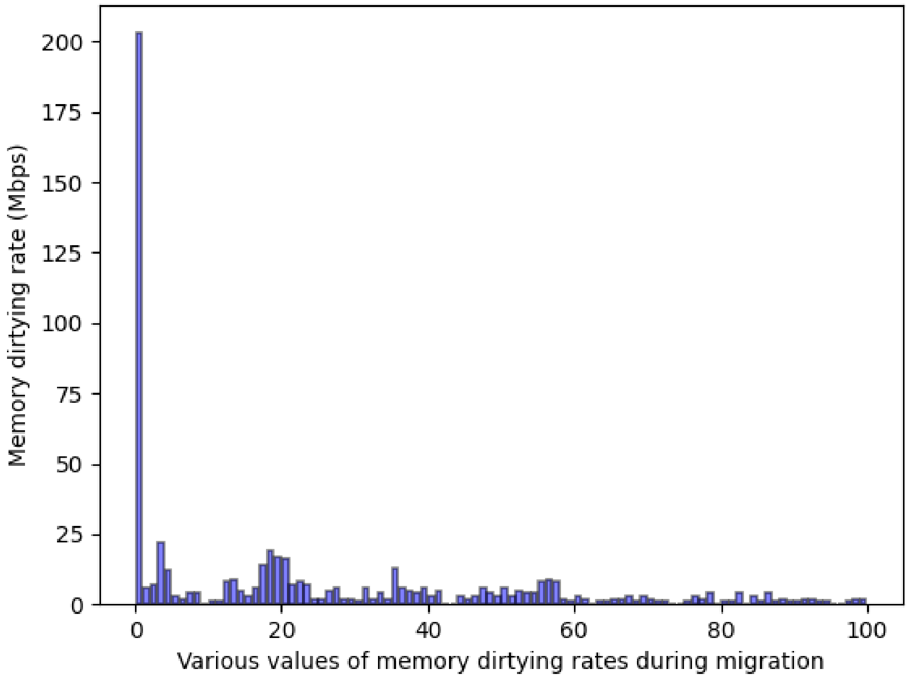

Figure 4.1 Various values of memory dirtying rates during migration.

Furthermore, another limitation of the previous work on migration is that the evaluation typically assumes average input parameter values during migrations. The input parameters considered in this research are the **transfer rate** (provisioned bandwidth), **memory dirtying rate**, and memory **size** of the VM/container (VM/container size). However, the given input parameter values **fluctuate** over time during migration. Memory dirtying rate and container size values vary during the migration process depending on the task for each VM/container of an application. Moreover, the transfer rate can vary throughout the migration since the user's **mobility** causes changes in the distance between the user's device and its services, resulting in diverse signal strength and available bandwidth for each VM/container of an application [85].

Studies [13]–[24], [37]–[42] use **migration modelling** to comprehend the future behavior of a system. Although most research employs average parameter values [13]–[24], [37]–[42] and **assumes** the **input parameter values remain unchanging**, our study demonstrates that the results vary since the input parameters can constantly change during the migration. Our results show it is essential to learn if these parameters are higher or lower at the beginning, middle, and end of the migration, considering migration time and downtime as output parameters. Downtime and migration time are the two primary output parameters in migration [14].

The models provided need to be more accurate to represent real-world scenarios. Figure 4.1 depicts the various memory dirtying rates during a migration process derived from the CSAP dataset [84]. The CSAP dataset consists of a comprehensive collection of over 40,000 instances of live migration samples accumulated over a span of several months. The dataset comprises multiple parameters, namely the memory dirtying rate and transfer rate.

The figure depicts hundred values representing the data that must be transferred from the source to the destination during migration. The values presented in figure 4.1 were obtained from a snapshot of the CSAP dataset. The values presented in figure 4.1 were obtained from a subset of the CSAP dataset and subsequently averaged. In our experiments, we observed the **identical trend** displayed in figure 4.1 and described in reference [29].

The problem is that while the stated input parameters can change continuously throughout the migration process, most migration models [13]–[24], [37]–[42] for multiple services **assume** that the **parameter values remain constant** in contrast to what we see in figure 4.1.

As a result, using only average parameter values can result in output parameters that differ from their realistic outputs because output parameters can be similar, when using the same average values for input parameters. In contrast, the outputs for non-average input parameters **can deviate**, while maintaining the same average input values. Consequently, utilizing non-average parameter values can produce different results while their averages remain unchanged. Using non-average input parameter values can result in more precise migration time and downtime outcomes. In addition, innovative strategies for migrating multiple VMs/containers are possible when considering non-average parameter values.

To exemplify the discussed **current migration models' limitations**, consider two migration procedures with the same average parameter values (e.g., transfer rate and memory dirtying rate) but varying values during the migration process. In this example, increasing the memory dirtying rate at the end of the migration significantly impacts downtime since downtime occurs when the migration is complete [6], and this increased memory dirtying rate requires transferring a higher-than-average memory dirtying rate. The same holds true when we decrease the transfer rate at the end of the migration and the dirtied memory data transfers at a lower-than-average transfer rate. In both cases, using average parameter values generates the same migration time and downtime, whereas using actual (non-average) parameter values generates different outputs.

The same concern as described above occurs due to employing average parameter values in current migration models while decreasing the actual (non-average) transfer rate at the beginning of the migration in this example. As a result, if input parameters are higher or lower than their average value at crucial points - at the beginning and the end of the migration process, they can negatively affect downtime and migration time. These situations worsen when parameters at the given critical migration points are significantly higher or lower than average, and in these cases, they extremely affect downtime and migration time.

However, **these crucial states are hidden** when developing the edge computing environment, relying exclusively on average input parameter values. As a result, **new migration models in edge computing are required** to achieve more accurate results for multi-container mission-critical 5G/6G mobile IoT applications. This paper presents mathematical models for the pre-copy and MiGrror migration methods, offering a new perspective on modelling by utilizing both average and non-average values during MiGrror migration, while typically, only average values are considered in the literature [13]–[24], [37]–[42]. The research in this paper takes into account non-average values of transfer rate, memory dirtying rate, and VM/container size for the MiGrror technique in addition to average values. Our experiments show that the results of both average and non-average parameter values for the pre-copy method are mostly identical since memory changes several times in each round. However, the MiGrror method can consider a larger number of synchronizing events, which is advantageous since using actual (non-average) parameter values rather than average ones is possible when employing the MiGrror method. We take this novel approach since some parameters, such as memory dirtying rate, may change several times during the migration. To the best of our knowledge, this is the first time that different values of bandwidth, memory dirtying rate, and VM/container size, rather than classical average values, are considered during the migration of each single VM or container. To distinguish between these two types of modelling, we also consider average parameter value results and compare them to non-average parameter value results in section 4.5. Furthermore, the non-average MiGrror migration model is applicable regardless of whether machine learning approaches, compression, or other methods are used to decrease migration time and downtime.

In this Chapter, we first model the migration of multiple containers for both stated migration methods. We do this by first using average values of the CSAP dataset [84], then non-average values, followed by non-dataset input values. We also compare the migration overhead of both methods listed and discuss which is better suited to specific scenarios.

The main contributions of this Chapter are summarized as follows:

- We present the MiGrror mathematical migration **model** for heterogeneous **multiple** VMs/containers. This is the first MiGrror model that considers the simultaneous migration of multiple VMs/containers.

- For the first time, we use **non-average** and classical average values for the transfer rate, memory dirtying rate, and VM/container size, during each migration period of every single VM/container for the MiGrror method.
- We conducted experiments to analyze the input parameters that impact the performance of the investigated migration methods.

The remainder of this Chapter is structured as follows: Section 4.2 delivers background and related work on classic migration strategies. Section 4.3 describes and compares the classic pre-copy live migration model used in this analysis with the new model. Section 4.4 presents models of the MiGrror migration for multiple VMs/containers. Section 4.5 provides evaluations and discussions, and Section 4.6 summarizes this Chapter.

4.2 Background and Related Work

This section provides a high-level overview of edge computing live migration techniques, as well as models for migrating multiple VMs and containers. Live migration allows virtual machines and containers to remain operational for most of the migration process [39]. First the pre-copy and post-copy live migration methods are summarized, followed by the MiGrror technique, and finally, the migration model studies are reviewed.

Pre-copy Live Migration Technique: With **pre-copy migration** [56], the entire VM/container state is sent from the current node to the target node. An **iteration** is a round in which the pre-copy waits for memory changes to send at the end of each round. The source then resends **dirty pages**, which are updated memory pages from the previous iteration, over a number of iterations. Upon receiving the hand-off signal, the source VM/container pauses execution to prevent memory and state modification and transfers the final dirty page and the latest changes in the runtime (execution) state, which includes CPU and register updates, to the target edge node. Finally, the VM/container resumes operation on the target edge node. Since the pre-copy technique typically transmits each memory page multiple times, it may have a negative impact on the **total amount of data transmitted** throughout the migration process and, consequently, the **total migration time** [28]. Figure 4.2 shows the pre-copy iterations and related symbols.

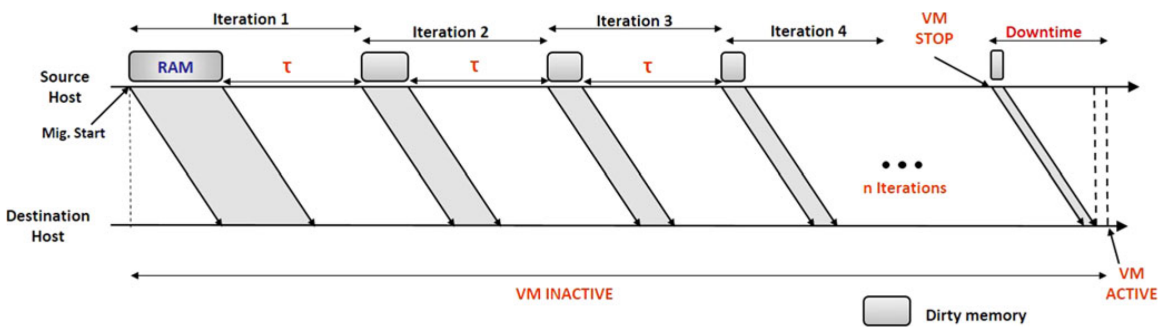

Figure 4.2 Pre-copy iterations (rounds) [42].

MiGrror Migration Technique: The **MiGrror** migration method [28] was introduced to reduce migration time and downtime when compared to the pre-copy method. This objective is accomplished by synchronizing the source and destination more frequently, resulting in a **mirror** of the VM/container at the destination, similar to how mirroring is done in wide-area network servers [28], [110]. This technique reduces the amount of data transferred during hand-off. Despite the intention to use more bandwidth, the results indicate that this method outperforms pre-copy in terms of downtime, delay, and total migration time. Furthermore, the amount of data transferred during migration is greater than that of live migration techniques. Since 5G and 6G networks have significantly more available bandwidth than previous generations, increasing bandwidth usage between MEC nodes in this approach should not substantially impact overall performance [28], [99]. This method will be examined by presenting a mathematical model in section 4.4, followed by results and discussion in section 4.5.

Post-copy Live Migration Technique: Before transferring the latest state from the source to the target, the **post-copy migration** technique [72] pauses the VM/container execution to prevent runtime state changes. The state is then transferred to the target, along with the minimum memory and state required to resume the execution of the VM/container. The VM/container is then resumed at the target. An access problem occurs when the VM attempts to access a page that the target has not yet received. In this condition, a page fault occurs, and the source transmits the faulty page to the target. When the VM/container is restarted at the target node during the post-copy process, any applications executing in the VM/container continue to run at the target. After sending all remaining pages, the page transfers to the target stops, and the VM/container post-copy migration is complete.

Modelling Live Migration Techniques: Several research studies on live migration modelling have been conducted over the last decade. Most of them base their research on the use of a **single** VM or container migration [16], [37]–[42]. The studies primarily focus on downtime and migration time and compare live migration techniques based on various input values, such as pre-copy iterations, page dirtying rate, bandwidth, and VM/container size, using datasets, implementations, or their assumptions. A subset of these papers provides models and compares various parameters of live migration methods [16], [37], [38], [40], while others employ estimation and optimization techniques to reduce migration costs, such as downtime and migration time [39], [41], [42]. Despite extensive research on modelling the migration of a single VM/container, few authors focus on modelling multiple-VM/multiple-container migration [13]–[15], [17]–[24]. Some of these studies focus on the number of VMs/containers and provisioned bandwidth in addition to the stated input values. Most of these studies focus on modelling **multiple-VM** migration [14], [15], [17], [19]–[24] to optimize the migration performance of multiple VMs, while authors in [13], [18] focus on modelling **multiple-container** migration. The authors in [13]–[24], [37]–[42] employ only average parameter values.

Single Service Migration: Altahat et al. [39] propose a neural network-based model that predicts VM migration performance metrics for pre-copy and post-copy methods as well as different application workloads to analyze the migration models under various workloads. Metrics include downtime, migration time, and the amount of data transferred during the migration process. They compare their model to Linear Regression, SVR, and SVR with bagging. The authors of [40] propose an adaptive VM monitoring strategy for migrating a single VM using pre-copy and post-copy methods. They develop an autoregressive model to predict the dirty memory rate and use it to reduce migration downtime, migration time, and the data transfer amount. The model's output value is determined by a linear combination of a stochastic variable and the previous model's values. Tang et al. [16] use reinforcement learning with deep Q-learning container migration to propose power consumption, delay, and migration cost models. The evaluation compares their algorithm to other ML algorithms, including static threshold, median absolute deviation, and interquartile range regression. Baccarelli et al. [41] use the pre-copy migration time, downtime, round-trip time, and energy consumption models to reduce delay and energy consumption in wireless connections with a bandwidth manager.

Multiple Service Migrations: The authors develop adaptive bandwidth allocation in [14], [18], [22] to minimize migration time in their models. Singh et al. [14] use Geometric Programming to assign transfer and compression rates to each VM in order to reduce the total migration time of multi VMs. The parameters considered include VM size, memory dirtying rate, transfer rate, compression ratio, and the number of pre-copy iterations of VMs. They evaluate their experiments with up to seven VMs and nine pre-copy iterations. Maheshwari et al. [18] developed a cost model for multi-container migration, considering container size, number of containers, memory dirtying rate, bandwidth, and load at an edge node that supports mobility. They use a Min-Max model to minimize the migration cost. Liu et al. [22] build a migration cost model by predicting the memory dirtying rate and employing parameters such as VM size and transfer rate. They use a cost model for multi-VM and employ adaptive bandwidth allocation to reduce migration costs.

Migration Models that Prioritize Downtime and Migration time: The following papers consider migration to be the primary or secondary contribution of their analysis. In more detail, Sun et al. [21] use an M/M/C/C queuing migration model to optimize migration and reduce downtime and migration time for multiple VMs. Satpathy et al. [15] compare migration model performance for multiple VMs, including comparisons based on VM size, memory dirtying rate, and available bandwidth. Using a platform based on a software-defined network (SDN), He et al. [17] evaluate the performance of multiple VM migration models. They consider migration time and downtime to be two of their most important criteria. To balance server load, Zhang et al. [13] propose a set of algorithms for optimizing load balancing and migrating multiple containers among cloud servers in order to balance server load. The primary focus is load balancing; migration would occur as a result of server load balancing with the migration time model. Similarly, Forsman et al. [23] present a load-balancing solution that reduces the migration cost of multiple VMs. They also include migration time and downtime in their cost model. In another study, Satpathy et al. [19] propose a VM placement strategy for cloud servers while modelling multiple-VM migration with downtime and migration time. Considering power constraints, Elsaid et al. [20] examine the migration cost of multiple VMs using migration time and power consumption. Cerroni [24] investigates the cost of migrating multiple VMs based on downtime and migration time using the Markovian model. The network overhead and throughput degradation are also components of the migration cost model.

4.3 A Pre-copy Migration Mathematical Model for Multiple VMs/Containers Using Average Parameter Values - the Pre-copy Migration Model

This section describes existing pre-copy migration models for multiple VMs/containers that use average parameter values [14]–[18], [21], [22], [37]–[42] that will be used in our experiments. Although the stated pre-copy migration models are not completely identical, the pre-copy migration for multiple VMs/containers in this section will be modelled derived from the models used in [14]–[18], [21], [22], [37]–[42], so that we can compare the pre-copy results with the proposed migration model. We use only average parameter values for the pre-copy method since considering non-average parameter values is ineffective. This method typically employs a limited number of rounds, e.g., 10-30 rounds in most research [14]; however, this is incompatible with constantly changing parameter values, such as memory dirtying and transfer rates. The transfer rate and memory can change multiple times in each round of the pre-copy migration in short intervals. Therefore, there is no single value for the stated parameters for each round of the pre-copy.

Furthermore, since the pre-copy migration method uses a limited number of rounds, results for both average and non-average parameter values are mostly identical. Utilizing non-average parameter values based on the preceding discussions would be inefficient, as the pre-copy method cannot employ every single value of each input parameter during the migration. Downtime and migration time are the two primary parameters for migration modelling analysis [14]. Downtime is an important performance metric for end-users, which must be as low as possible to avoid service interruptions [14]. The total migration time must be as short as possible because it consumes computational and network resources from both the origin and the target MEC nodes [14]. The amount of data that must be transmitted during the migration of multiple VMs/containers is also considered as an overhead metric of the migration process in this paper. Table 1 defines a number of key parameters and their notations for the models described in this paper. In the table, M_j, $\bar{d}_j$, and $\bar{r}_j$ represent the VM/container memory size, average memory dirtying rate, and average transfer rate (average bandwidth) available during migration for any $VM_j/container_j$, respectively. The parameters specified affect migration time $\left(TM_j^{Pre}\right)$ and downtime $\left(TD_j^{Pre}\right)$. Higher M_j and $\bar{d}_j$ levels increase migration time and downtime, while higher $\bar{r}_j$ levels decrease migration time and downtime. The remainder of this section will examine the **downtime**, **migration time**, and **migration overhead** of the pre-copy migration model.

During round one, the entire memory of any VM_j or $container_j$ is transferred from the source to the destination. As a result, the data transmitted during round one, i.e., $V_{1,j}^{Pre}$, may be calculated using the equation below:

$$V_{1,j}^{Pre} = M_j \tag{4.1}$$

The memory becomes dirty throughout the transfer as the $VM/container$ remains active at the source during pre-copy migration. Then, the pre-copy rounds transfer just the memory that was dirtied during the preceding round. The amount of data sent at round i for every $VM_j/container_j$ is:

$$V_{i,j}^{Pre} = \bar{d}_j t_{i-1,j}^{Pre} \tag{4.2}$$

Table 4.1: Symbols and Definitions

Parameter	Description
m	The number of migration iterations in the pre-copy method
n	The number of migration events in the MiGrror method
p	The number of VMs/Containers to be migrated
$V_{i,j}^{Pre}$	Dirty memory generated during round i for any $VM_j/Container_j$ in pre-copy, $\forall\, j \in \{1, \dots, p\}$ and $\forall\, i \in \{1, \dots, m\}$
$V_{i,j}^{Mir}$	Dirty memory generated during event i for any $VM_j/Container_j$ in MiGrror, $\forall\, j \in \{1, \dots, p\}$ and $\forall\, i \in \{1, \dots, n\}$
$V_{s,j}^{Pre}, V_{s,j}^{Mir}$	Memory amount in the stop-and-copy phase for any $VM_j/Container_j$ in pre-copy and MiGrror migration methods
M_j	Memory size of any $VM_j/Container_j$, $\forall\, j \in \{1, \dots, p\}$
$\bar{r}_j$	Average transfer rate (average bandwidth) available for any $VM_j/Container_j$ in pre-copy migration method
$r_{i,j}$	Available transfer rate (bandwidth) during event i for any $VM_j/Container_j$in MiGrror migration method
$r_{s,j}$	Available transfer rate (bandwidth) in the stop-and-copy phase for any $VM_j/Container_j$ in MiGrror migration method
$t_{i,j}^{Pre}$	Time to transfer $V_{i,j}^{Pre}$ in pre-copy migration, $\forall\, j \in \{1, \dots, p\}$ and $\forall\, i \in \{1, \dots, m\}$
$t_{i,j}^{Mir}$	Time to transfer $V_{i,j}^{Mir}$ in MiGrror migration, $\forall\, j \in \{1, \dots, p\}$ and $\forall\, i \in \{1, \dots, n\}$
$\bar{d}_j$	Average memory dirtying rate for any $VM_j/Container_j$ in pre-copy migration method
$d_{i,j}$	Memory dirtying rate during event i for any $VM_j/Container_j$in MiGrror migration method
TM_j^{Pre}, TM_j^{Mir}	Total migration time for any $VM_j/Container_j$ for both pre-copy and MiGrror methods
TD_j^{Pre}, TD_j^{Mir}	Downtime for any $VM_j/Container_j$ for both pre-copy and MiGrror migration methods
λ_j	$\bar{d}_j/\bar{r}_j$, $\forall\, j \in \{1, \dots, p\}$
$\lambda_{i,j}$	$d_{i-1,j}/r_{i,j}$, $\forall\, j \in \{1, \dots, p\}$ and $\forall\, i \in \{1, \dots, m\}$
$\lambda_{s,j}$	$d_{n,j}/r_{s,j}$
B	Total maximum bandwidth reserved for the entire migration between two MEC nodes
TA_j^{Pre}, TA_j^{Mir}	Migration overhead, amount of data to be migrated during migration, for any $VM_j/Container_j$ in pre-copy and MiGrror methods
τ	The inter-iteration delay in pre-copy migration method
$\tau_{i,j}$	The time between two consecutive events i and $i+1$ in MiGrror migration method for $VM_j/Container_j$

As soon as i reaches m, the final round, i.e. stop-and-copy, begins. We assume that all VMs and containers have m rounds and that every single $VM_j/container_j$ stops execution after m rounds before the stop-and-copy phase. Furthermore, the time necessary for the transfer round i for $VM_j/container_j$, i.e., $t_{1,j}^{Pre}$, may be recursively calculated using equations (4.1) and (4.2) as follows:

$$t_{1,j}^{Pre} = \frac{V_{1,j}^{Pre}}{\bar{r}_j} + \tau = \frac{M_j}{\bar{r}_j} + \tau \tag{4.3}$$

where τ is the inter-iteration delay shown in Figure 2.

$$\begin{aligned} t_{2,j}^{Pre} &= \frac{V_{2,j}^{Pre}}{\bar{r}_j} + \tau = \frac{\bar{d}_j t_{1,j}^{Pre}}{\bar{r}_j} + \tau = \lambda_j t_{1,j}^{Pre} + \tau = \lambda_j \left(\frac{M_j}{\bar{r}_j} + \tau\right) + \tau \\ &= \frac{\lambda_j M_j}{\bar{r}_j} + \lambda_j \tau + \tau = \frac{\lambda_j M_j}{\bar{r}_j} + \tau(1 + \lambda_j) \end{aligned} \tag{4.4}$$

where λ_j is the average memory dirtying rate divided by the average transfer rate, $\bar{d}_j/\bar{r}_j$, for any $VM_j/container_j$.

$$t_{3,j}^{Pre} = \frac{V_{3,j}^{Pre}}{\bar{r}_j} + \tau = \frac{\bar{d}_j t_{2,j}^{Pre}}{\bar{r}_j} + \tau = \lambda_j t_{2,j}^{Pre} + \tau = \frac{M_j}{\bar{r}_j} {\lambda_j}^2 + \tau\left(\frac{1 - {\lambda_j}^3}{1 - \lambda_j}\right) \tag{4.5}$$

...

$$t_{i,j}^{Pre} = \frac{V_{i,j}^{Pre}}{\bar{r}_j} + \tau = \lambda_j t_{i-1,j}^{Pre} + \tau = \frac{M_j}{\bar{r}_j} {\lambda_j}^{i-1} + \tau\left(\frac{1 - {\lambda_j}^i}{1 - \lambda_j}\right) \tag{4.6}$$

Thus, the migration **downtime** for $VM_j/container_j$, i.e., TD_j^{Pre}, may be calculated as:

$$TD_j^{Pre} = \frac{V_{s,j}^{Pre}}{\bar{r}_j} = \frac{\bar{d}_j t_{m,j}^{Pre}}{\bar{r}_j} = \lambda_j t_{m,j}^{Pre} = \frac{M_j}{\bar{r}_j} {\lambda_j}^m + \lambda_j \tau\left(\frac{1 - {\lambda_j}^m}{1 - \lambda_j}\right) \tag{4.7}$$

where $V_{s,j}^{Pre}$ represents the data during hand-off for any $VM_j/container_j$. We use a maximum value here since containers are dependent on, and interact with, one another, and some must wait for others to respond to each user. The maximum amount of downtime during pre-copy migration is expressed as follows:

$$T_{Downtime}^{Pre} = max\{TD_1^{Pre},\ TD_2^{Pre}, TD_3^{Pre},\ \dots,\ TD_p^{Pre}\} \tag{4.8}$$

Further, the total **migration time** for every $VM_j/Container_j$, i.e., TM_j^{Pre} with m number of pre-copy transfer rounds followed by a final stop-and-copy round, is given by:

$$\begin{aligned} TM_j^{Pre} &= \sum_{i=1}^{m} t_{i,j}^{Pre} + TD_j^{Pre} = \left(\frac{M_j}{\bar{r}_j}\sum_{i=1}^{m}(\lambda_j)^{i-1} + \frac{\tau}{1-\lambda_j}\sum_{i=1}^{m}(1-\lambda_j)^{i}\right) + TD_j^{Pre} \\ &= \frac{M_j}{\bar{r}_j}\frac{1-\lambda_j{}^m}{1-\lambda_j} + \tau\frac{m(1-\lambda_j)-\lambda_j(1-\lambda_j{}^{m+1})}{(1-\lambda_j)^2} + TD_j^{Pre} \\ &= \frac{M_j}{\bar{r}_j}\frac{1-\lambda_j{}^m}{1-\lambda_j} + \tau\frac{m(1-\lambda_j)-\lambda_j(1-\lambda_j{}^{m+1})}{(1-\lambda_j)^2} + \frac{M_j}{\bar{r}_j}\lambda_j{}^m \\ &+ \lambda_j\tau\left(\frac{1-\lambda_j{}^m}{1-\lambda_j}\right) \end{aligned} \tag{4.9}$$

The maximum migration time by assigning network transfer rate $\bar{r}_j$ for each $VM_j/container_j$ in pre-copy migration can be expressed as:

$$T_{migration}^{Pre} = max\{TM_1^{Pre},\ TM_2^{Pre}, TM_3^{Pre},\ \dots,\ TM_p^{Pre}\} \tag{4.10}$$

Thus, the total amount of data, **migration overhead**, to be sent during migration for any $VM_j/container_j$, i.e., TA_j^{Pre}, is given by:

$$\begin{aligned}TA_j^{Pre} &= \sum_{i=1}^{m} V_{i,j}^{Pre} + V_{s,j}^{Pre} = M_j + \sum_{i=2}^{m} \bar{d}_j t_{i-1,j}^{Pre} + \bar{d}_j t_{m,j}^{Pre} \\ &= M_j + \left(\bar{d}_j \sum_{i=2}^{m} t_{i-1,j}^{Pre}\right) + \bar{d}_j t_{m,j}^{Pre} \\ &= M_j + M_j\lambda_j \frac{1-\lambda_j{}^m}{1-\lambda_j} + \tau\bar{d}_j \frac{m(1-\lambda_j) - \lambda_j(1-\lambda_j{}^{m+1})}{(1-\lambda_j)^2} \\ &+ \bar{d}_j t_{m,j}^{Pre} \\ &= M_j + M_j\lambda_j \frac{1-\lambda_j{}^m}{1-\lambda_j} + \tau\bar{d}_j \frac{m(1-\lambda_j) - \lambda_j(1-\lambda_j{}^{m+1})}{(1-\lambda_j)^2} \\ &+ M_j\lambda_j{}^m + \bar{d}_j\tau\left(\frac{1-\lambda_j{}^m}{1-\lambda_j}\right)\end{aligned} \tag{4.11}$$

The total migration overhead during migration for all VMs/containers is expressed as follows:

$$Data_{migration}^{Pre} = \{TA_1^{Pre} + TA_2^{Pre} + TA_3^{Pre} + \cdots + TA_p^{Pre}\} \tag{4.12}$$

For downtime, migration time, and total migration overhead, subject to:

$$\sum_{j=1}^{p} \bar{r}_j \leq B \tag{4.13}$$

where B is the total maximum reserved bandwidth for the entire migration between two edge (MEC) nodes, and:

$$0 \leq \bar{r}_j \leq B \text{ and } \lambda_j < 1 \tag{4.14}$$

4.4 Mathematical Model of Multiple VMs/Containers Migration using Non-average Parameter Values – MiGrror Model

This section describes the MiGrror [28] migration model, which for the first time, uses non-average parameter values for transfer rate (bandwidth), memory dirtying rate, and VM/container size during migration. Downtime, migration time, and migration overhead (the amount of data that must be transferred during migration) of multiple VMs/containers are all modelled. Since stated parameters, such as memory dirtying rate, are likely to change and do not have a fixed value during the migration process, the use of non-average parameter values can lead to more accurate migration time and downtime results. Most researchers used average parameter values. However, our proposed migration time and downtime models revealed that the results would be different if these parameters were higher or lower at the beginning, middle, and end of the migration process with the same average parameter value, as discussed in the introduction. The problem with relying solely on average parameter values is that the migration and downtime results will be identical, while these results for non-average parameter values will vary. This is a critical limitation of the current models.

To illustrate the previously discussed **limitation of the current migration models**, consider, for instance, two migration processes with the same average parameter values (e.g., transfer rate and memory dirtying rate) but with varying values throughout the migration. In this example, if the memory dirtying rate increases at the end of the migration, it can significantly impact downtime since downtime occurs at the end of the migration [6], and this increased memory dirtying rate requires transferring a higher-than-average dirty memory rate. The same holds true if the transfer rate drops at the end of the migration and the dirtied memory data must transfer at a slower-than-average transfer rate. In both cases above, using average parameter values yields the same results, whereas using actual (non-average) parameter values yields different results. In this case, data must be transferred at a slower-than-average rate.

The same situation as described above occurs due to using average parameter values in current models, while the actual (non-average) transfer rate was lower at the beginning in this example. As a result, if input parameters are higher or lower than their average value at crucial points during the migration, they can severely affect downtime and migration time. These situations deteriorate when critical parameters are extremely higher or lower than average, and in these cases, they significantly affect downtime and migration time.

These findings imply that it is advantageous to know when the value of a parameter has a more significant impact on the result and that we can control the result by precisely selecting other parameter values, when it is possible, to achieve desired results. As a result, using non-average parameter values can provide greater insight and control over the migration process, particularly for 5G and 6G networks. However, **these critical conditions are hidden** when constructing the edge computing environment using only average input parameter values. As a result, **new migration models in edge computing are required** to achieve more accurate and precise results for multi-container mission-critical IoT applications that consider user mobility.

To address these issues, we propose developing **new models for multi-container migrations** which support **mobility** and **more accurately** characterize migration downtime, migration time, and migration overhead, than current models for ultra-low and real-time applications. The models will employ explicit input parameter **details** which occur throughout the migration process as well as the requirements of applications that provide services to end users. These input parameters include memory dirtying rate, transfer rate, and container size.

Using **non-average** input parameter values can prevent failure related to a lack of comprehension of the stated crucial conditions. In addition, using non-average parameters can provide a more encouraging understanding, including **additional details of output parameter values**, specifically **downtime** and **migration time**, throughout the migration process. Furthermore, the migration process is **more precisely controlled** by selecting available input parameters, namely transfer rate, in order to achieve the desired results, particularly for mobility support and latency-sensitive applications in 5G/6G mobile networks.

To the best of our knowledge, the MiGrror migration method [28] is the only option for using non-average parameter values during migration, as it transfers dirty memory as soon as it becomes available. Therefore, we will employ the MiGrror migration method in our models.

Table 1 describes the modelling parameters. In the table, M_j, $d_{i,j}$, and $r_{i,j}$ represent the memory size, memory dirtying rate during event i, and available transfer rate during event i in migration for any $VM_j/container_j$, respectively. The specified parameters affect migration time $\left(TM_j^{Mir}\right)$ and downtime $\left(TD_j^{Mir}\right)$. Higher M_j and $d_{i,j}$ levels increase migration time and downtime, whereas higher $r_{i,j}$ levels decrease migration time and downtime. It is also critical that different levels of $r_{i,j}$ and $d_{i,j}$ occur at the beginning, end (during hand-off), as well as the middle of the migration process. The $r_{i,j}$ and $d_{i,j}$ levels are more critical for migration time at the beginning of the migration process, and these levels are more critical for downtime at the end of the migration process. The levels of both stated parameters have the least impact on migration time and downtime in the middle. The remainder of this section presents the MiGrror model.

During the first event, the entire memory of any VM_j or $container_j$ is transferred from the source to the destination. So, the data sent during event one, i.e., $V_{1,j}$, can be computed using the equation given below:

$$V_{1,j}^{Mir} = M_j \tag{4.15}$$

The memory becomes dirty throughout the transfer as the $VM/container$ remains active at the source during MiGrror migration. Then, memory-change events transfer just the memory that was dirtied during the preceding event. The amount of data sent at event i for every $VM_j/container_j$ is:

$$V_{i,j}^{Mir} = d_{i-1,j} t_{i-1,j}^{Mir} \tag{4.16}$$

Furthermore, the time necessary for the transfer event i for $VM_j/container_j$, i.e., $t_{1,j}^{Mir}$, may be recursively calculated using equations (4.15) and (4.16) as follows:

$$t_{1,j}^{Mir} = \frac{V_{1,j}}{r_{1,j}} + \tau_{1,j} = \frac{M_j}{r_{1,j}} + \tau_{1,j} \tag{4.17}$$

where $r_{1,j}$ is the available transfer rate during the first event in migration for $VM_j/container_j$, and $\tau_{1,j}$ is the time between the first and second consecutive events of MiGrror migration for $VM_j/container_j$.

$$\begin{aligned} t_{2,j}^{Mir} &= \frac{V_{2,j}^{Mir}}{r_{2,j}} + \tau_{2,j} = \frac{d_{1,j} t_{1,j}^{Mir}}{r_{2,j}} + \tau_{2,j} = \lambda_{2,j} t_{1,j}^{Mir} + \tau_{2,j} \\ &= \lambda_{2,j}\left(\frac{M_j}{r_{1,j}} + \tau_{1,j}\right) + \tau_{2,j} = \lambda_{2,j}\frac{M_j}{r_{1,j}} + \lambda_{2,j}\tau_{1,j} + \tau_{2,j} \end{aligned} \tag{4.18}$$

$$\begin{aligned} t_{3,j}^{Mir} &= \frac{V_{3,j}^{Mir}}{r_{3,j}} + \tau_{3,j} = \frac{d_{2,j} t_{2,j}^{Mir}}{r_{3,j}} + \tau_{3,j} = \lambda_{3,j} t_{2,j}^{Mir} + \tau_{3,j} \\ &= \lambda_{3,j}\lambda_{2,j}\frac{M_j}{r_{1,j}} + \lambda_{3,j}\lambda_{2,j}\tau_{1,j} + \lambda_{3,j}\tau_{2,j} + \tau_{3,j} \end{aligned} \tag{4.19}$$

$$\begin{aligned} t_{4,j}^{Mir} &= \frac{V_{4,j}^{Mir}}{r_{4,j}} + \tau_{4,j} = \lambda_{4,j} t_{3,j}^{Mir} + \tau_{4,j} \\ &= \lambda_{4,j}\lambda_{3,j}\lambda_{2,j}\frac{M_j}{r_{1,j}} + \lambda_{4,j}\lambda_{3,j}\lambda_{2,j}\tau_{1,j} + \lambda_{4,j}\lambda_{3,j}\tau_{2,j} + \lambda_{4,j}\tau_{3,j} + \tau_{4,j} \end{aligned} \tag{4.20}$$

...

$$t_{i,j}^{Mir} = \frac{V_{i,j}^{Mir}}{r_{i,j}} + \tau_{i,j} = \lambda_{i,j} t_{i-1,j}^{Mir} + \tau_{i,j} \tag{4.21}$$

where $\tau_{i,j}$ is the time between two consecutive events i and $i+1$ in MiGrror migration for $VM_j/container_j$. Moreover, $\lambda_{i,j}$ is the memory dirtying rate of the previous event, event $i-1$, divided by the transfer rate of the current event, event i, for any $VM_j/container_j$, and is equal to $d_{i-1,j}/r_{i,j}$. Then:

$$\begin{aligned} t_{i,j}^{Mir} &= \lambda_{i,j}\lambda_{i-1,j} \ldots \lambda_{2,j} \frac{M_j}{r_{1,j}} + \lambda_{i,j}\lambda_{i-1,j} \ldots \lambda_{2,j}\tau_{1,j} + \lambda_{i,j}\lambda_{i-1,j} \ldots \lambda_{3,j}\tau_{2,j} \\ &\quad + \lambda_{i,j}\lambda_{i-1,j} \ldots \lambda_{4,j}\tau_{3,j} + \cdots + \lambda_{i,j}\lambda_{i-1,j}\tau_{i-2,j} + \lambda_{i,j}\tau_{i-1,j} + \tau_{i,j} \end{aligned} \tag{4.22}$$

Thus, the migration **downtime** for $VM_j/container_j$ i.e., TD_j^{Mir} can be calculated as:

$$\begin{aligned} TD_j^{Mir} &= \frac{V_{s,j}^{Mir}}{r_{s,j}} = \frac{d_{n,j} t_{n,j}^{Mir}}{r_{s,j}} = \lambda_{s,j} t_{n,j}^{Mir} \\ &= \lambda_{s,j} \left(\prod_{i=2}^{n} \lambda_{i,j} \frac{M_j}{r_{1,j}} + \prod_{i=2}^{n} \lambda_{i,j}\, \tau_{1,j} + \prod_{i=3}^{n} \lambda_{i,j}\, \tau_{2,j} + \prod_{i=4}^{n} \lambda_{i,j}\, \tau_{3,j} \right. \\ &\quad \left. + \cdots + \lambda_{n,j}\lambda_{n-1,j}\tau_{n-2,j} + \lambda_{n,j}\tau_{n-1,j} + \tau_{n,j} \right) \end{aligned} \tag{4.23}$$

where $V_{s,j}^{Mir}$ and $r_{s,j}$ represent the data sent and the available transfer rate during hand-off, respectively, and $\lambda_{s,j}$ is $d_{n,j}/r_{s,j}$ for any $VM_j/container_j$. We use a maximum value here since containers are dependent on, and interact with, one another, and some must wait for others to respond to each user. The maximum amount of downtime during MiGrror migration is expressed as follows:

$$T_{Downtime}^{Mir} = max\{TD_1^{Mir},\ TD_2^{Mir}, TD_3^{Mir},\ \ldots,\ TD_p^{Mir}\} \tag{4.24}$$

Further, the total MiGrror **migration time** for every $VM_j/container_j$, i.e., TM_j^{Mir} with n number of transfer events followed by a final stop-and-copy event, is given by:

$$TM_j^{Mir} = \sum_{i=1}^{n} t_{i,j}^{Mir} + TD_j^{Mir} \tag{4.25}$$

The maximum migration time by assigning network transfer rate $r_{i,j}$ for each $VM_j/Container_j$ in MiGrror migration can be expressed as:

$$T_{migration}^{Mir} = max\{TM_1^{Mir},\ TM_2^{Mir}, TM_3^{Mir},\ \ldots,\ TM_p^{Mir}\} \tag{4.26}$$

Thus, the total amount of data, **migration overhead**, to be sent during MiGrror migration for any $VM_j/container_j$, i.e., TA_j^{Mir}, is given by:

$$TA_j^{Mir} = \sum_{i=1}^{n} V_{i,j}^{Mir} + V_{s,j}^{Mir} = M_j + \sum_{i=2}^{n} d_{i-1,j}t_{i-1,j}^{Mir} + d_{n,j}t_{n,j}^{Mir} \tag{4.27}$$

The total migration overhead during migration for all VMs/containers is expressed as follows:

$$Data_{migration}^{Mir} = \{TA_1^{Mir} + TA_2^{Mir} + TA_3^{Mir} + \cdots + TA_p^{Mir}\} \tag{4.28}$$

For downtime, migration time, and total migration overhead, subject to:

$$\sum_{j=1}^{p} r_{i,j} \le B \tag{4.29}$$

where B is the total maximum reserved bandwidth for the entire migration between two edge (MEC) nodes, and:

$0 \leq r_{i,j} \leq B$ and $\lambda_{i,j} < 1$ (4.30)

4.5 Performance Evaluation and Discussions

Several parameters may affect migration performance, including container size, transfer rate, and memory dirtying rate. This section investigates how various parameters affect migration performance. We use the CSAP dataset [84] and our experiments to model pre-copy and MiGrror migration methods. The migration time, downtime, and migration overhead (transferred data) numbers given in the results are the averages of ten distinctive migration runs of each model using the Python code we developed. The pre-copy method terminates when the number of rounds (iterations) reaches a predefined threshold of 10 rounds $(m = 10)$. To be fair in our comparisons, we trigger the hand-off for both migration methods at the same time. Furthermore, we use 20 VMs/containers $(p = 20)$ to migrate from the source to the destination during migration. We divide the total bandwidth (B) by the number of VMs/containers (p) for non-dataset values, and the transfer rate for each $VM_j/container_j$ is the same. Additional considered parameters vary and are detailed in the subsections that follow.

Since we, unlike other researchers, consider non-average parameter values, we calculate the minimum, maximum, median, average, and standard deviations of the stated parameters to examine the dataset in more depth. The transfer rate $(r_{i,j})$ ranges between a minimum of 50 $megabits\ per\ seconds$ $(Mbps)$ and a maximum of 150 $Mbps$. The median, average, and standard deviation are 108.5, 105.385, and 29.79, respectively. The memory dirtying rate $(d_{i,j})$ is another parameter for which we consider non-average values. The minimum value is 0.02323 $Mbps$, and the maximum is 145.076 $Mbps$. The median and average memory dirtying rates are 18.52 and 28.979, respectively, with a standard deviation of 31.89. Memory sizes for VMs and containers (M_j) range from a minimum of 249.41 $megabytes$ (MB) to a maximum of 4080.94 MB. The median, average, and standard deviation are 813.326, 1049.83, and 625.268, respectively. The final parameter considered is $\lambda_{i,j}$, which is the memory dirtying rate divided by the transfer rate. During the migration process, the minimum and maximum $\lambda_{i,j}$ are 0.000196718 and 0.999938322. The median and average are 0.166656325 and 0.274938801, respectively, with a standard deviation of 0.292347702.

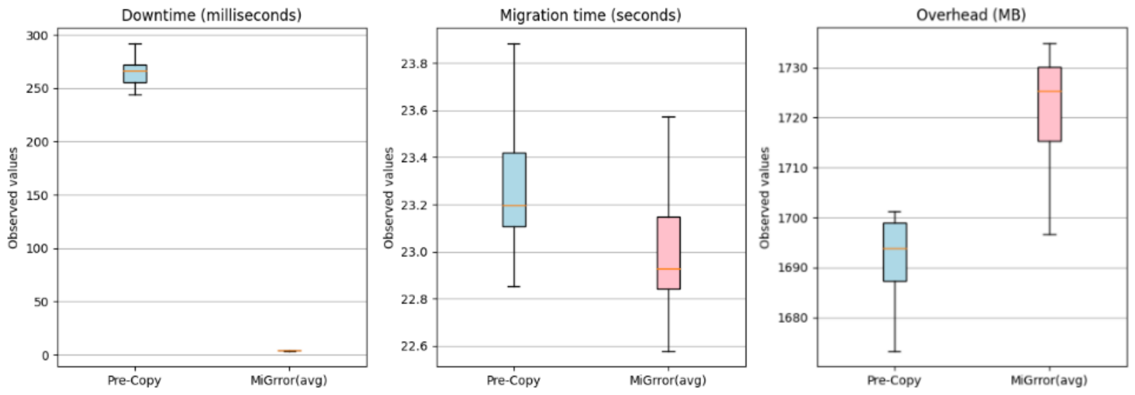

Figure 4.3 Comparison of the pre-copy and MiGrror migration methods using the dataset's average parameter values (blue: Pre-Copy, red: MiGrror) (left: Downtime, middle: Migration Time, right: Migration Overhead).

4.5.1 Results using Average Parameter Values and the Dataset

We investigate the performance of the pre-copy and MiGrror using average parameter values of VM/container size, memory dirtying rate, and transfer rate for each VM/container in this subsection. Figure 4.3 illustrates that downtime is the most noticeable distinction between the pre-copy and MiGrror. In our experiments, the median downtime for pre-copy is 265.924 ms, while the median downtime for MiGrror is less than 1 ms. The pre-copy downtime is unacceptable for future 5G and 6G delay-sensitive applications since it results in prolonged service interruptions during migration. The MiGrror technique, on the other hand, generates much less downtime than the pre-copy technique since it uses live mirroring between the source and destination.

Furthermore, as illustrated in the figure, the migration time using the MiGrror technique is less than that of the pre-copy technique, with maximums of 23.65 $seconds$ (s) and 23.93 s, respectively. Using non-average parameter values in the subsequent subsection, non-average parameter value subsection, reveals a significantly larger difference. With the shorter migration time, resources at the source can be made available to other containers more quickly.

Although the MiGrror reduces downtime and migration time, it comes at a cost: migration overhead. However, the cost is negligible. Since the MiGrror method synchronizes faster than the pre-copy method, it requires more bandwidth to mirror changes from the source to the destination. The MiGrror consumes more bandwidth than pre-copy, but it is only a negligible 1.16% increase in total migration overhead. Despite the additional overhead of 1.16%, downtime and migration time are reduced.

4.5.2 Non-average Parameter Value Results using the Dataset

This subsection examines the performance of the pre-copy and MiGrror techniques. For the MiGrror method, we use average and non-average parameter values for memory dirtying rate, transfer rate, and VM/container size and compare them to the results of the classical pre-copy model, which only uses average parameter values.

We cannot compare non-average parameter values of MiGrror and pre-copy since the pre-copy uses a limited number of rounds, and the stated parameters can change many times during each round. Therefore, only non-average parameter values of the MiGrror migration method, and average parameter values of both the pre-copy and MiGrror migration methods, are presented in Figure 4.4.

Figure 4.4 is the best representation of why we must consider non-average parameter values in contrast to the traditional view of using only average parameter values. The figure depicts that fluctuations of downtime, migration time, and migration overhead are unanticipated even when using the same method, MiGrror. To clarify, when using average and non-average parameter values, neither the mean nor the median of the results is identical or even close. The same pattern holds true for the maximum and the minimum of the results.

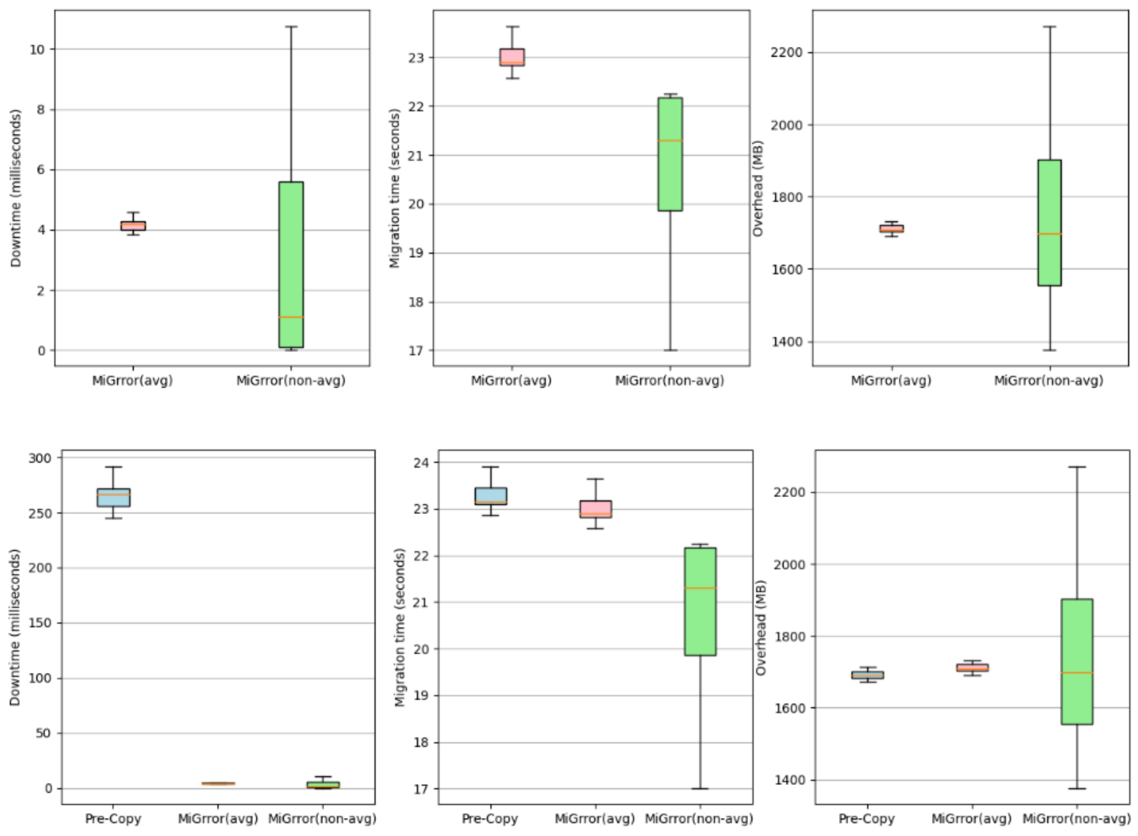

Figure 4.4 Comparison of the dataset's average and non-average parameter values using MiGrror migration (left: Downtime, middle: Migration Time, right: Migration Overhead, blue: pre-copy, red: MiGrror using average values, green: MiGrror using non-average values).

Consider the MiGrror results with average and non-average parameter values. Although the median downtime for average parameter values is roughly four times that of non-average parameter values, the maximum downtime using average parameter values is roughly 50% of non-average parameter values. Furthermore, the maximum migration time using average parameter values and non-average parameter values differs by about 25%, while the minimum migration time using average and non-average parameter values differs by more than 32%. Moreover, when using average parameter values, the standard deviation of the downtime is only 0.21, whereas when using non-average parameter values, it is 3.81. This high standard deviation results from memory dirtying rate of the last event, which directly affects downtime, according to the model and results. The standard deviation of migration time follows the same pattern, when using average and non-average parameter values at 0.30 and 3.46, respectively. The difference in standard deviation between average and non-average parameter values for migration overhead is substantial; 22.36 and 346.33, respectively. Consequently, using non-average parameter values provides us with a new perspective to improve future applications and prevent unanticipated outcomes, such as those shown in Figure 4.4, when implementing them in the real world. These findings highlight the practicality of using non-average parameter values when analyzing data.

We demonstrate that the MiGrror migration overhead is only 0.5% greater than the pre-copy migration overhead on average, which is an additional advantage of utilizing non-average parameter values. However, when using average parameter values, the MiGrror overhead amount is 1.19% greater than the pre-copy migration overhead on average.

4.5.3 Non-dataset Parameter Values

This section studies the performance of the pre-copy and MiGrror models in terms of several parameters: container size, memory dirtying rate, and transfer rate. We use average parameter values with synthesized data in this subsection to make a fair comparison since we cannot calculate the pre-copy results using the non-average parameter values as explained in the previous subsection. Each subsection focuses on one of these three parameters, with the final subsection focusing on $\lambda_{i,j}$ variations.

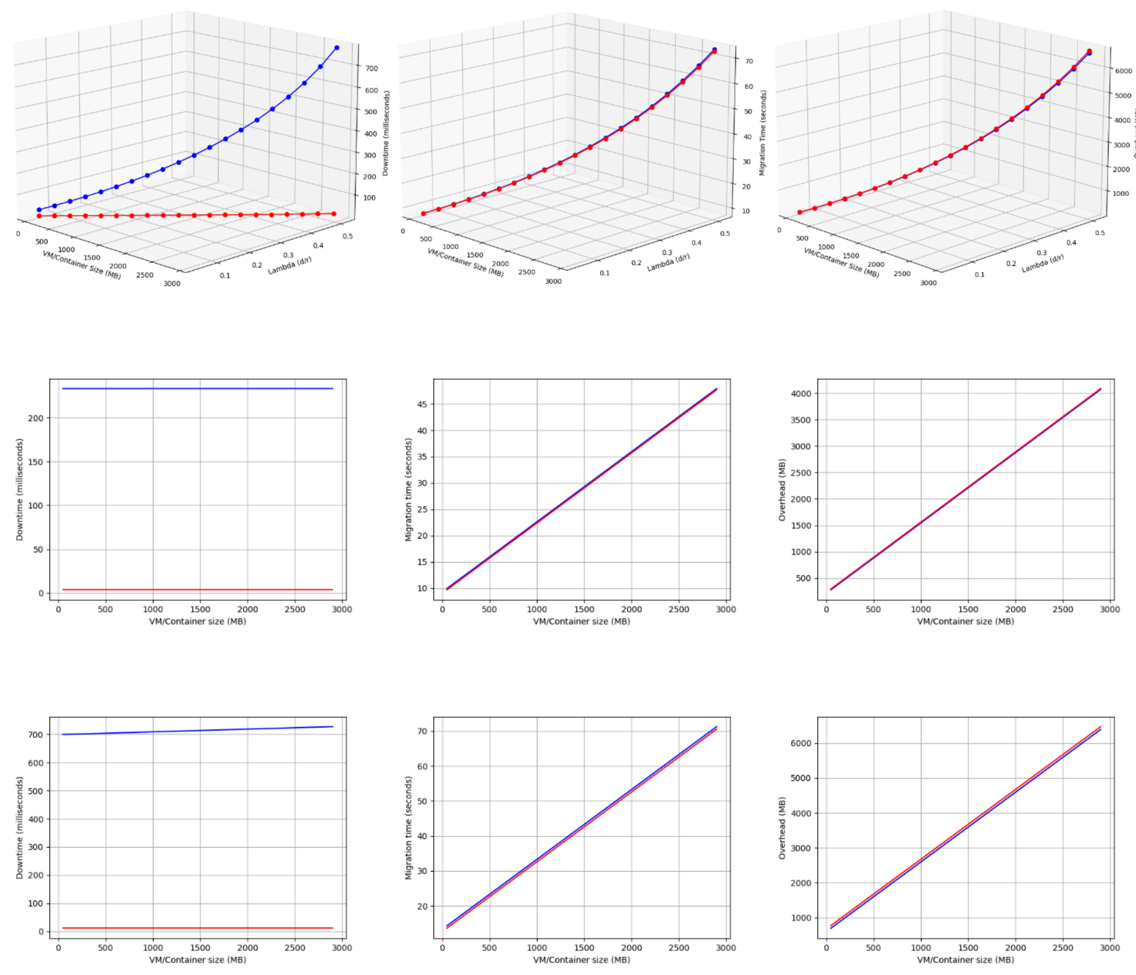

Figure 4.5 Downtime, Migration Time, and Migration Overhead as functions of VM/container size for the pre-copy and MiGrror migration methods using non-dataset parameter values (blue: Pre-Copy, red: MiGrror) (top: $\lambda_{i,j}$ varies between 0.04 and 0.515, middle: $\lambda_{i,j}$ is constant and set to 0.25, bottom: $\lambda_{i,j}$ is constant and set to 0.5)

4.5.3.1 Performance based on Varying VM/Container Size

Figure 4.5 illustrates the impact of varying VM/container sizes on downtime, migration time, and migration overhead. The first row of 3-D figures represents the variations in VM/container size in terms of different $\lambda_{i,j}$ ratios and the corresponding results. The figure also depicts the downtime, migration time, and migration overhead for various VM/container sizes with $\lambda_{i,j}$ set to 0.25 in the middle row and 0.5 in the bottom row.

This figure shows that **increasing the VM/container** size has **no meaningful** effect on MiGrror **downtime**. Only with high $\lambda_{i,j}$ ratios does pre-copy downtime increase, but the increase is negligible since the difference is less than 4% between the smallest and largest VM/container sizes. However, increasing the VM/container size increases migration time and overhead. It is also evident that the difference between migration time and overhead of the researched methods becomes more apparent with a higher $\lambda_{i,j}$. In addition, as shown in the figure, migration time and overhead increase as the VM/container size increases.

4.5.3.2 Performance based on Variation of Transfer Rate

Figure 4.6 illustrates how various transfer rates affect downtime, migration time, and migration overhead for the pre-copy and MiGrror migration methods. We consider 200 MB as a typical size for a container which also is considered a lightweight VM. Furthermore, we consider that there is a 1000 $Mbps$ total bandwidth for all VMs/containers. The VM/container size and memory dirtying rate are fixed at 200 MB and 50 $Mbps$ (since we assume there are 20 VMs/containers), respectively. The figure represents that increasing transfer rates reduces pre-copy downtime significantly, from 1421 $milliseconds$ (ms) to 70 ms, but the value is still relatively high. However, MiGrror downtime begins at 22 ms and ends at 1.1 ms, which is still superior to the pre-copy. In terms of migration time, the MiGrror is 5.65% less than that of pre-copy at low transfer rates, and less than 1% at high transfer ratios. In terms of migration overhead, the MiGrror consumes 14.61% more bandwidth than pre-copy at a low transfer rate, but this decreases to 4.15% more bandwidth consumption at a high transfer rate.

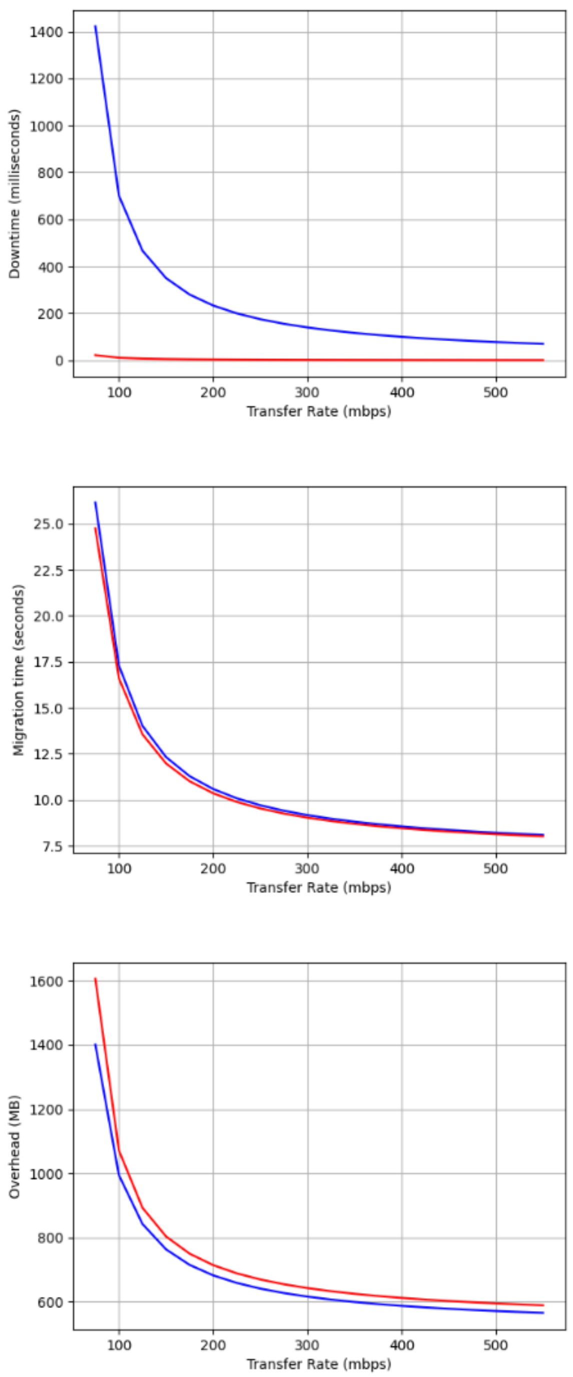

Figure 4.6 Downtime, Migration Time, and Migration Overhead as functions of Transfer Rate for the pre-copy and MiGrror migration methods using non-dataset parameter values (blue: Pre-Copy, red: MiGrror).

These findings imply that pre-copy downtime is still unacceptable for latency-sensitive mobile IoT applications, even with a high transfer rate. In addition, increasing the transfer rate converges the migration time and overhead of both methods. These results indicate that when the transfer rate is high, and the downtime is nonessential, such as in applications with no delay constraints. The performance of both approaches is nearly identical. However, when the transfer rate is limited, MiGrror outperforms pre-copy in terms of downtime and migration time.

Furthermore, as shown in the figure, migration time and overhead decrease as the transfer rate rises. The same pattern applies to pre-copy downtime but not MiGrror downtime. MiGrror downtime is low from the beginning.

4.5.3.3 Performance based on Variation of Memory Dirtying Rate

Figure 4.7 illustrates how various memory dirtying rates affect downtime, migration time, and migration overhead for the pre-copy and MiGrror migration methods. The VM/container size and transfer rate are fixed at 200 MB and 200 $Mbps$, respectively. The figure depicts that increasing memory dirtying rates increases pre-copy downtime significantly, from around 20 ms at a 5 $Mbps$ transfer rate to more than 700 ms at a 100 $Mbps$ transfer rate. However, MiGrror downtime begins at less than 1 ms and ends at 11 ms with the same transfer rates, which is still superior to the pre-copy. In terms of migration time, the MiGrror is 0.21% less than that of the pre-copy at a low transfer rate and the migration time improves to 4.72% at a high transfer rate. In terms of migration overhead, the MiGrror consumes 0.91% more bandwidth than pre-copy at a low transfer rate, but this increases to about 9% more bandwidth consumption at a high transfer rate.

These findings imply that pre-copy downtime is still unacceptable for latency-sensitive mobile IoT applications, even with a low memory dirtying rate. In addition, decreasing the transfer rate converges migration time and overhead of both methods. These results indicate that when the memory dirtying rate is low and the downtime is nonessential, such as in applications with no delay constraints, the performance of both approaches is nearly identical. However, when the memory dirtying rate is high, MiGrror outperforms pre-copy in terms of downtime and migration time.

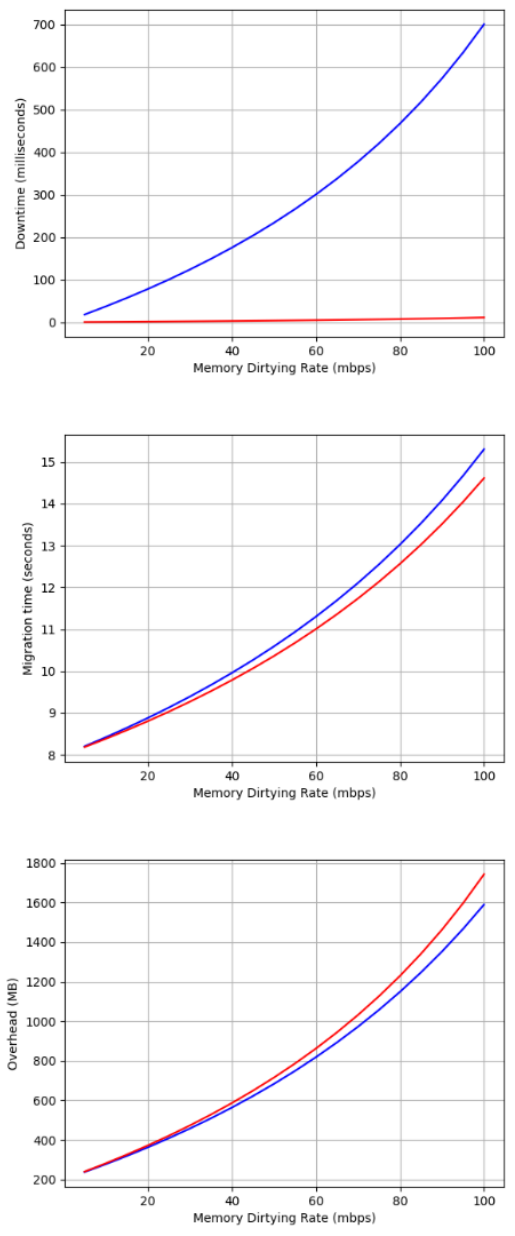

Figure 4.7 Downtime, Migration Time, and Migration Overhead as functions of Memory Dirtying Rate for the pre-copy and MiGrror migration methods using non-dataset parameter values (blue: Pre-Copy, red: MiGrror).

Furthermore, as shown in the figure, migration time and overhead increase as the memory dirtying rate rises. The same pattern applies to pre-copy downtime but not MiGrror downtime.

4.5.3.4 Performance based on variation of $\lambda_{i,j}$ (memory dirtying rate divided by transfer rate)

Figure 4.8 illustrates how various $\lambda_{i,j}$ rates affect downtime, migration time, and migration overhead for the pre-copy and MiGrror migration methods. The VM/container size is fixed at 200 MB. The figure shows increasing $\lambda_{i,j}$ rates increase pre-copy downtime significantly, from 60 ms at a 0.08 $\lambda_{i,j}$ rate to more than 700 ms at a 0.50 $\lambda_{i,j}$ rate. However, MiGrror downtime begins at 0.95 ms and ends at 11 ms with the same $\lambda_{i,j}$ rates, which is still superior to the pre-copy. In terms of migration time, the MiGrror is 0.69% less than that of the pre-copy at a low $\lambda_{i,j}$ rate and the difference is raised to more than 6.65% at 0.65 $\lambda_{i,j}$ rate. In terms of migration overhead, the MiGrror consumes 0.98% more bandwidth than the pre-copy at a 0.0275 $\lambda_{i,j}$ rate, but this increases to a 9.63% more bandwidth consumption at a 0.5 $\lambda_{i,j}$ rate.

These findings imply that while the MiGrror downtime is acceptable, pre-copy downtime is still unacceptable for latency-sensitive mobile IoT applications, even with a low $\lambda_{i,j}$ rate. In addition, decreasing the transfer rate converges migration time and overhead of both methods. These results indicate that when the $\lambda_{i,j}$ rate is low, and the downtime is nonessential, such as in applications with no delay constraints, the performance of both approaches is nearly identical. However, when the $\lambda_{i,j}$ rate is high, MiGrror outperforms pre-copy in terms of downtime and migration time. In fact, only when the $\lambda_{i,j}$ is low the pre-copy downtime is acceptable.

Furthermore, as shown in the figure, migration time and overhead increase as the $\lambda_{i,j}$ rate rises. The same pattern applies to pre-copy downtime but not MiGrror downtime.

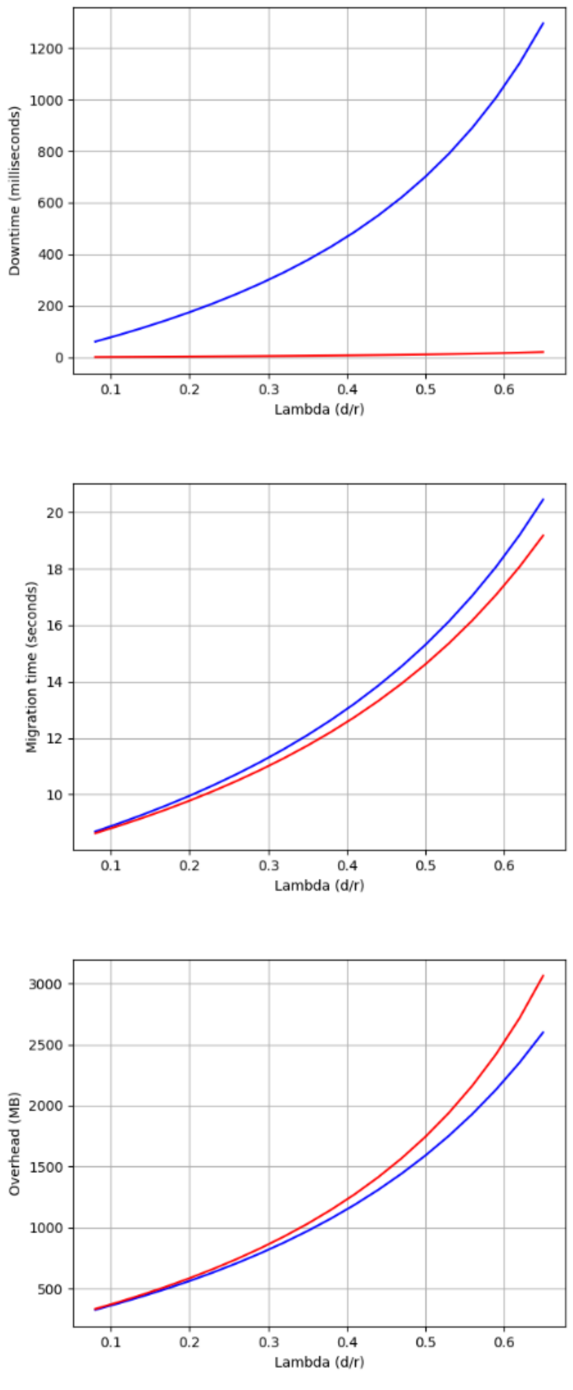

Figure 4.8 Downtime, Migration Time, and Migration Overhead as functions of λ_(i,j) for the pre-copy and MiGrror migration methods using non-dataset parameter values (blue: Pre-Copy, red: MiGrror).

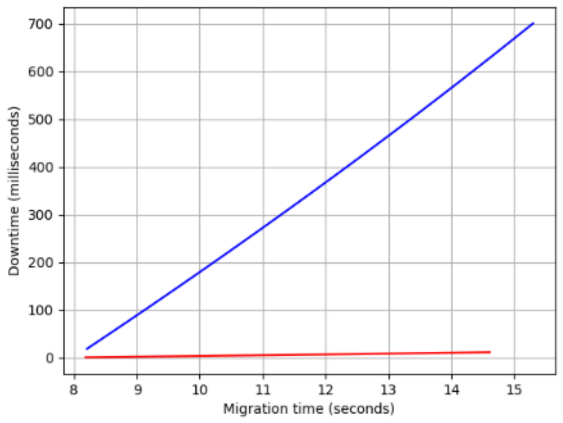

Figure 4.9 The pre-copy and MiGrror Downtime as a function of their Migration Time (blue: Pre-Copy, red: MiGrror).

4.5.3.5 Further Discussion

Figure 4.9 shows the relationship between downtime and migration duration for the pre-copy and MiGrror migration methods. As pre-copy downtime increases, migration time increases linearly. This finding means that we cannot use the pre-copy method after a certain value when application downtime or migration time is critical. For instance, if an application cannot tolerate more than 100 ms without interrupting users, we cannot use the pre-copy for that application, even if the migration time falls within that application's tolerance range. Using MiGrror, however, the rate of increase in downtime is significantly lower than its migration time. This finding indicates that by employing the MiGrror method, the MEC is able to service applications, as in the stated example, with a greater amount of migration time, since the MiGrror downtime is still within the tolerance range of the application.

Furthermore, as shown in figures 4.6, 4.7, and 4.8 as well as in the modelling results, the memory dirtying rate directly impacts the amount of memory transfer at each

MiGrror migration event. The MiGrror method reduced downtime by lowering the final amount of memory transfer, as shown in the results and equation (4.23).

Moreover, based on equations (4.7, 4.9, 4.23, 4.25) and figures 4.6 and 4.8, it is evident that with the same overall bandwidth, increasing the number of containers will reduce the transfer rate of each container, resulting in longer migration times. The transfer rate also slightly increases container downtimes. We skip the figure for the preceding argument since it can be inferred from figures 4.6 and 4.8.

4.6 Summary

In this Chapter, we model the MiGrror method for multi-service migrations for the first time. We use non-average parameter values as well as traditional average parameter values for downtime, migration time, and migration overhead for the first time. We illustrated that the MiGrror migration time and downtime outperform the pre-copy ones. As demonstrated in the Chapter, utilizing non-average parameters allows for a better understanding of what occurs during migration and more accurate results. Outputs can deviate drastically during crucial migration phases when actual (non-average) input parameters vary while their averages are unchanged. As previously stated, outputs during critical stages of the migration process can vary significantly if actual (non-average) input parameter values differ while their averages remain constant. Using non-average input parameter values in migration models can provide a refined, rational, migration analysis, which takes into account low response time and mobility, in multi-containerized edge computing environments. As a result, we use non-average parameter values to obtain more accurate results and comprehension of the environment.

Furthermore, we demonstrated that MiGrror improves user service continuity and availability. Keeping these in mind, using non-average parameter values enables us to understand the memory dirtying rate pattern during migration and enables us to adjust bandwidth accordingly to achieve the desired results with higher performance and better bandwidth utilization. In the following Chapter, we use MiGrror models to reduce downtime and migration time further by adjusting the bandwidth during migration.

Chapter 5

A Novel Bandwidth Allocation Strategy for Multi-service Migration

We intend to use bandwidth more efficiently. In this regard, we want to reduce downtime and migration time by adjusting the bandwidth at specific periods. To accomplish this, we employ the non-average MiGrror migration model, which enables us to adjust the bandwidth in accordance with the migration timing and provides a comprehensive understanding of the migration process. We utilize the MiGrror migration method and its downtime, migration time, and migration overhead models to propose a novel strategy for increasing bandwidth for short critical periods instead of maintaining increased bandwidth throughout the entire migration process. Our novel strategy drastically reduces downtime and migration time while enhancing bandwidth utilization by freeing up additional bandwidth during migration, which increases bandwidth availability for other services. The findings indicate that, even without using additional bandwidth, adjusting the timing of a bandwidth increase/decrease for services improves performance. The results are comparable to increasing the bandwidth for the entire migration duration with equivalent bandwidth consumption.

This chapter is derived from:

- **Arshin Rezazadeh**, Hanan Lutfiyya “A Novel Sustainable Bandwidth Allocation Strategy for Multiple Service Migration in 5G/6G Edge Computing,“ *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, IEEE, Dec. 2023, pp. 1211–1217

5.1 Introduction

Mobile IoT applications often require low response time and high bandwidth [1]. These applications include virtual reality (VR) and augmented reality (AR) in the context of the metaverse, as well as gaming and smart vehicle applications. Due to limited computing resources on the mobile device, there may be a need for application services to be hosted elsewhere. Cloud computing resources are an option, but the latency may be too high for some mobile IoT applications. To satisfy the need for a lower response time, computing power must be closer to the data sources [28], [111]. This is often referred to as edge computing. However, as mobile devices move, the latency may increase. This can be addressed by migrating services where a service runs on a virtual machine (VM) or a container. An application may consist of multiple services where there is dependency among the application services [5], [12]. With the increasing use of multi-containerized applications [5], it is crucial to ensure that all application containers minimize delays for latency-sensitive and real-time applications.

Migrating services in **parallel** can be used to reduce delays. For a real-time application, it is recommended that the maximum response time is 17 milliseconds to maintain end-user quality of experience at a desired level [3]. Most of the research has yet to achieve this response time [28]. One reason for the lengthy response time is that most research uses the **pre-copy** live migration method [56] to analyze the migration of multiple services in parallel, as found in [14], [17]. The input parameters that influence the migration includes **memory dirtying rate**, **transfer rate** (provisioned bandwidth), and **container size.** The values of these input parameters can change continuously throughout the migration for any of the containers of an application. **Downtime** and **migration time** are metrics used to assess migration [14]. Downtime is the time required to transfer a stopped container from one location to another location. Migration time is the time required to perform various transfers and preparation to restart the container at the new location, and migration overhead is the amount of data transferred between the two locations during the migration process [28]. Containers use their assigned bandwidth during migration; thus, the assigned bandwidth is unavailable until the container releases the bandwidth after the migration. Reducing migration time and downtime can be achieved by increasing the bandwidth for containers so that migration can be completed faster. However, using increased bandwidth throughout the entire duration of the migration may not be efficient [112].

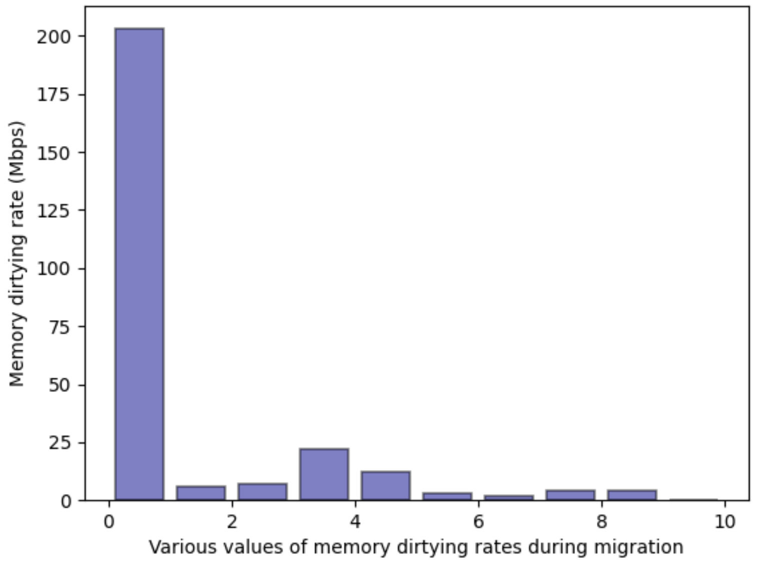

Figure 5.1 Various values of memory dirtying rates during migration.

The contribution of this Chapter is an approach that uses **planned bandwidth adjustments** at specified times to **reduce migration time and downtime**. The motivation is illustrated in Fig. 5.1, which depicts the various memory dirtying rates during a migration process derived from our experiments. The figure depicts ten values representing the data that must be transferred from the source to the destination during migration. The values presented in Fig. 5.1 were obtained from a snapshot of the experiments. In our experiments, we observed the **identical trend** displayed in Fig. 5.1 and described in reference [29].

With our approach we **increase the bandwidth** for short periods at **specified times**, i.e., the initial migration stage that transfers the entire container significantly impacts migration time. Therefore, increasing the bandwidth during this stage would have a higher impact than when the data being transferred is small. The bandwidth can also be increased during the migration's final stage (the stop-and-copy phase) when the final dirty memory is transferred. Increasing bandwidth during this stage reduces downtime. Our evaluation shows this strategy can reduce downtime, migration time, and migration overhead without additional bandwidth.

This Chapter is organized as follows: Section 5.2 provides related work, and Section 5.3 delivers the pre-copy limitations and MiGrror model [113]. Section 5.4 proposes a bandwidth strategy using the MiGrror method for multiple service migrations. Section 5.5 provides evaluations and discussions, and Section 5.6 summarizes this Chapter.

5.2 Related Work

Multiple strategies are employed by researchers to decrease migration and downtime. This includes reducing data transfer, using migration models and ML techniques, and adjusting the bandwidth**.**

Reduction of Data to be Transferred: One approach to reduce downtime is to reduce handoff data. Ma et al. [29] proposed a hierarchical container file system-based migration technique to reduce hand-off transfer size. Hand-off transfer size can be reduced by compression and transferring iterative memory difference. Machen et al. [35] proposed a framework that divides each application by layers and transmits only the missing layers only. By transmitting less data, this layered method reduces hand-off downtime. Puliafito et al. [2] proposed a smart helmet AR response time of 20 ms. This study used the pre-copy technique and compression before hand-off to reduce data transfer during hand-off. To reduce the total migration time of multiple VMs, Singh et al. [14] assigned compression and transfer rates to each VM using Geometric Programming by considering parameters such as VM size, memory. Ma et al. [29] proposed a hierarchical container file system-based migration technique to reduce hand-off transfer size. Hand-off transfer size can be reduced by compression and transferring iterative memory difference.

Bandwidth Adjustment: Bhardwaj et al. [112] highlights that allocating maximum bandwidth for VM migration leads to bandwidth wastage. Zhang et al. [13] proposed algorithms for optimizing load balancing and the migration of multi-container between cloud servers to balance server load. Migration is used to balance the load of servers. Zhang et al. [114], [115] proposed a bandwidth modification for the stop-and-copy phase to guarantee bandwidth allocation for the final stage of the migration based on the anticipated memory dirtying rate. This work considered migration of a single VM. To reduce the total migration time of multiple VMs, Singh et al. [14] assigned compression and transfer rates to each VM using Geometric Programming by considering parameters such as VM size, memory dirtying rate, transfer rate, and compression ratio of VMs.

Chaufournier et al. [116] use multi-path TCP to improve VM migration time by using several paths simultaneously for a connection, i.e., packets to and from an application traverse multiple network paths concurrently. The use of multiple paths provides the potential for increased bandwidth. However, this work is limited to focusing on a single VM/container without considering the VMs/containers that compose an application.

Liu et al. [22] describe an adaptive network bandwidth allocation algorithm that relies on predicting the dirtying rate within a specified time window. The prediction is based on measuring the page dirtying rate of each VM before the VM is deployed. This is used to model the memory access of the VM in order to determine bandwidth. Dharmaraj et al. [117] proposed a bandwidth allocation mechanism based on estimating data transmission cost, response time and throughput. This is used to determine the best server and communication path within a cloud environment. Zhang et al. [114] developed a theoretical model to analyze how much bandwidth is required to guarantee the required migration time and downtime. Sampling is done every ten milliseconds for 10 seconds. This sampled data is used to determine the distribution of dirtying pages' frequency to be used in a theoretical model to determine bandwidth. The second aspect of this work is that a new transport model is proposed since TCP focuses on fairly allocating bandwidth to flows and thus is difficult in practice [118]. Generally, the predictions are not always accurate [119].

Migration Models: Maheshwari et al. [18] created a cost model and used the Min-Max model to reduce migration costs for migrating multiple containers. Their model is based on container size and number, bandwidth, and memory dirtying rate at a mobility-supporting edge infrastructure. Satpathy et al. [15] used VM size, memory dirtying rate, and available bandwidth to compare migration performance for multiple VMs. He et al. [17] evaluated migration models on a software-defined network platform. Migration and downtime are considered the most crucial metrics in their model.

Limitations: Despite some progress in reducing downtimes, the research cited does not address the issue of time-sensitive applications adequately. Furthermore, there has not been a study that incorporates strategic bandwidth allocation during migration at specific times.

5.3 MiGrror Migration Model

This section presents a migration model that is used for the proposed bandwidth adjustment technique. The pre-copy migration method [56] transfers the complete state of a container from the edge node that currently hosts the container to the destination. The procedure is carried out in iterations, with memory changes transferred at the end of each iteration. The source node transfers dirty memory over multiple iterations, which are updated memory pages from the previous iteration. Once the hand-off signal is triggered, the source container stops memory modification and transfers the final dirty memory to the destination. According to section 5.2, the pre-copy method can result in lengthy downtimes, which is unsuitable for modern applications that require low latency. Another **problem** with using the **pre-copy** migration method is that pre-copy uses a limited number of rounds (iterations), e.g., 10-30 rounds [14]. Memory dirtying and transfer rates can fluctuate within short intervals in a round, i.e., these values can **change** multiple times during any pre-copy round [29], [113]. Adjusting the bandwidth for every change in memory (dirty memory) is infeasible when a high fluctuation is not always predictable. The bandwidth could be increased for the entire pre-copy process, but it may not be effective since many of the rounds may have little data to send [17].

The MiGrror migration method [28] is designed to reduce migration time and downtime compared to the pre-copy method. This is achieved by mirroring the container between the source and destination, as in servers [120]. The MiGrror method mirrors the data from the sender to the receiver as each memory change generates a transfer event, which can occur multiple times during the migration. MiGrror uses migration stages numbered from 1 to n, with the stop-and-copy phase labelled as s. The MiGrror migration method [28] uses every single parameter value during migration rather than waiting for a round to complete, as it transfers dirty memory as soon as it becomes available. As a result, less data is transferred during the hand-off process, resulting in **less downtime and migration time** compared to the pre-copy method [28]. Therefore, we use the MiGrror method to adjust bandwidth in short periods, resulting in better bandwidth utilization. This is due to MiGrror's fine granularity, which employs a high number with short durations of dirty memory transfers. The results [28] show that MiGrror has lower downtime. Therefore, our research will employ the MiGrror migration method to apply the proposed bandwidth strategy. While Pre-copy can produce good results, incorporating MiGrror can further elevate them.

We are studying a multiple-container MiGrror migration model [113], which we use in equations (5.1) – (5.3). The MiGrror model can use and observe the effects of every parameter value during migration. This provides information about the various stages of migration and how parameter values affect downtime and migration time. This information is then used to develop the proposed bandwidth allocation strategy. The M_j is the memory size of any $container_j$. The migration **downtime** for any $container_j$, i.e., TD_j^{Mir}, may be calculated as:

$$TD_j^{Mir} = \lambda_{s,j}\left(\prod_{i=2}^{n}\lambda_{i,j}\frac{M_j}{r_{1,j}} + \prod_{i=2}^{n}\lambda_{i,j}\,\tau_{1,j} + \prod_{i=3}^{n}\lambda_{i,j}\,\tau_{2,j} + \prod_{i=4}^{n}\lambda_{i,j}\,\tau_{3,j} + \cdots + \lambda_{n,j}\lambda_{n-1,j}\tau_{n-2,j} + \lambda_{n,j}\tau_{n-1,j} + \tau_{n,j}\right) \tag{5.1}$$

where $\tau_{i,j}$ is the time between two consecutive stages of migration i and $i+1$ $\left(t_{i,j}^{Mir}, t_{i-1,j}^{Mir}\right)$ in MiGrror migration for $container_j$. Moreover, $\lambda_{i,j}$ is the memory dirtying rate of the previous stage, stage $i-1$, divided by the transfer rate of the current event, stage i, for any $container_j$, and is equal to $d_{i-1,j}/r_{i,j}$. Accordingly, $\lambda_{s,j}$ is equal to $d_{n,j}/r_{s,j}$. Equation (5.1) shows that increasing the transfer rate, $r_{s,j}$, in the dominator of $\lambda_{s,j}$ will decrease downtime. The **migration time** for every $container_j$, i.e., TM_j^{Mir} with n number of transfer events followed by a final stop-and-copy event, is given by:

$$TM_j^{Mir} = \left(\sum_{i=1}^{n}\lambda_{i,j}\lambda_{i-1,j}\ldots\lambda_{2,j}\frac{M_j}{r_{1,j}} + \lambda_{i,j}\lambda_{i-1,j}\ldots\lambda_{2,j}\tau_{1,j} + \lambda_{i,j}\lambda_{i-1,j}\ldots\lambda_{3,j}\tau_{2,j} + \lambda_{i,j}\lambda_{i-1,j}\ldots\lambda_{4,j}\tau_{3,j} + \cdots + \lambda_{i,j}\lambda_{i-1,j}\tau_{i-2,j} + \lambda_{i,j}\tau_{i-1,j} + \tau_{i,j}\right) + TD_j^{Mir} \tag{5.2}$$

Equation (5.2) shows that increasing the transfer rate of the initial stage of this equation will reduce the migration time. This is due to M_j, the $container$ size to transfer, having a higher value than the other stages. The **migration overhead** to be sent during MiGrror migration for any $container_j$, i.e., TA_j^{Mir}, is given by:

$$TA_j^{Mir} = M_j + \sum_{i=2}^{n} d_{i-1,j} t_{i-1,j}^{Mir} + d_{n,j} t_{n,j}^{Mir} \tag{5.3}$$

Equation (5.3) shows that the migration overhead is related to the $t_{i,j}^{Mir}$ $(1 \leq i \leq n)$, which will be used in Section IV to clarify how the bandwidth strategy can reduce migration overhead.

5.4 Bandwidth Allocation Strategy

This section describes a bandwidth strategy based on the MiGrror migration model. A container is typically assigned a specific bandwidth, which is used throughout the migration process [14]. As a result, the assigned bandwidth amount is unavailable to other containers until the container releases it at the end of the migration. Having additional bandwidth during migration will decrease the downtime and migration time. The proposed bandwidth strategy is to use only additional bandwidth when it highly impacts downtime and migration time. **Additional bandwidth** can be accessed by any container, not for the entire migration, but only when needed most. **Efficient multiple-service migration** requires planned bandwidth adjustments at specified times to **reduce migration time and downtime.** This makes it easier to make bandwidth available to other containers. The rest of this section describes how this is achieved. In this Chapter, bandwidth refers to the available bandwidth for migration.

The MiGrror model indicates that the stated parameters are more influential based on their position in the formulas. As an illustration, based on equation (5.2), the transfer rate of the initial stage is more critical for migration time than middle migration stages. This is because the initial migration stage consists of M_1, and its value is greater than the middle migration stages since represent the copy of the container at the destination node. Increasing the $r_{1,j}$ on the dominator in the equation, reduces its value and, consequently, migration time. Similarly, for downtime, the transfer rate of the final event is more critical than the transfer rates of other events during the migration process. This is because the latest dirty memory is transferred during the final migration stage. For example, increasing the transfer rate during the final migration stage reduces downtime more than increasing it during the middle migration stages.

Figure 5.2 compares using the original bandwidth and the bandwidth allocation strategy. We assume that each application uses multiple containers. The increased bandwidth strategy is favorable when multiple users require migration due to mobility. Moreover, as depicted in figure 5.2-A, containers can use the entire reserved bandwidth for migration in a parallel manner to achieve improved bandwidth utilization. In a typical edge environment, there are typically multiple users, and each user runs multiple applications, with each application receiving service from one or more containers. Therefore, there are numerous containers on an edge node, and at any given time slot, a number of containers could have started migration (initial migration event, event number 1), be in the midst of migration (migration event number $i, where\ 2 \leq i \leq n$), or have completed the migration and handing off (final migration event, event number s) from the current to the next edge node. This is how this strategy responds to the real world.

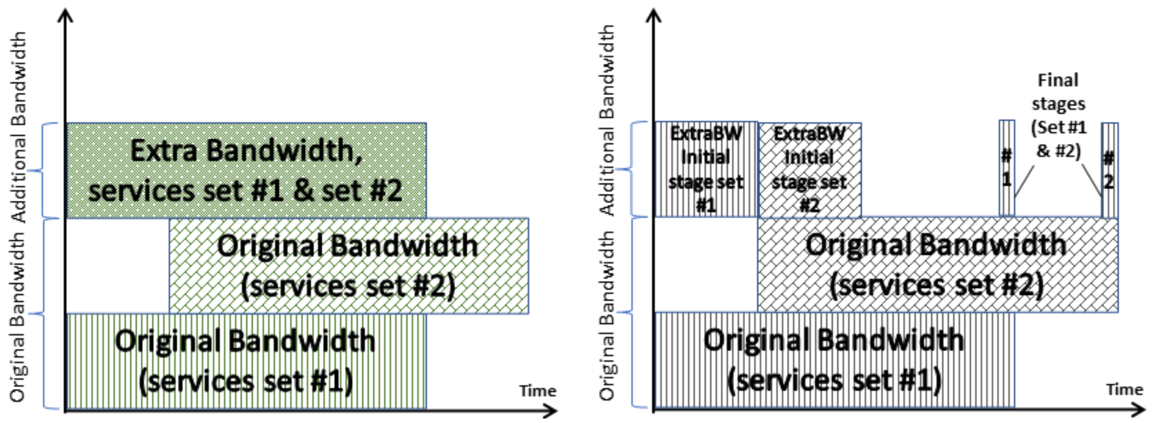
Original Bandwidth
Additional Bandwidth
Extra Bandwidth, services set #1 & set #2
Original Bandwidth (services set #2)
Original Bandwidth (services set #1)
Time
ExtraBW Initial stage set #1
ExtraBW Initial stage set #2
Final stages (Set #1 & #2)
#1
#2

Figure 5.2 The concept of use increased bandwidth during the initial stage of migration. Left (A): Original bandwidth, Right (B): Increased Bandwidth for the initial and final stages.

Figures 5.2-A and 5.2-B illustrate two sets of services that require migration, namely applications and their corresponding containers. The services in set #1 will undergo parallel migration, as will those in set #2. The scenario assumes services in set #1 will begin migration prior to the services in set #2. We use the above concept with container bandwidth assignment modifications, as shown in figure 5.2-B. As depicted in figure 5.2-B, we assign the extra bandwidth only to the initial and final migration stages. As a result, this strategy can reduce the output parameters, namely migration time, only when specified and without using additional bandwidth for the whole migration process.

Therefore, rather than using additional bandwidth throughout the migration process, our bandwidth strategy focusses on using increased bandwidth selectively. Equations (5.1) and (5.2) indicates that additional bandwidth should be used during the **initial** migration event. This event occurs during the first step of MiGrror migration when transferring the container. At $t_{1,j}^{Mir}$, we increase $r_{1,j}$ for any $container_j$. t_2 is related to t_1, t_3 is related to t_2, and so on, until t_n. Equation (5.2) shows that increasing the transfer rate of the initial event will decrease t_1, causing domino-like effects on the rest of the migration timeslots t_1 to t_n. The percentage of improvement varies depending on container size, memory dirtying rate, and transfer rate. As seen in experiments in section 5.5, using this increased bandwidth strategy significantly reduces migration time which is substantial for future 5G and 6G applications. Moreover, with a shorter migration time, resources at the source can be made available to other containers more quickly.

We employ the same concept to reduce downtime: Equation (5.1) of the MiGrror model, increasing bandwidth **only** during the **hand-off**, when the last dirty memory is transferred, directly affects downtime. In fact, during the stop-and-copy phase, we increase the $r_{s,j}$ for any $VM_j/container_j$. The $r_{s,j}$ is in the denominator of $\lambda_{s,j}$, so an increase has a direct impact on downtime. Providing more bandwidth in this short period for a specific container handing off from a source to a destination, will reduce downtime by the factor of the increased bandwidth during hand-off. Given that downtime is a small percentage of the overall migration process, according to our results, providing additional bandwidth during this time is worthwhile to ensure smooth migration for future 5G and 6G applications.

Furthermore, there is an additional benefit to implementing the new bandwidth allocation strategy. As seen with equations (5.2) and (5.3), this strategy decreases migration overhead since these equations depend on $t_{i,j}^{Mir}$ $(1 \leq i \leq n)$, employing the increased bandwidth allocation strategy will reduce migration time. This shorter migration time forces the source to have less time to synchronize containers to the destination, reducing migration overhead. The percentage of improvement varies depending on container size, memory dirtying rate, and transfer rate. Consequently, the migration overhead decreases despite the utilization of additional bandwidth, and the reduced bandwidth is made available for use by other applications.

Investigating the MiGrror model [113] and according to equation (5.2), we discovered that the transfer rate of the initial migration phase significantly impacts migration time and downtime, whereas the middle migration phases have less impact on the given parameters. This is because the data to be transferred is high and this is likely higher than data in other rounds. As a result, rather than increasing total bandwidth for all services, we maintain total bandwidth for all users while increasing transfer rates only in the initial and final phases and decreasing transfer rates in the middle phases. Since multiple containers are servicing various users' applications at any given time, a transfer rate reduction in the middle migration phases will save bandwidth for services that are either in the initial or final migration phases. The most significant point is that the initial and final migration phases account for only a portion of the total migration time. Therefore, reducing the transfer rate of the middle migration phases by $x\%$ will save this bandwidth for the initial and final migration phases by more than $x\%$. The input parameters determine the saved bandwidth; for instance, if half of the containers (which correspond to different users) want to migrate simultaneously, they can have approximately $2x\%$ more bandwidth for the initial and final migration phases. Assume a simple scenario: we have 1000 (megabits per second) total bandwidth for 20 containers, each with 50 Mbps. We can save 100 Mbps with a 10% transfer rate reduction for the middle phases, which means 5 Mbps for each container. If we need to migrate half of the containers simultaneously, we can use 100 Mbps for ten users during the initial and final migration phases, which is a 20% increase. This number can vary, and in section 5.5, we conduct experiments based on various assumptions. In most cases, the performance is better than the traditional bandwidth approach, which assigns 100% bandwidth to all migration phases. In the following section, we demonstrate the effectiveness of this idea by presenting results.

5.5 Testbed Evaluation

Several factors, such as container size, transfer rate, and memory dirtying rate, can impact performance in the migration process. This section will explore how adjusting the transfer rate can affect these parameters. We used experiments for the first part of the analysis. Our results include the average migration time, downtime, and migration overhead from ten separate migration runs.

5.5.1 Testbed Setup

The experiments were carried out using the following setup: four Raspberry Pi 3 B+ (Broadcom BCM2837B0 Cortex-A53 @1.4GHz, 1GB RAM) as IoT devices running various application containers, two Raspberry Pi 4B (Broadcom BCM2711 Cortex-A72 @1.8GHz, 4GB RAM) as edge nodes, and a PC (Intel 12 cores @2.10 GHz, 16.0 GB RAM) as a coordinator edge node that sends requests to other nodes to start migrations and ends them. We used UDP for communication since it transfers messages faster than TCP. To create a heterogeneous container environment, we applied containers of varying given parameter values. Fedora 38 is used as the operating system, with CRIU 3.17 [121] for checkpoint-restore and the Linux *stress* tool to generate memory loads. Furthermore, we use 12 heterogeneous containers (for multiple applications) to migrate from the source to the destination during migration in our experiments. In this section, we divide the results into two parts: one uses the experimental data, and another, synthesized data. Considered parameter values vary and are discussed in detail in the following subsections.

To analyze the data, we selected parameters randomly and calculated the minimum, maximum, median, average, and standard deviations of $r_{i,j}$ and M_j. The transfer rate $(r_{i,j})$ varies between 40 $megabits\ per\ seconds$ $(Mbps)$ and 250 $Mbps$. The median, average, and standard deviation are 115.81, 118.37, and 33.88, respectively. Memory sizes (M_j) range from a minimum of 163.72 $megabytes$ (MB) to a maximum of 1165.29 M. The median, average, and standard deviation are 364.14, 403.92, and 294.56, respectively. Input parameters are randomly selected from the given values. In this section, we use the same values for all experiments rather than the increase/decrease in transfer rates for bandwidth strategy. For example, if $container_1$ has a size of 400 MB, we use the same amount in our experiments.

5.5.2 Experimental Results

This section presents the performance of the MiGrror methods with and without using the proposed strategy based on the given parameters using experiments for the given testbed in this chapter. Figure 5.3 depicts the downtime, migration time, and migration overhead of the MiGrror migration method with and without the increased bandwidth allocation strategy. As shown in the figure, we increase the bandwidth of the last event by a factor of two, five, and ten compared to its original bandwidth. When compared to the initial bandwidth results, the downtime for the MiGrror method that employs the bandwidth strategy is 50% at twofold bandwidth, 20% at fivefold bandwidth, and 10% at tenfold bandwidth. For instance, when the downtime is 8.50 ms with the original bandwidth, it drops to 4.25 ms using double the bandwidth. Furthermore, the downtime decreases to 1.70 ms and 0.85 ms, respectively, when bandwidth is increased by fivefold and tenfold compared to the original bandwidth. We achieved these results while using the additional bandwidth for less than 0.02 percent of the total migration duration.

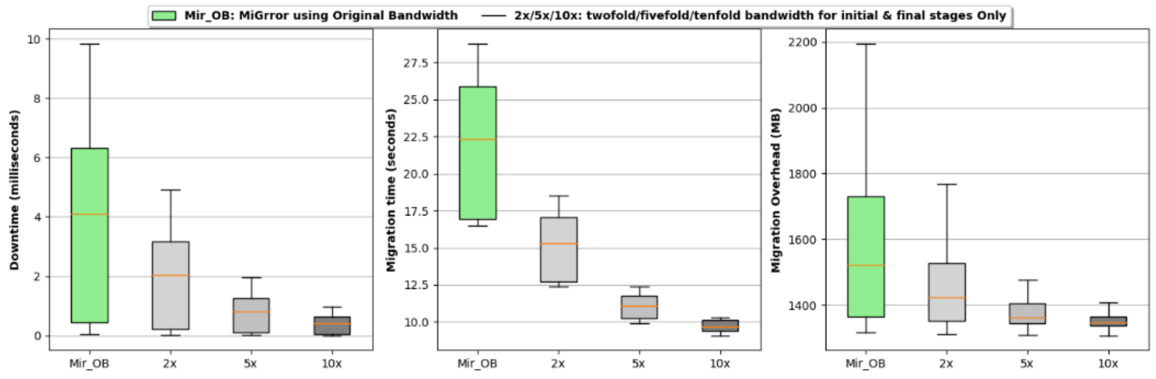

Figure 5.3 Comparison of the MiGrror migration method with and without the use of bandwidth strategy (light green: MiGrror without bandwidth strategy - using original bandwidth, greys: MiGrror using bandwidth strategy; 2x, 5x, and 10x: twofold, fivefold, and tenfold transfer rate only during the first and last migration stages) (left (A): Downtime, middle (B): Migration Time, right (C): Migration Overhead).

Furthermore, as shown in the figure, using the MiGrror technique with the increased bandwidth allocation strategy reduces migration time between 25% and 36%, using a twofold increase in transfer rate only at the first migration event, compared to using the initial bandwidth. In addition, compared to the original bandwidth of the MiGrror technique, the migration time decreases between 36% and 58% using a fivefold increased transfer rate at the first event of migration. Migration time reduction for a tenfold increased transfer rate at the first migration event is between 44 and 63 percent. Because of the shorter migration time, resources at the source can be made more quickly available to other containers, allowing more users and containers to be served with higher performance while using the same available resources, such as memory and processing power—additionally, the shorter migration time results in a reduction in migration overhead.

The decreased migration overhead is a result of fewer dirty memory transfers during the migration, as the migration is shorter when using an increased bandwidth allocation strategy. The source node must transfer fewer dirty memories, resulting in a reduction in migration overhead. For example, compared to the original bandwidth of the MiGrror technique, the migration overhead decreases by 14% using a twofold increased transfer rate at the first event of migration. In addition, compared to the original bandwidth of the MiGrror technique, the migration overhead decreases by 19% using a fivefold increased transfer rate at the first event of migration. Migration overhead reduction for a tenfold increased transfer rate at the first migration event is 21 percent. These results indicate that, despite increasing bandwidth for a period of time during migration, we can **reduce migration overhead**, and this benefit should be considered when employing the **bandwidth allocation strategy**.

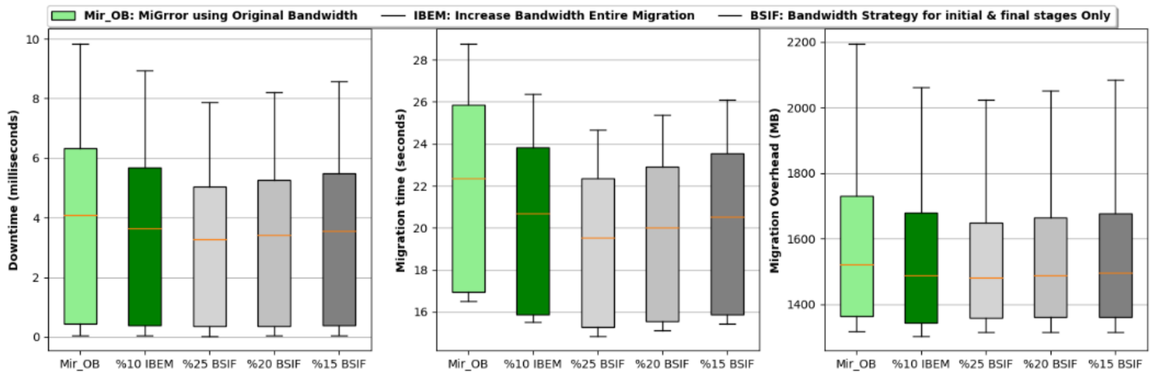

Figure 5.4 Comparison of the MiGrror migration method with and without the use of bandwidth strategy (light green: MiGrror without bandwidth strategy, dark green: increase transfer rate for entire migration by %10, greys: MiGrror using bandwidth strategy; %15, %20, and %25 increase transfer rate only during the first and last migration stages) (left (A): Downtime, middle (B): Migration Time, right (C): Migration Overhead).

Figure 5.4 shows another example of the proposed strategy's strength. The light green bars show the results of MiGrror, while the dark green bars show what happens when 10% more bandwidth is assigned to each container for the entire migration duration. The grey bars represent bandwidth increases of 15%, 20%, and 25% for only the initial and final migration stages. The plots show that bandwidth utilization increased as downtime, migration time, and migration overhead decreased when the proposed strategy was used. For example, when there is a 20% increase in bandwidth for the initial and final stages of migration, 50% of containers require simultaneous migration. Bandwidth utilization increases as the proportion of containers requiring simultaneous migration decreases. This means that the better the performance, the lower the percentage of available bandwidth for the initial and final migration stages.

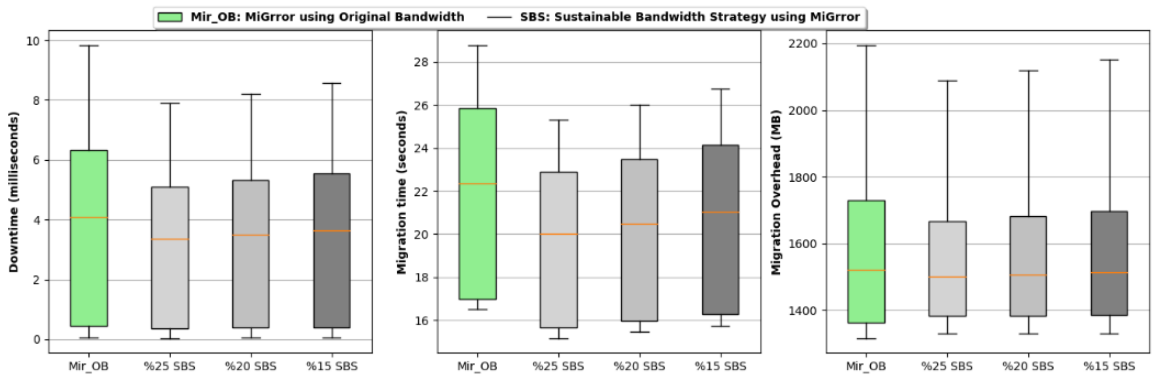

Figure 5.5 Comparison of the MiGrror migration method with and without the use of bandwidth strategy (light green: MiGrror without bandwidth strategy, greys: MiGrror using bandwidth strategy; %15, %20, and %25 increase transfer rate only during the first and last migration stages, and decrease transfer rate during middle migration stages by %10) (left (A): Downtime, middle (B): Migration Time, right (C): Migration Overhead).

Another example of the proposed strategy's strength is shown in figure 5.5. The light green bars represent the MiGrror results. The grey bars represent bandwidth increases of 15%, 20%, and 25% for only the initial and final migration stages, while the transfer rate of the middle migration stages is reduced by 10% to free up bandwidth and is assigned to the initial and final migration stages. Again, the plots show that bandwidth utilization increased when the proposed strategy was used while downtime, migration time, and migration overhead decreased when compared to the results of the original bandwidth.

For example, when there is a 20% increase in bandwidth for the initial and final stages of migration, 50% of containers require simultaneous migration. Bandwidth utilization increases as the proportion of containers requiring simultaneous migration decreases. This means that the better the performance, the lower the percentage of available bandwidth for the initial and final migration stages.

5.5.3 Results using Synthesized Data

This section presents the performance of the proposed strategy based on the given parameters using synthesized parameter values, which include container size, memory dirtying rate, and transfer rate. This section demonstrates the contrast between using the proposed bandwidth strategy compared to the original bandwidth by employing various input parameter values. Further details regarding these values will be provided in this section. Figure 5.6 illustrates how various rates of increased bandwidth affect downtime and migration time when using the MiGrror migration method. The container size is fixed at 200 MB. According to the results, increasing bandwidth rates reduce MiGrror downtime by the factor of increased bandwidth. When the downtime is 10 ms, for instance, doubling the last event's transfer rate (bandwidth) reduces the downtime to 5 ms. The bandwidth strategy has a non-linear effect on MiGrror migration time. For example, when using a $2x$ transfer rate than the original transfer rate during the migration first event, the reduction is 12.57% at a 0.025 $\lambda_{i,j}$ rate, but this drops to a 7.51% reduction at a 0.4 $\lambda_{i,j}$ rate.

These results indicate that a bandwidth strategy can significantly reduce downtime with minimal additional bandwidth consumption. This strategy also reduces migration time, which varies according to the parameters. In addition, with this strategy, the doubled bandwidth of the first migration event lasts 3.66 percent of the entire migration process at a 0.5 $\lambda_{i,j}$ rate and 7.14 percent at a 0.25 $\lambda_{i,j}$ rate. These findings indicate that we require double the bandwidth to achieve the stated results in only 3.66 to 7.14 percent of the total migration time at a 0.025-0.5 $\lambda_{i,j}$ rate. It gets even better when we see a fivefold and tenfold bandwidth reduction of 11.49-16.33 percent and 12.82-17.58 percent, respectively. The first event of fivefold and tenfold increased bandwidth lasts 1.58-3.08 and 0.84-1.65 percent of the entire migration process. The results also indicate that as the bandwidth of the initial migration event increases, the migration time decreases in a nonlinear fashion. For instance, a twofold transfer rate reduces migration time by 12.57 percent at the initial event, while a tenfold transfer rate reduces it by 17.58 percent. Moreover, from a different perspective, the period required to maintain a twofold transfer rate in the twofold case is 7.13 percent of the entire migration process, whereas the tenfold case requires only 1.65 percent of the entire migration process.

The bandwidth strategy also has a non-linear effect on MiGrror migration overhead. For instance, when using a $2x$ transfer rate instead of the original transfer rate during the migration first event, the reduction is 1.61% at a 0.025 $\lambda_{i,j}$ rate, but this increases to an 8.22% reduction at a 0.4 $\lambda_{i,j}$ rate. The results of the increased bandwidth strategy show that the bandwidth usage is reduced compared to when using the original bandwidth. This means even though we increase the bandwidth for a short period, we **decrease the overall bandwidth usage**. This leads to a better bandwidth utilization. Fig. 6 shows that when the migration first event uses a twofold transfer rate rather than the original transfer rate, and the $\lambda_{i,j}$ rate is less than 0.175, the migration overhead is greater than the overhead decrement using the bandwidth strategy. However, for higher $\lambda_{i,j}$ rates, the bandwidth strategy results in lower total migration overhead, taking into account the amount of increased bandwidth during the migration process as well as the amount of reduction by the bandwidth strategy. For example, using a $2x$ transfer rate compared to the original transfer rate during the first migration event reduces the migration overhead by 8.70% at a 0.5 $\lambda_{i,j}$ rate, while the first migration event only lasts 4.4% of the total migration time. This data indicates that, in some cases, despite temporarily increasing the bandwidth during migration, the overall migration overhead is less than when using the original bandwidth. Therefore, this advantage should be considered when employing the bandwidth allocation strategy.

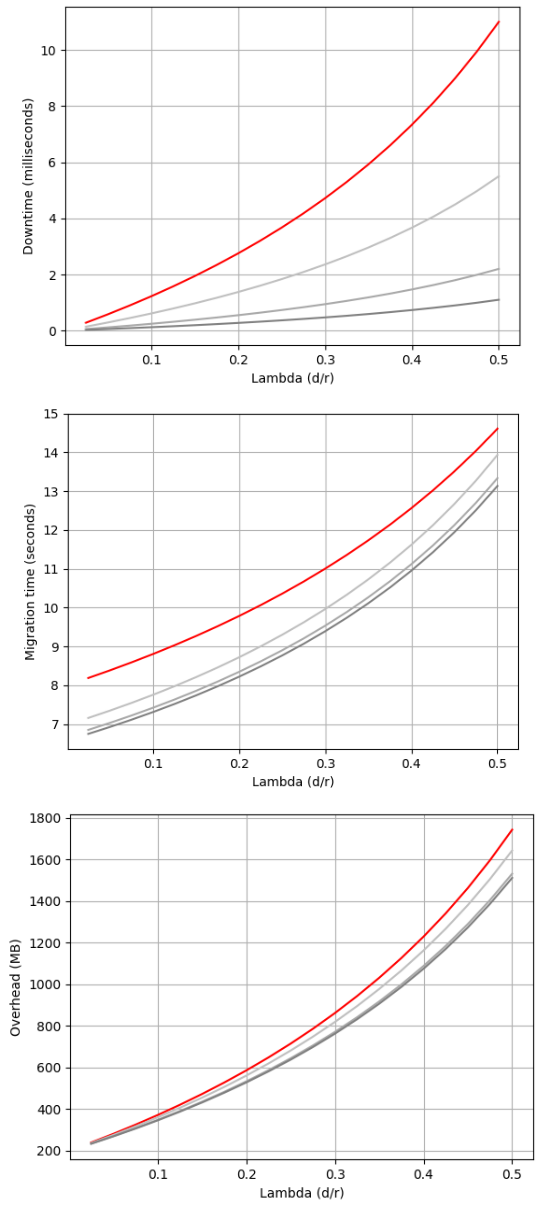

Figure 5.6 Comparison of the MiGrror migration method with and without the use of bandwidth strategy (red: MiGrror, greys: MiGrror using bandwidth strategy; 2x, 5x, and 10x: twofold, fivefold, and tenfold transfer rate only during the first and last migration events) (left (A): downtime, middle (B): Migration time, right (C): Migration Overhead).

This strategy is ideal for latency-sensitive mobile IoT applications and those that require rapid migration, such as users with high-speed mobility. These users require a swift migration from one MEC node to another because the user speed does not permit a slow migration since there are data to transfer before the user leaves the MEC range, and they risk an incomplete migration.

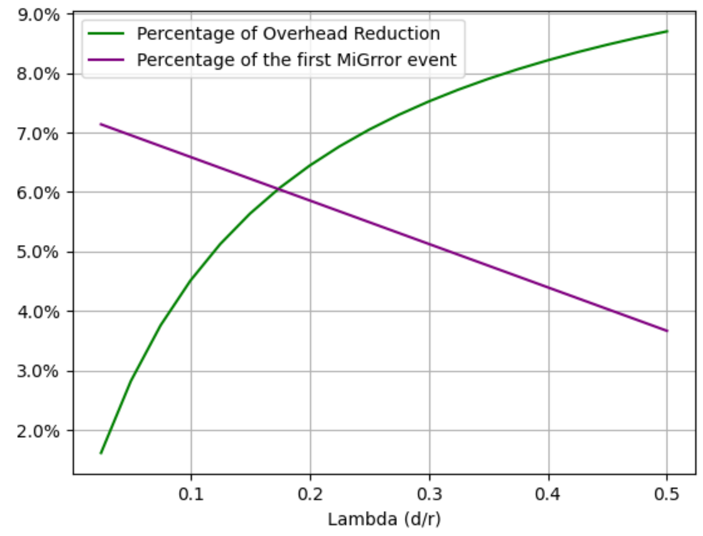

Figure 5.7 Comparison of the MiGrror migration method with the use of bandwidth strategy with twofold transfer rate only during the first migration event (green: Migration Overhead Reduction, purple: Percentage of the first MiGrror event).

Figure 5.7 illustrates how and when we can transfer less data in order to save network traffic. The intersection of two lines to the right of the figure indicates that with higher Lambda, the transferred data is reduced in comparison to the original bandwidth. This diagram gives network designers a good idea of how to design their infrastructure sustainably with less network traffic.

5.6 Summary

This Chapter employs the MiGrror migration method. The nature of the MiGrror migration method introduced in Chapter 3 requires using non-average parameter values in models. We were able to consider realistic values for each parameter using MiGrror since we used non-average parameter values for MiGrror models in Chapter 4, unlike other research that only uses average parameter values. As a result, we were able to develop a novel bandwidth allocation strategy that reduces downtime and migration time while transferring less data than traditional approaches. In addition, we investigate the optimal time to increase the transfer rate for the increased bandwidth allocation strategy during both the initial migration event to reduce migration time and during the final migration event to reduce downtime. **As a result of utilizing the proposed strategy, we reduced downtime and migration time**.

Chapter 6

Conclusions and Future Directions

This chapter concludes the thesis by summarizing the works and principal contributions. It also highlights and discusses future research directions to improve edge computing concepts.

6.1 Summary of Contributions

The Internet of Things (IoT) has become a vital component of everyday life due to the perpetual evolution of software and hardware technologies and the widespread availability of the Internet. The Internet of Things involves a wide range of application scenarios with various resource requirements, including computationally intensive or latency-sensitive ones. As a distributed computing paradigm, edge computing involves including remote servers in addition to those located close to IoT devices. Thus, particularly for mobile users, it provides heterogeneous resources to support a wide variety of applications. It has already attracted considerable interest from both industry and academia. However, with the advent of modern applications, the demand for latency-sensitive and real-time responses is becoming increasingly unavoidable. The mobility of IoT devices increases this demand. According to existing research, current migration methods cannot satisfy the latency requirements of mission-critical applications. Edge computing migration requires additional research to satisfy the demand for rapid response, availability, and continuity of services for users. Since they are fundamentally designed for cloud applications, these methods cannot meet the latency and availability requirements for mobile IoT devices. This thesis focused on modifying and developing new migration methods, models, and strategies to overcome the stated limitations as an alternative to current migration methods in highly distributed and dynamic edge computing environments.

Chapter 1 provides a high-level overview of edge computing migration. Subsequently, critical challenges associated with edge computing migration are highlighted and explained. This chapter also addressed the research questions addressed in this thesis.

Chapter 2 provides the existing migration techniques for IoT applications in edge computing from different perspectives, including traditional migration techniques, how downtime and migration time were reduced in other research, single and multiple migrations, and migration models. Then, a survey of the recent literature is provided, considering each perspective. Finally, the limitations and research gaps of each perspective are described.

Chapter 3 presents a novel migration technique that reduces downtime and migration time in edge computing environments. Our findings show that MiGrror performs well when used to support mobile IoT applications, particularly when latency is critical. These results enable delay-sensitive applications that require low latency to run smoothly during migration. First, a comparison of MiGrror and pre-copy methods is presented. Second, a MiGrror workflow is proposed. Following that, two algorithms are presented, one for the source-side edge node and one for the destination-side edge node. The results are then discussed.

Chapter 4 analyzes the performance of the MiGrror migration method and the pre-copy live migration method when migrating heterogeneous multiple VMs/containers. In this Chapter, we use non-average parameter values as well as traditional average parameter values for downtime, migration time, and migration overhead. This Chapter demonstrates how MiGrror can improve user service continuity and availability. This is accomplished using non-average values for various parameters, allowing us to better understand the migration process and produce more accurate results. This Chapter shows how using only average parameter values in migration can result in inaccurate results. Initially, the pre-copy mathematical model was examined. The non-average MiGrror downtime, migration time, and migration overhead models were then presented. Finally, we evaluated the performance of the models utilizing both a dataset and synthesized data.

Chapter 5 puts forward a novel bandwidth allocation strategy for migrating multiple services in edge computing environments. The non-average MiGrror migration model provides a better understanding of the migration process, allowing us to adjust the bandwidth based on migration timing. In this Chapter, we reduced downtime, migration time, and migration overhead by better utilizing the bandwidth. Initially, we exemplified the problem. We subsequently presented the bandwidth allocation strategy for the purpose of migrating multiple services. Finally, we conducted experiments and utilized synthesized data to illustrate the efficacy of the strategy.

The preceding chapters share several migration techniques, models, and strategies in edge computing environments, which is a timely contribution to the state-of-the-art.

6.2 Future Research Directions

This thesis addressed several challenges and limitations associated with the migration of IoT applications in edge computing environments. However, edge computing paradigms can be improved further by addressing several key issues that require additional research. Figure 6.1 provides an overview of the future directions discussed in this section.

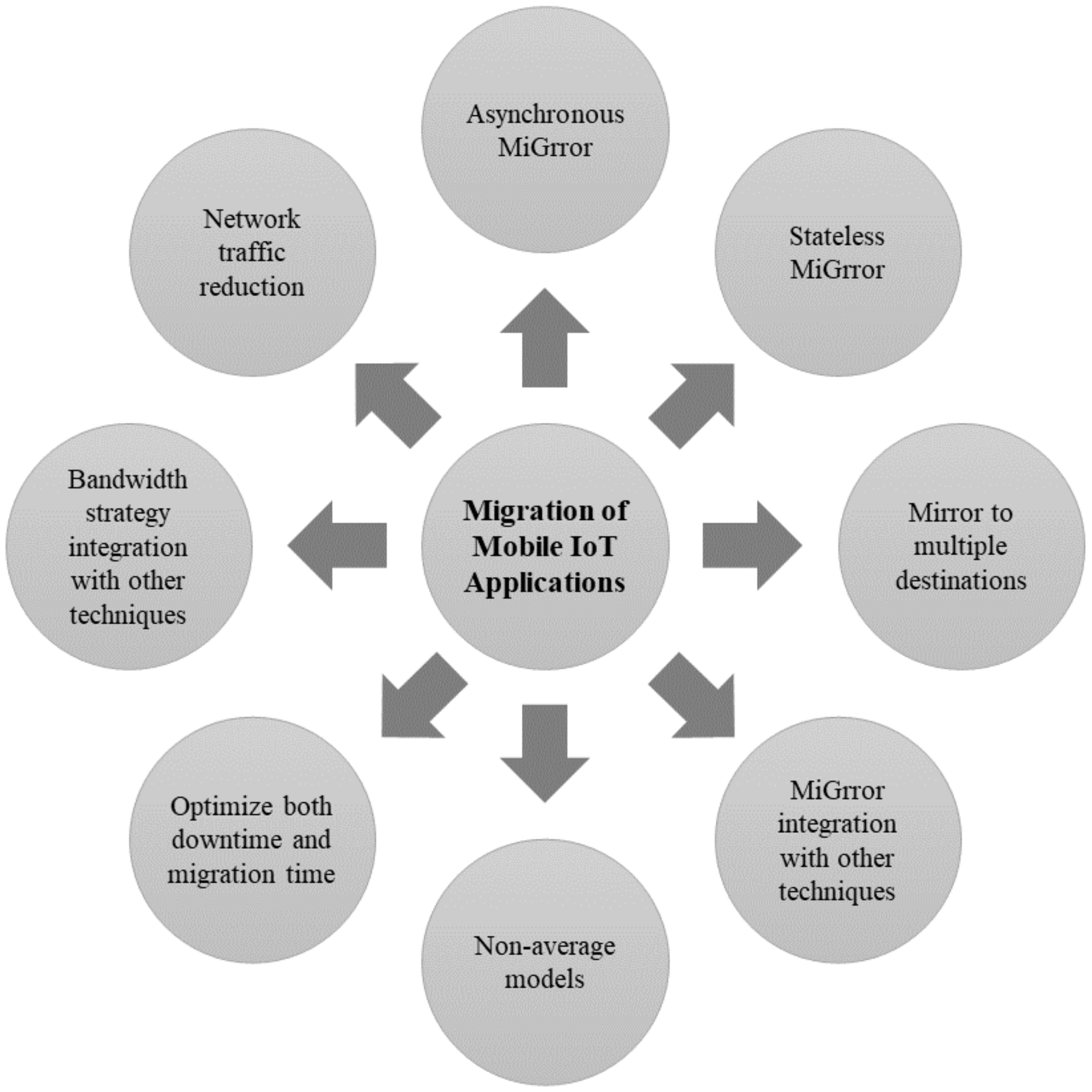
Asynchronous MiGrror
Stateless MiGrror
Mirror to multiple destinations
MiGrror integration with other techniques
Non-average models
Optimize both downtime and migration time
Bandwidth strategy integration with other techniques
Network traffic reduction
Migration of Mobile IoT Applications

Figure 6.1 Summary of future research directions.

6.2.1 Asynchronous MiGrror – Generalizing Migration Methods

MiGrror migration method rapidly synchronizes the source and destination edge nodes when dirty memory becomes available. Consequently, the MiGrror method improves service availability. Meanwhile, the pre-copy synchronizes the source and destination edge nodes, typically 10-30 times during each migration procedure. As a result, MiGrror is very fast for some applications, e.g., backup/restore and online banking applications, that do not require real-time or quick response time, and pre-copy may need to be faster. Asynchronous synchronization (Async MiGrror) could be developed for these situations based on various migration parameters such as time periods, amount of dirty memory generated, data transfer amount in each data transmission, and other factors such as distance between the device and the edge node. It could also be a combination of several parameters that determine when it is appropriate for a scenario to transmit data during the migration procedure. The use of various parameters to decide when to transfer data during migration can be achieved by generalizing both MiGrror and pre-copy methods that can be customized to meet the needs of the service. These requirements include not only the "time periods" parameter used in the pre-copy but also any parameter that affects performance and available resources.

6.2.2 Stateless MiGrror

A MiGrror extension could be used in stateless migration. Since stateless containers do not need to maintain states, they can be replicated by cloning or mirroring from the source node or cloud to the potential destination nodes without requiring migration. However, some data may still be attached to containers, which those containers require in order to preserve the user's data.

6.2.3 Mirror to Multiple Destinations

Using MiGrror and mirroring from a source to multiple potential destination nodes should also be advantageous when the source is unsure of the succeeding possible node to migrate to or when the application requires collaboration with more than one node to achieve higher performance. This is also advantageous when multiple edge nodes use the same container.

6.2.4 MiGrror Integration with Other Techniques

Since the MiGrror migration method is an alternative to the pre-copy, integrating MiGrror with existing ML and non-ML techniques can result in novel strategies to reduce downtime, delay, and migration time even further. As an example, the MiGrror model can be **integrated** with other migration approaches, such as that described in [29], using compression for the migration model to improve accuracy. In addition, the new migration model can be integrated with ML-based techniques to provide a detailed view of expected outputs. In another example, the new MiGrror model with non-average parameter values can improve output precision and reduce migration times for multiple containers of the research described in [14].

6.2.5 Non-average Models

This thesis would lead to future research exploring models using non-average parameter values in order to better understand and optimize migration parameters using the non-average parameter values. Most current migration models assume average inputs, which can make models less effective in real-world scenarios. Furthermore, designing and working with non-average models is more challenging than working with current models that use average inputs and require additional research.

6.2.6 Optimize both Downtime and Migration Time for Multi-containerized Applications

Current research is focused only on optimizing migration times for multiple services. The proposed MiGrror model can be used to dynamically optimize both downtime and migration times in multi-containerized (and multi-VM) environments using non-average parameter values, unlike previous research that only focused on optimizing migration times using average parameter values. This optimization could be accomplished by dynamically modifying the allocated transfer rate for multiple services.

6.2.7 Bandwidth Allocation Strategy Integration with Other Techniques

The proposed bandwidth allocation strategy described in this thesis can also be incorporated into other approaches, such as machine learning techniques, to achieve even higher performance than current techniques.

6.2.8 Network Traffic Reduction

It could be investigated whether using the proposed bandwidth strategy can use less bandwidth during the migration and achieve the same results as when the original bandwidth is used without employing a bandwidth strategy throughout the migration.

6.2.9 Load Balancing and Cloud Computing

The MiGrror method, models, and bandwidth strategy could be investigated for use in load balancing in edge networks and cloud servers. Current migration techniques for load balancing require supporting applications with fast response times. As a result, in an edge environment, some applications may require service migration from one edge node to another to balance the load, with or without considering mobility. Those applications that require a quick response can benefit from the MiGrror migration method, models, and bandwidth strategy.

6.2.10 Energy Consumption

It could be investigated whether using the proposed migration method, model, and bandwidth strategy can use less or more energy during the migration, and the results could be compared with the pre-copy migration method.

6.2.11 Inter-Vendors Migration

It might be examined while shifting between vendor partners. Several challenges, such as privacy and security, must be addressed when migrating between different edge computing environments. Furthermore, both parties should be aware of the protocols.

6.3 Final Remarks

The edge computing paradigm has evolved into the digital world's backbone, allowing IoT-driven solutions to be deployed for various use cases, such as healthcare, transportation, science, and entertainment, just to mention a few. To effectively utilize the potential of edge computing, efficient and dynamic migration methods of mobile IoT applications are vital, affecting user QoE. In this thesis, we investigated how to migrate services efficiently for the smooth execution of heterogeneous IoT applications across distributed edge servers. The migration method, algorithms, mathematical models, and strategy proposed in this thesis reduce downtime and migration time of mobile IoT applications while also improving user QoE. The research presented in this thesis on the migration of mobile IoT applications will enable edge service providers to perform migration successfully and efficiently at scale in highly heterogeneous, dynamic, and complex edge computing environments. Furthermore, the findings of this research can help to advance the innovations and developments of IoT and edge computing systems.

Bibliography

[1] R. Singh, R. Sukapuram, and S. Chakraborty, ‘A survey of mobility-aware Multi-access Edge Computing: Challenges, use cases and future directions’, *Ad Hoc Networks*, vol. 140, p. 103044, Mar. 2023, doi: 10.1016/j.adhoc.2022.103044.

[2] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, and F. Longo, ‘Design and evaluation of a fog platform supporting device mobility through container migration’, *Pervasive Mob Comput*, vol. 74, p. 101415, 2021, doi: https://doi.org/10.1016/j.pmcj.2021.101415.

[3] S. M. Salman, T. A. Sitompul, A. V. Papadopoulos, and T. Nolte, ‘Fog Computing for Augmented Reality: Trends, Challenges and Opportunities’, in *2020 IEEE International Conference on Fog Computing (ICFC)*, IEEE, Apr. 2020, pp. 56–63. doi: 10.1109/ICFC49376.2020.00017.

[4] K. Rao, G. Coviello, W.-P. Hsiung, and S. Chakradhar, ‘ECO: Edge-Cloud Optimization of 5G applications’, in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, IEEE, May 2021, pp. 649–659. doi: 10.1109/CCGrid51090.2021.00078.

[5] O. Oleghe, ‘Container Placement and Migration in Edge Computing: Concept and Scheduling Models’, *IEEE Access*, vol. 9, pp. 68028–68043, 2021, doi: 10.1109/ACCESS.2021.3077550.

[6] S. Wang, J. Xu, N. Zhang, and Y. Liu, ‘A Survey on Service Migration in Mobile Edge Computing’, *IEEE Access*, vol. 6, pp. 23511–23528, 2018, doi: 10.1109/ACCESS.2018.2828102.

[7] S. Kekki *et al.*, ‘MEC in 5G networks’, *European Telecommunications Standards Institute (ETSI)*, no. 28. pp. 1–28, Jun. 2018. Accessed: Nov. 26, 2022. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp28_mec_in_5G_FINAL.pdf

[8] F. Giust *et al.*, ‘MEC Deployments in 4G and Evolution Towards 5G’, *European Telecommunications Standards Institute (ETSI)*, no. 24. pp. 1–24, Feb. 2018. Accessed: Nov. 26, 2022. [Online]. Available: https://www.etsi.org/images/files/etsiwhitepapers/etsi_wp24_mec_deployment_in_4g_5g_final.pdf

[9] D. Sabella, ‘ETSI MEC Standard Explained – Part II’, IEEE Communications Society. Accessed: Nov. 26, 2022. [Online]. Available:

https://techblog.comsoc.org/2021/12/18/etsi-mec-standard-explained-part-ii/

[10] D. Sabella and A. J. Weissberger, 'Multi-access Edge Computing (MEC) Market, Applications and ETSI MEC Standard-Part I', IEEE Communications Society. Accessed: Nov. 26, 2022. [Online]. Available: https://techblog.comsoc.org/2021/12/15/multi-access-edge-computing-mec-market-applications-and-technology-part-i/

[11] N. Verba, K.-M. Chao, A. James, D. Goldsmith, X. Fei, and S.-D. Stan, 'Platform as a service gateway for the Fog of Things', *Advanced Engineering Informatics*, vol. 33, pp. 243–257, Aug. 2017, doi: 10.1016/j.aei.2016.11.003.

[12] M. Goudarzi, M. Palaniswami, and R. Buyya, 'Scheduling IoT Applications in Edge and Fog Computing Environments: A Taxonomy and Future Directions', *ACM Comput Surv*, vol. 55, no. 7, pp. 1–41, Jul. 2023, doi: 10.1145/3544836.

[13] W. Zhang, L. Chen, J. Luo, and J. Liu, 'A two-stage container management in the cloud for optimizing the load balancing and migration cost', *Future Generation Computer Systems*, vol. 135, pp. 303–314, Oct. 2022, doi: 10.1016/j.future.2022.05.002.

[14] G. Singh and A. K. Singh, 'Optimizing multi-VM migration by allocating transfer and compression rate using Geometric Programming', *Simul Model Pract Theory*, vol. 106, p. 102201, Jan. 2021, doi: 10.1016/j.simpat.2020.102201.

[15] A. Satpathy, M. N. Sahoo, A. Mishra, B. Majhi, J. J. P. C. Rodrigues, and S. Bakshi, 'A Service Sustainable Live Migration Strategy for Multiple Virtual Machines in Cloud Data Centers', *Big Data Research*, vol. 25, p. 100213, Jul. 2021, doi: 10.1016/j.bdr.2021.100213.

[16] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, 'Migration Modeling and Learning Algorithms for Containers in Fog Computing', *IEEE Trans Serv Comput*, vol. 12, no. 5, pp. 712–725, Sep. 2019, doi: 10.1109/TSC.2018.2827070.

[17] T. He, A. N. Toosi, and R. Buyya, 'Performance evaluation of live virtual machine migration in SDN-enabled cloud data centers', *J Parallel Distrib Comput*, vol. 131, pp. 55–68, Sep. 2019, doi: 10.1016/j.jpdc.2019.04.014.

[18] S. Maheshwari, S. Choudhury, I. Seskar, and D. Raychaudhuri, 'Traffic-Aware Dynamic Container Migration for Real-Time Support in Mobile Edge Clouds', in *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, IEEE, Dec. 2018, pp. 1–6. doi: 10.1109/ANTS.2018.8710163.

[19] A. Satpathy, S. K. Addya, A. K. Turuk, B. Majhi, and G. Sahoo, ‘Crow search based virtual machine placement strategy in cloud data centers with live migration’, *Computers & Electrical Engineering*, vol. 69, pp. 334–350, Jul. 2018, doi: 10.1016/j.compeleceng.2017.12.032.

[20] M. E. Elsaid, A. Shawish, and C. Meinel, ‘Enhanced Cost Analysis of Multiple Virtual Machines Live Migration in VMware Environments’, in *2018 IEEE 8th International Symposium on Cloud and Service Computing (SC2)*, IEEE, Nov. 2018, pp. 16–23. doi: 10.1109/SC2.2018.00010.

[21] G. Sun, D. Liao, V. Anand, D. Zhao, and H. Yu, ‘A new technique for efficient live migration of multiple virtual machines’, *Future Generation Computer Systems*, vol. 55, pp. 74–86, Feb. 2016, doi: 10.1016/j.future.2015.09.005.

[22] H. Liu and B. He, ‘VMbuddies: Coordinating Live Migration of Multi-Tier Applications in Cloud Environments’, *IEEE Transactions on Parallel and Distributed Systems*, vol. 26, no. 4, pp. 1192–1205, Apr. 2015, doi: 10.1109/TPDS.2014.2316152.

[23] M. Forsman, A. Glad, L. Lundberg, and D. Ilie, ‘Algorithms for automated live migration of virtual machines’, *Journal of Systems and Software*, vol. 101, pp. 110–126, Mar. 2015, doi: 10.1016/j.jss.2014.11.044.

[24] W. Cerroni, ‘Network performance of multiple virtual machine live migration in cloud federations’, *Journal of Internet Services and Applications*, vol. 6, no. 6, pp. 1–20, Dec. 2015, doi: 10.1186/s13174-015-0020-x.

[25] M. Schneider, J. Rambach, and D. Stricker, ‘Augmented reality based on edge computing using the example of remote live support’, in *2017 IEEE International Conference on Industrial Technology (ICIT)*, IEEE, Mar. 2017, pp. 1277–1282. doi: 10.1109/ICIT.2017.7915547.

[26] P. Habibi, M. Farhoudi, S. Kazemian, S. Khorsandi, and A. Leon-Garcia, ‘Fog Computing: A Comprehensive Architectural Survey’, *IEEE access*, vol. 8, pp. 69105–69133, 2020, doi: 10.1109/ACCESS.2020.2983253.

[27] C. Puliafito *et al.*, ‘MobFogSim: Simulation of mobility and migration for fog computing’, *Simul Model Pract Theory*, vol. 101, pp. 1–25, 2020, doi: https://doi.org/10.1016/j.simpat.2019.102062.

[28] A. Rezazadeh, D. Abednezhad, and H. Lutfiyya, ‘MiGrror: Mitigating Downtime in Mobile Edge Computing, An Extension to Live Migration’, *Procedia Comput Sci*, vol. 203, pp. 41–50, 2022, doi: 10.1016/j.procs.2022.07.008.

[29] L. Ma, S. Yi, N. Carter, and Q. Li, ‘Efficient Live Migration of Edge Services Leveraging Container Layered Storage’, *IEEE Trans Mob*

Comput, vol. 18, no. 9, pp. 2020–2033, Sep. 2019, doi: 10.1109/TMC.2018.2871842.

[30] R. Das and S. Sidhanta, 'LIMOCE: Live Migration of Containers in the Edge', in *2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, IEEE, May 2021, pp. 606–609. doi: 10.1109/CCGrid51090.2021.00070.

[31] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, and H. Flinck, 'Fast Service Migration in 5G Trends and Scenarios', *IEEE Netw*, vol. 34, no. 2, pp. 92–98, 2020, doi: 10.1109/MNET.001.1800289.

[32] M. Pomalo, V. T. Le, N. El Ioini, C. Pahl, and H. R. Barzegar, 'Service Migration in Multi-domain Cellular Networks based on Machine Learning Approaches', in *2020 7th International Conference on Internet of Things: Systems, Management and Security (IOTSMS)*, IEEE, Dec. 2020, pp. 1–8. doi: 10.1109/IOTSMS52051.2020.9340223.

[33] Z. Zhou, X. Li, X. Wang, Z. Liang, G. Sun, and G. Luo, 'Hardware-assisted Service Live Migration in Resource-limited Edge Computing Systems', in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, IEEE, Jul. 2020, pp. 1–6. doi: 10.1109/DAC18072.2020.9218677.

[34] R. Yang, H. He, and W. Zhang, 'Multitier Service Migration Framework Based on Mobility Prediction in Mobile Edge Computing', *Wirel Commun Mob Comput*, vol. 2021, pp. 1–13, 2021, doi: 10.1155/2021/6638730.

[35] A. Machen, S. Wang, K. K. Leung, B. J. Ko, and T. Salonidis, 'Live Service Migration in Mobile Edge Clouds', *IEEE Wirel Commun*, vol. 25, no. 1, pp. 140–147, Feb. 2018, doi: 10.1109/MWC.2017.1700011.

[36] J. Jeong, J. Ha, and M. Kim, 'ReSeT: Reducing the Service Disruption Time of Follow Me Edges over Wide Area Networks', in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, IEEE, Feb. 2019, pp. 159–166. doi: 10.1109/ICIN.2019.8685918.

[37] M. A. Altahat, A. Agarwal, N. Goel, and M. Zaman, 'Analysis and Comparison of Live Virtual Machine Migration Methods', in *2018 IEEE 6th International Conference on Future Internet of Things and Cloud (FiCloud)*, IEEE, Aug. 2018, pp. 251–258. doi: 10.1109/FiCloud.2018.00044.

[38] M. A. Altahat, A. Agarwal, N. Goel, and J. Kozlowski, 'Dynamic Hybrid-copy Live Virtual Machine Migration: Analysis and Comparison', *Procedia Comput Sci*, vol. 171, pp. 1459–1468, 2020, doi: 10.1016/j.procs.2020.04.156.

[39] M. A. Altahat, A. Agarwal, N. Goel, and M. Zaman, 'Neural Network Based Regression Model for Virtual Machines Migration Method

Selection', in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, IEEE, Jun. 2021, pp. 1–6. doi: 10.1109/ICCWorkshops50388.2021.9473749.

[40] Z. Wang, D. Sun, G. Xue, S. Qian, G. Li, and M. Li, 'Ada-Things: An adaptive virtual machine monitoring and migration strategy for internet of things applications', *J Parallel Distrib Comput*, vol. 132, pp. 164–176, Oct. 2019, doi: 10.1016/j.jpdc.2018.06.009.

[41] E. Baccarelli, M. Scarpiniti, and A. Momenzadeh, 'Fog-Supported Delay-Constrained Energy-Saving Live Migration of VMs Over MultiPath TCP/IP 5G Connections', *IEEE Access*, vol. 6, pp. 42327–42354, 2018, doi: 10.1109/ACCESS.2018.2860249.

[42] U. Mandal, P. Chowdhury, M. Tornatore, C. U. Martel, and B. Mukherjee, 'Bandwidth Provisioning for Virtual Machine Migration in Cloud: Strategy and Application', *IEEE Transactions on Cloud Computing*, vol. 6, no. 4, pp. 967–976, Oct. 2018, doi: 10.1109/TCC.2016.2545673.

[43] C.-H. Hong and B. Varghese, 'Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms', *ACM Comput Surv*, vol. 52, no. 5, pp. 1–37, Sep. 2020, doi: 10.1145/3326066.

[44] J. Zhang, B. Chen, Y. Zhao, X. Cheng, and F. Hu, 'Data Security and Privacy-Preserving in Edge Computing Paradigm: Survey and Open Issues', *IEEE Access*, vol. 6, pp. 18209–18237, 2018, doi: 10.1109/ACCESS.2018.2820162.

[45] H. Lin, S. Zeadally, Z. Chen, H. Labiod, and L. Wang, 'A survey on computation offloading modeling for edge computing', *Journal of Network and Computer Applications*, vol. 169, p. 102781, 2020, doi: https://doi.org/10.1016/j.jnca.2020.102781.

[46] D. Liu, Z. Yan, W. Ding, and M. Atiquzzaman, 'A Survey on Secure Data Analytics in Edge Computing', *IEEE Internet Things J*, vol. 6, no. 3, pp. 4946–4967, 2019, doi: 10.1109/JIOT.2019.2897619.

[47] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, 'Vehicular Edge Computing and Networking: A Survey', *Mobile Networks and Applications*, vol. 26, no. 3, pp. 1145–1168, 2021, doi: 10.1007/s11036-020-01624-1.

[48] L. U. Khan, I. Yaqoob, N. H. Tran, S. M. A. Kazmi, T. N. Dang, and C. S. Hong, 'Edge-Computing-Enabled Smart Cities: A Comprehensive Survey', *IEEE Internet Things J*, vol. 7, no. 10, pp. 10200–10232, 2020, doi: 10.1109/JIOT.2020.2987070.

[49] P. Porambage, J. Okwuibe, M. Liyanage, M. Ylianttila, and T. Taleb, 'Survey on Multi-Access Edge Computing for Internet of Things

Realization', *IEEE Communications Surveys & Tutorials*, vol. 20, no. 4, pp. 2961–2991, 2018, doi: 10.1109/COMST.2018.2849509.

[50] R. Yang, F. R. Yu, P. Si, Z. Yang, and Y. Zhang, 'Integrated Blockchain and Edge Computing Systems: A Survey, Some Research Issues and Challenges', *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1508–1532, 2019, doi: 10.1109/COMST.2019.2894727.

[51] Q. Pham *et al.*, 'A Survey of Multi-Access Edge Computing in 5G and Beyond: Fundamentals, Technology Integration, and State-of-the-Art', *IEEE Access*, vol. 8, pp. 116974–117017, 2020, doi: 10.1109/ACCESS.2020.3001277.

[52] Y. Siriwardhana, P. Porambage, M. Liyanage, and M. Ylianttila, 'A Survey on Mobile Augmented Reality With 5G Mobile Edge Computing: Architectures, Applications, and Technical Aspects', *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1160–1192, 2021, doi: 10.1109/COMST.2021.3061981.

[53] A. Yousefpour *et al.*, 'All one needs to know about fog computing and related edge computing paradigms: A complete survey', *Journal of Systems Architecture*, vol. 98, pp. 289–330, 2019, doi: https://doi.org/10.1016/j.sysarc.2019.02.009.

[54] H. Elazhary, 'Internet of Things (IoT), mobile cloud, cloudlet, mobile IoT, IoT cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions', *Journal of Network and Computer Applications*, vol. 128, pp. 105–140, 2019, doi: https://doi.org/10.1016/j.jnca.2018.10.021.

[55] P. Zhang, M. Zhou, and G. Fortino, 'Security and trust issues in Fog computing: A survey', *Future Generation Computer Systems*, vol. 88, pp. 16–27, 2018, doi: https://doi.org/10.1016/j.future.2018.05.008.

[56] C. Clark *et al.*, 'Live Migration of Virtual Machines', in *Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation - Volume 2*, in NSDI'05. USA: USENIX Association, 2005, pp. 273–286.

[57] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, 'Fog computing and its role in the internet of things'. Association for Computing Machinery, pp. 13–16, 2012. doi: 10.1145/2342509.2342513.

[58] ETSI GR MEC 031 V2.1.1 (2020-10), 'Multi-access Edge Computing (MEC) MEC 5G Integration', *European Telecommunications Standards Institute (ETSI)*. pp. 1–47, 2020. Accessed: Nov. 26, 2022. [Online]. Available: https://www.etsi.org/deliver/etsi_gr/MEC/001_099/031/02.01.01_60/gr_MEC031v020101p.pdf

[59] I. Martinez, A. S. Hafid, and A. Jarray, ‘Design, Resource Management and Evaluation of Fog Computing Systems: A Survey’, *IEEE Internet Things J*, pp. 2494–2516, 2020, doi: 10.1109/JIOT.2020.3022699.

[60] P. S. Junior, D. Miorandi, and G. Pierre, ‘Stateful Container Migration in Geo-Distributed Environments’, in *2020 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, Dec. 2020, pp. 49–56. doi: 10.1109/CloudCom49646.2020.00005.

[61] M. Bonanni, F. Chiti, and R. Fantacci, ‘Mobile Mist Computing for the Internet of Vehicles’, *Internet Technology Letters*, vol. 3, no. 6, pp. 1–6, 2020, doi: https://doi.org/10.1002/itl2.176.

[62] C. Puliafito, C. Vallati, E. Mingozzi, G. Merlino, F. Longo, and A. Puliafito, ‘Container Migration in the Fog: A Performance Evaluation’, *Sensors*, vol. 19, no. 7, pp. 1–22, 2019, [Online]. Available: https://www.mdpi.com/1424-8220/19/7/1488

[63] L. Conforti, A. Virdis, C. Puliafito, and E. Mingozzi, ‘Extending the QUIC Protocol to Support Live Container Migration at the Edge’, in *2021 IEEE 22nd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, IEEE, Jun. 2021, pp. 61–70. doi: 10.1109/WoWMoM51794.2021.00019.

[64] U. Deshpande, D. Chan, T.-Y. Guh, J. Edouard, K. Gopalan, and N. Bila, ‘Agile Live Migration of Virtual Machines’, in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, IEEE, May 2016, pp. 1061–1070. doi: 10.1109/IPDPS.2016.120.

[65] A. A. Majeed, P. Kilpatrick, I. Spence, and B. Varghese, ‘Modelling Fog Offloading Performance’, in *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, IEEE, May 2020, pp. 29–38. doi: 10.1109/ICFEC50348.2020.00011.

[66] M. V. Ngo, T. Luo, H. T. Hoang, and T. Q. S. Ouek, ‘Coordinated Container Migration and Base Station Handover in Mobile Edge Computing’, in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, IEEE, Dec. 2020, pp. 1–6. doi: 10.1109/GLOBECOM42002.2020.9322368.

[67] L. Ma, S. Yi, and Q. Li, ‘Efficient service handoff across edge servers via docker container migration’. Association for Computing Machinery, p. Article 11, 2017. doi: 10.1145/3132211.3134460.

[68] I. Farris, T. Taleb, H. Flinck, and A. Iera, ‘Providing ultra-short latency to user-centric 5G applications at the mobile network edge’, *Transactions on Emerging Telecommunications Technologies*, vol. 29, no. 4, pp. 1–14, 2018, doi: https://doi.org/10.1002/ett.3169.

[69] C. Puliafito, E. Mingozzi, C. Vallati, F. Longo, and G. Merlino, 'Virtualization and Migration at the Network Edge: An Overview', in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, IEEE, Jun. 2018, pp. 368–374. doi: 10.1109/SMARTCOMP.2018.00031.

[70] R. de J. Martins, C. B. Both, J. A. Wickboldt, and L. Z. Granville, 'Virtual Network Functions Migration Cost: from Identification to Prediction', *Computer Networks*, vol. 181, pp. 1–16, 2020, doi: https://doi.org/10.1016/j.comnet.2020.107429.

[71] M. Terneborg, J. K. Ronnberg, and O. Schelen, 'Application Agnostic Container Migration and Failover', in *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, IEEE, Oct. 2021, pp. 565–572. doi: 10.1109/LCN52139.2021.9525029.

[72] M. R. Hines, U. Deshpande, and K. Gopalan, 'Post-copy live migration of virtual machines', *SIGOPS Oper. Syst. Rev.*, vol. 43, no. 3, pp. 14–26, 2009, doi: 10.1145/1618525.1618528.

[73] D. Fernando, J. Terner, K. Gopalan, and P. Yang, 'Live Migration Ate My VM: Recovering a Virtual Machine after Failure of Post-Copy Live Migration', in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, IEEE, Apr. 2019, pp. 343–351. doi: 10.1109/INFOCOM.2019.8737452.

[74] M. Gundall, J. Stegmann, C. Huber, and H. D. Schotten, 'Towards Organic 6G Networks: Virtualization and Live Migration of Core Network Functions', in *Mobile Communication - Technologies and Applications; 25th ITG-Symposium*, 2021, pp. 1–6. Accessed: Feb. 22, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9657178

[75] U. Deshpande and K. Keahey, 'Traffic-sensitive Live Migration of Virtual Machines', *Future Generation Computer Systems*, vol. 72, pp. 118–128, 2017, doi: https://doi.org/10.1016/j.future.2016.05.003.

[76] S. Sahni and V. Varma, 'A Hybrid Approach to Live Migration of Virtual Machines', in *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, IEEE, Oct. 2012, pp. 1–5. doi: 10.1109/CCEM.2012.6354587.

[77] F. Zhang, G. Liu, X. Fu, and R. Yahyapour, 'A Survey on Virtual Machine Migration: Challenges, Techniques, and Open Issues', *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1206–1243, 2018, doi: 10.1109/COMST.2018.2794881.

[78] M. Gundall, J. Stegmann, C. D. Huber, and H. D. Schotten, 'Towards Organic 6G Networks: Virtualization and Live Migration of Core Network Functions', *ArXiv*, vol. abs/2110.12737, 2021.

[79] K. Govindaraj, M. Saha, A. Artemenko, and A. Kirstaedter, 'Investigation of Uninterrupted Service Live Migration Using Software-Defined Networking', in *2019 International Conference on Networked Systems (NetSys)*, IEEE, Mar. 2019, pp. 1–6. doi: 10.1109/NetSys.2019.8854521.

[80] A. Randazzo and I. Tinnirello, 'Performance Analysis of Memory Cloning Solutions in Mobile Edge Computing', in *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, IEEE, Oct. 2018, pp. 256–261. doi: 10.1109/IoTSMS.2018.8554666.

[81] Z. Lei, E. Sun, S. Chen, J. Wu, and W. Shen, 'A Novel Hybrid-Copy Algorithm for Live Migration of Virtual Machine', *Future Internet*, vol. 9, no. 3, pp. 1–13, 2017, [Online]. Available: https://www.mdpi.com/1999-5903/9/3/37

[82] D. Sabella, A. Reznik, and R. Frazao, *Multi-Access Edge Computing in Action*. CRC Press, 2019. doi: 10.1201/9780429056499.

[83] H. Liu, C.-Z. Xu, H. Jin, J. Gong, and X. Liao, 'Performance and energy modeling for live migration of virtual machines', in *Proceedings of the 20th international symposium on High performance distributed computing*, New York, NY, USA: ACM, Jun. 2011, pp. 171–182. doi: 10.1145/1996130.1996154.

[84] C. Jo, Y. Cho, and B. Egger, 'A machine learning approach to live migration modeling', in *Proceedings of the 2017 Symposium on Cloud Computing*, New York, NY, USA: ACM, Sep. 2017, pp. 351–364. doi: 10.1145/3127479.3129262.

[85] M. M. Rajendra, M. Patra, and M. Srinivasan, 'Optimal Rate and Distance Based Bandwidth Slicing in UAV Assisted 5G Networks', in *ICC 2022 - IEEE International Conference on Communications*, IEEE, May 2022, pp. 1–6. doi: 10.1109/ICC45855.2022.9839047.

[86] A. Javed, J. Robert, K. Heljanko, and K. Främling, 'IoTEF: A Federated Edge-Cloud Architecture for Fault-Tolerant IoT Applications', *J Grid Comput*, vol. 18, no. 1, pp. 57–80, 2020, doi: 10.1007/s10723-019-09498-8.

[87] A. V Dastjerdi and R. Buyya, 'Fog Computing: Helping the Internet of Things Realize Its Potential', *Computer (Long Beach Calif)*, vol. 49, no. 8, pp. 112–116, 2016, doi: 10.1109/MC.2016.245.

[88] O. Debauche *et al.*, 'Towards Landslides Early Warning System With Fog - Edge Computing And Artificial Intelligence', *Journal of Ubiquitous Systems and Pervasive Networks*, vol. 15, no. 02, pp. 11–17, Mar. 2021, doi: 10.5383/JUSPN.15.02.002.

[89] J. Abdelaziza, M. Adda, and H. Mcheick, ‘An Architectural Model for Fog Computing’, *Journal of Ubiquitous Systems and Pervasive Networks*, vol. 10, no. 1, pp. 21–25, Mar. 2018, doi: 10.5383/JUSPN.10.01.003.

[90] M. Chiang and T. Zhang, ‘Fog and IoT: An Overview of Research Opportunities’, *IEEE Internet Things J*, vol. 3, no. 6, pp. 854–864, Dec. 2016, doi: 10.1109/JIOT.2016.2584538.

[91] C. Zhang and Z. Zheng, ‘Task migration for mobile edge computing using deep reinforcement learning’, *Future Generation Computer Systems*, vol. 96, pp. 111–118, Jul. 2019, doi: 10.1016/j.future.2019.01.059.

[92] S. Moon, J. Park, and Y. Lim, ‘Task Migration Based on Reinforcement Learning in Vehicular Edge Computing’, *Wirel Commun Mob Comput*, vol. 2021, pp. 1–10, May 2021, doi: 10.1155/2021/9929318.

[93] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, ‘Delay-Aware Microservice Coordination in Mobile Edge Computing: A Reinforcement Learning Approach’, *IEEE Trans Mob Comput*, vol. 20, no. 3, pp. 939–951, Mar. 2021, doi: 10.1109/TMC.2019.2957804.

[94] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, ‘iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments’, *Softw Pract Exp*, vol. 47, no. 9, pp. 1275–1296, 2017, doi: 10.1002/spe.2509.

[95] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, ‘CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms’, *Softw Pract Exp*, vol. 41, no. 1, pp. 23–50, 2011, doi: 10.1002/spe.995.

[96] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, ‘SUMO – Simulation of Urban MObility: An Overview’. ThinkMind, 2011. [Online]. Available: https://elib.dlr.de/71460/

[97] L. Codeca, R. Frank, and T. Engel, ‘Luxembourg SUMO Traffic (LuST) Scenario: 24 hours of mobility for vehicular networking research’, in *2015 IEEE Vehicular Networking Conference (VNC)*, IEEE, Dec. 2015, pp. 1–8. doi: 10.1109/VNC.2015.7385539.

[98] S. Panwar, ‘Breaking the millisecond barrier: Robots and self-driving cars will need completely reengineered networks’, *IEEE Spectr*, vol. 57, no. 11, pp. 44–49, 2020, doi: 10.1109/MSPEC.2020.9262144.

[99] M. Agiwal, A. Roy, and N. Saxena, ‘Next Generation 5G Wireless Networks: A Comprehensive Survey’, *IEEE Communications Surveys & Tutorials*, vol. 18, no. 3, pp. 1617–1655, 2016, doi: 10.1109/COMST.2016.2532458.

[100] B. Sonkoly *et al.*, ‘Scalable edge cloud platforms for IoT services’, *Journal of Network and Computer Applications*, vol. 170, pp. 1–18, 2020, doi: https://doi.org/10.1016/j.jnca.2020.102785.

[101] S. Yangui, ‘A Panorama of Cloud Platforms for IoT Applications Across Industries’, *Sensors*, vol. 20, no. 9, pp. 1–19, 2020, [Online]. Available: https://www.mdpi.com/1424-8220/20/9/2701

[102] P. Jamshidi, C. Pahl, N. C. Mendonca, J. Lewis, and S. Tilkov, ‘Microservices: The Journey So Far and Challenges Ahead’, *IEEE Softw*, vol. 35, no. 3, pp. 24–35, May 2018, doi: 10.1109/MS.2018.2141039.

[103] Y. Qiu, C.-H. Lung, S. Ajila, and P. Srivastava, ‘Experimental evaluation of LXC container migration for cloudlets using multipath TCP’, *Computer Networks*, vol. 164, p. 106900, Dec. 2019, doi: 10.1016/j.comnet.2019.106900.

[104] P. Kathiravelu, Z. Zaiman, J. Gichoya, L. Veiga, and I. Banerjee, ‘Towards an internet-scale overlay network for latency-aware decentralized workflows at the edge’, *Computer Networks*, vol. 203, p. 108654, Feb. 2022, doi: 10.1016/j.comnet.2021.108654.

[105] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, ‘Mobile Edge Computing A key technology towards 5G’, *European Telecommunications Standards Institute (ETSI)*, no. 11. pp. 1–16, Sep. 2015. Accessed: Sep. 09, 2022. [Online]. Available: https://www.etsi.org/images/files/etsiwhitepapers/etsi_wp11_mec_a_key_technology_towards_5g.pdf

[106] C. Li, C. Qianqian, and Y. Luo, ‘Low-latency edge cooperation caching based on base station cooperation in SDN based MEC’, *Expert Syst Appl*, vol. 191, p. 116252, Apr. 2022, doi: 10.1016/j.eswa.2021.116252.

[107] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, ‘Mobile Edge Computing and Networking for Green and Low-Latency Internet of Things’, *IEEE Communications Magazine*, vol. 56, no. 5, pp. 39–45, May 2018, doi: 10.1109/MCOM.2018.1700882.

[108] A. A. Mutlag, M. K. Abd Ghani, N. Arunkumar, M. A. Mohammed, and O. Mohd, ‘Enabling technologies for fog computing in healthcare IoT systems’, *Future Generation Computer Systems*, vol. 90, pp. 62–78, 2019, doi: https://doi.org/10.1016/j.future.2018.07.049.

[109] U. Ramachandran, H. Gupta, A. Hall, E. Saurez, and Z. Xu, ‘A Case for Elevating the Edge to be a Peer of the Cloud’, *GetMobile: Mobile Computing and Communications*, vol. 24, no. 3, pp. 14–19, Jan. 2021, doi: 10.1145/3447853.3447859.

[110] A. Rezazadeh, D. Abednezhad, and H. Lutfiyya, ‘Hybrid-MiGrror: An Extension to the Hybrid Live Migration to Support Mobility in Edge

Computing', *Journal of Ubiquitous Systems & Pervasive Networks*, vol. 18, no. 1, pp. 39–48, 2023, doi: 10.5383/JUSPN.18.01.006.

[111] S. Aleyadeh, A. Moubayed, P. Heidari, and A. Shami, 'Optimal Container Migration/Re-Instantiation in Hybrid Computing Environments', *IEEE Open Journal of the Communications Society*, vol. 3, pp. 15–30, 2022, doi: 10.1109/OJCOMS.2022.3140272.

[112] A. Bhardwaj and C. Rama Krishna, 'Efficient multistage bandwidth allocation technique for virtual machine migration in cloud computing', *Journal of Intelligent & Fuzzy Systems*, vol. 35, no. 5, pp. 5365–5378, Nov. 2018, doi: 10.3233/JIFS-169819.

[113] A. Rezazadeh and H. Lutfiyya, 'Multi-microservice migration modelling, comparison, and potential in 5G/6G mobile edge computing: A non-average parameter values approach', *arXiv:2305.10977 [cs.NI]*, 2023, Accessed: May 17, 2023. [Online]. Available: https://arxiv.org/abs/2305.10977

[114] J. Zhang, F. Ren, R. Shu, T. Huang, and Y. Liu, 'Guaranteeing Delay of Live Virtual Machine Migration by Determining and Provisioning Appropriate Bandwidth', *IEEE Transactions on Computers*, vol. 65, no. 9, pp. 2910–2917, Sep. 2016, doi: 10.1109/TC.2015.2500560.

[115] J. Zhang, F. Ren, and C. Lin, 'Delay guaranteed live migration of Virtual Machines', in *IEEE INFOCOM 2014 - IEEE Conference on Computer Communications*, IEEE, Apr. 2014, pp. 574–582. doi: 10.1109/INFOCOM.2014.6847982.

[116] L. Chaufournier, P. Sharma, F. Le, E. Nahum, P. Shenoy, and D. Towsley, 'Fast transparent virtual machine migration in distributed edge clouds', in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, New York, NY, USA: ACM, Oct. 2017, pp. 1–13. doi: 10.1145/3132211.3134445.

[117] S. Dharmaraj and P. Kavitha, 'Cloud Network Communication Performance Improvement Using a Stochastic Bandwidth Allocation and Swarm Optimization Algorithm', 2023, pp. 113–125. doi: 10.1007/978-981-19-2538-2_11.

[118] A. Al-Najjar, S. Layeghy, and M. Portmann, 'Pushing SDN to the end-host, network load balancing using OpenFlow', in *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, IEEE, Mar. 2016, pp. 1–6. doi: 10.1109/PERCOMW.2016.7457129.

[119] J. Zhang, E. Dong, J. Li, and H. Guan, 'MigVisor: Accurate Prediction of VM Live Migration Behavior using a Working-Set Pattern Model', *ACM*

SIGPLAN Notices, vol. 52, no. 7, pp. 30–43, Sep. 2017, doi: 10.1145/3140607.3050753.

[120] A. Leivadeas, N. Pitaev, and M. Falkner, ‘Analyzing the Performance of SD-WAN Enabled Service Function Chains Across the Globe with AWS’, in *Proceedings of the 2023 ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA: ACM, Apr. 2023, pp. 125–135. doi: 10.1145/3578244.3583722.

[121] ‘CRIU’. [Online]. Available: https://criu.org/Docker

Appendix A: Bandwidth Strategy for Pre-copy

This section presents the performance of the pre-copy migration with and without using the proposed bandwidth strategy in Chapter 5 to reduce downtime and migration time of multiple services.

Figure A.1 depicts the downtime, migration time, and migration overhead of the pre-copy migration method with and without the increased bandwidth allocation strategy. As shown in Figure A.1, we increase the bandwidth of the last migration round by a factor of two, five, and ten compared to its original bandwidth. When compared to the initial bandwidth results, the downtime for the pre-copy method that employs the bandwidth strategy is 50% at twofold bandwidth, 20% at fivefold bandwidth, and 10% at tenfold bandwidth. For instance, when the downtime is 320 ms with the original bandwidth, it drops to 160 ms using double the bandwidth. Furthermore, the downtime decreases to 80 ms and 40 ms, respectively, when bandwidth is increased by fivefold and tenfold compared to the original bandwidth. We achieve these results while using the additional bandwidth (additional transfer rate) for less than 2.5 percent of the total migration duration.

Furthermore, as shown in Figure A.1, using the pre-copy technique with the increased bandwidth allocation strategy reduces migration time between 21.99% and 35.31% using a twofold increase in transfer rate only at the first migration round, compared to using the initial bandwidth. In addition, compared to the original bandwidth of the pre-copy technique, the migration time decreases between 37.31% and 56.51% using a fivefold increased transfer rate at the first event of migration. Migration time reduction for a tenfold increased transfer rate at the first migration event is between 42.90 and 63.57 percent.

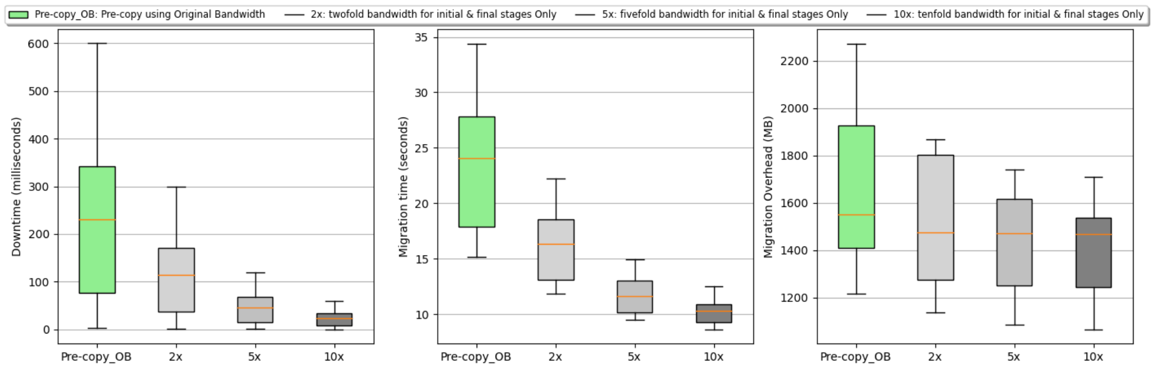

Figure A.1 Comparison of the pre-copy migration method with and without the use of bandwidth strategy (light green: pre-copy without bandwidth strategy - using original bandwidth, greys: pre-copy using bandwidth strategy; 2x, 5x, and 10x: twofold, fivefold, and tenfold transfer rate only during the first and last migration stages) (left (A): Downtime, middle (B): Migration Time, right (C): Migration Overhead).

Because of the shorter migration time, resources at the source can be made more quickly available to other containers, allowing more users and containers to be served with higher performance while using the same available resources. Additionally, the shorter migration time results in a reduction in migration overhead. The decreased migration overhead is a result of fewer dirty memory transfers during the migration, as the migration is shorter when using an increased bandwidth allocation strategy. The source node must transfer fewer dirty memories, resulting in a reduction in migration overhead. For example, compared to the original bandwidth of the pre-copy technique, the migration overhead decreases by 17% using a twofold increased transfer rate at the first event of migration. In addition, compared to the original bandwidth of the pre-copy technique, the migration overhead decreases by 23% using a fivefold increased transfer rate at the first event of migration. Migration overhead reduction for a tenfold increased transfer rate at the first migration event is 24 percent. These results indicate that, despite increasing bandwidth for a period of time during migration, we can reduce migration overhead, and this benefit should be considered when employing the bandwidth allocation strategy.

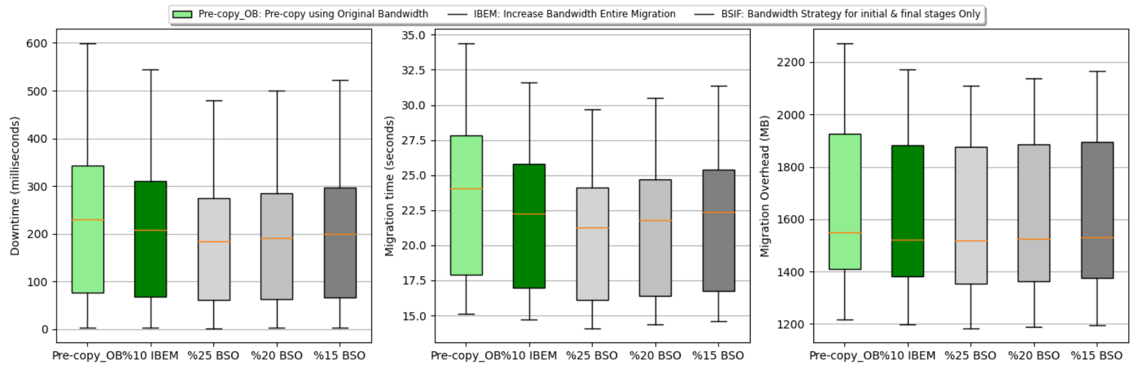

Figure A.2 Comparison of the pre-copy migration method with and without the use of bandwidth strategy (light green: pre-copy without bandwidth strategy, dark green: increase transfer rate for entire migration, greys: pre-copy using bandwidth strategy; %15, %20, and %25 increase transfer rate only during the first and last migration stages) (left (A): Downtime, middle (B): Migration Time, right (C): Migration Overhead).

Figure A.2 shows another example of the proposed strategy's strength. The light green bars show the results of pre-copy using the original bandwidth, while the dark green bars show what happens when 10% more bandwidth is assigned to each container for the entire migration duration. The grey bars represent bandwidth increases of 15%, 20%, and 25% for only the initial and final migration stages. The plots show that bandwidth utilization increased as downtime, migration time, and migration overhead decreased when the proposed strategy was used. For example, when there is a 20% increase in bandwidth for the initial and final stages of migration, 50% of containers require simultaneous migration. Bandwidth utilization increases as the proportion of containers requiring simultaneous migration decreases. This means that the better the performance, the lower the percentage of available bandwidth for the initial and final migration stages.

Another example of the proposed strategy's strength is shown in Figure A.3. The light green bars represent the pre-copy results using the original bandwidth. The grey bars represent bandwidth increases of 15%, 20%, and 25% for only the initial and final migration stages, while the transfer rate of the middle migration stages is reduced by 10% to free up bandwidth and is assigned to the initial and final migration stages. Again, the plots show that bandwidth utilization increased when the proposed strategy was used while downtime, migration time, and migration overhead decreased when compared to the results of the original bandwidth. For example, when there is a 20% increase in bandwidth for the initial and final stages of migration, 50% of containers require simultaneous migration. Bandwidth utilization increases as the proportion of containers requiring simultaneous migration decreases. This means that the better the performance, the lower the percentage of available bandwidth for the initial and final migration stages.

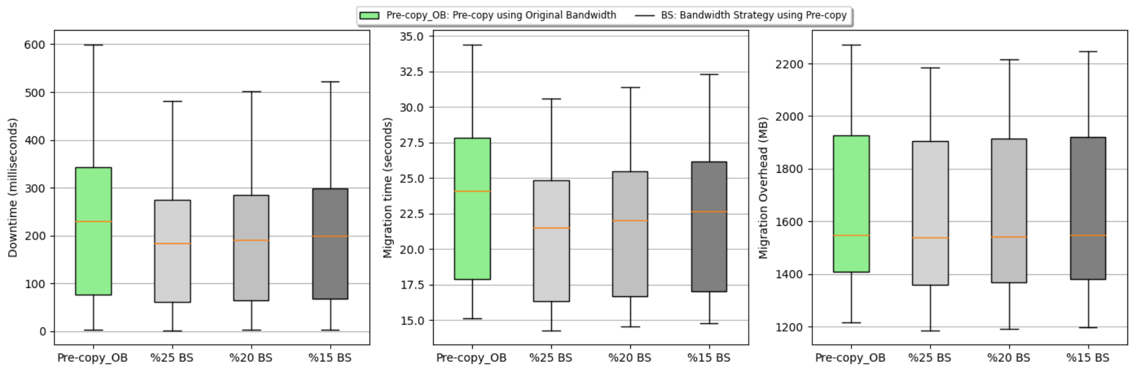

Figure A.3 Comparison of the pre-copy migration method with and without the use of bandwidth strategy (light green: pre-copy without bandwidth strategy, greys: pre-copy using bandwidth strategy; %15, %20, and %25 increase transfer rate only during the first and last migration stages, and decrease transfer rate during middle migration stages by %10) (left (A): Downtime, middle (B): Migration Time, right (C): Migration Overhead).

Figure A.4 depicts the most significant results. To generate this figure, we use the bandwidth strategy to reduce total bandwidth by 10% (grey bars) and compare the results to regular pre-copy using 100% bandwidth (light green bar). Even with less bandwidth, the downtime is lower than the original pre-copy, as shown in Figure A.4. Furthermore, the migration time is slightly shorter than when using the original pre-copy. The only parameter that has increased is migration overhead, which has increased slightly. The results of Figure A.4 show that, despite the bandwidth strategy's reduced bandwidth, downtime and migration time are still lower than when using 100% bandwidth and the original pre-copy.

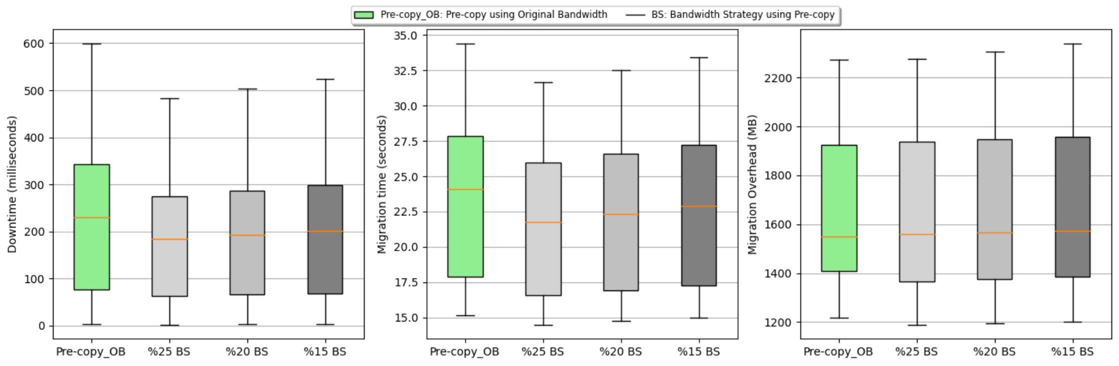

Figure A.4 Comparison of the pre-copy migration method with and without the use of bandwidth strategy (light green: pre-copy without bandwidth strategy, greys: pre-copy using bandwidth strategy; %15, %20, and %25 increase transfer rate only during the first and last migration stages, and decrease transfer rate during middle migration stages by %10) (left (A): Downtime, middle (B): Migration Time, right (C): Migration Overhead).

Curriculum Vitae

Name:	Arshin Rezazadeh
Post-secondary Education and Degrees:	The University of Western Ontario London, Ontario, Canada 2023 Ph.D.
	Iran University of Science and Technology Tehran, Tehran, Iran 2008 M.Sc.
	Shahid Chamran University of Ahvaz Ahvaz, Khuzestan, Iran 2004 B.Sc.
Honors and Awards:	Western Graduate Research Scholarship (WGRS) 2018-2022
	Best Paper Award, FNC 2022 conference
	Invited submission to special issue of the International Journal of Ubiquitous Systems and Pervasive Networks (for the FNC 2022 paper)
	IEEE Globecom conference 2023 Travel Grant.
Related Work Experience	Graduate Research Assistant The University of Western Ontario 2018-2023
	Graduate Teaching Assistant The University of Western Ontario 2018-2023

Related Publications:

Arshin Rezazadeh, Hanan Lutfiyya “A Novel Sustainable Bandwidth Allocation Strategy for Multiple Service Migration in 5G/6G Edge Computing,“ *GLOBECOM 2023 - 2023 IEEE Global Communications Conference*, IEEE, Dec. 2023, pp. 1211–1217

Arshin Rezazadeh, Hanan Lutfiyya “Multi-microservice Migration Modelling, Comparison, and Potential in 5G/6G Mobile Edge Computing: A Non-average Parameter Values Approach,” IEEE Access –accepted

Arshin Rezazadeh, Davoud Abednejad, Hanan Lutfiyya “Hybrid-MiGrror: An Extension to the Hybrid Live Migration to Support Mobility in Edge Computing,” *Journal of Ubiquitous Systems & Pervasive Networks, Vol. 18, No. 1,* pp. 39-48, January 2023 [INVITED PAPER]

Arshin Rezazadeh, Davoud Abednejad, Hanan Lutfiyya “MiGrror: Mitigating Downtime in Mobile Edge Computing, An Extension to Live Migration, “*Procedia Computer Science, Vol. 203,* pp. 41-50, 2022 [BEST PAPER AWARD]

Arshin Rezazadeh, Hanan Lutfiyya “Migration in Edge Computing: Review and Challenges,” –submitted