CMPE 492

COMIC-RAN: Container-based Migration in C-RAN

Ömer Şükrü Uyduran

Advisor:

Tuna Tuğcu

January 13, 2025

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to my project supervisor, Professor Tuna Tuğcu, for his invaluable guidance, insightful feedback, and unwavering support throughout the duration of this project.

TABLE OF CONTENTS

1. INTRODUCTION

In recent years, cellular network technologies have seen significant advancements, driven by the growing demands for higher data rates, lower latency, and improved energy efficiency. The evolution from 4G to 5G has paved the way for new architectures like Cloud Radio Access Networks (Cloud-RAN), which offer promising solutions for the increasing challenges faced by mobile network providers. With the rapid expansion of network coverage and the rise of mobile devices, the necessity for a more efficient approach to infrastructure and resource management is apparent. This project focuses on container-based migration within Cloud-RAN, utilizing virtualization techniques to migrate running server processes between virtual machines (VMs) with minimal disruption, as part of an effort to contribute to the emerging technological landscape of 5G.

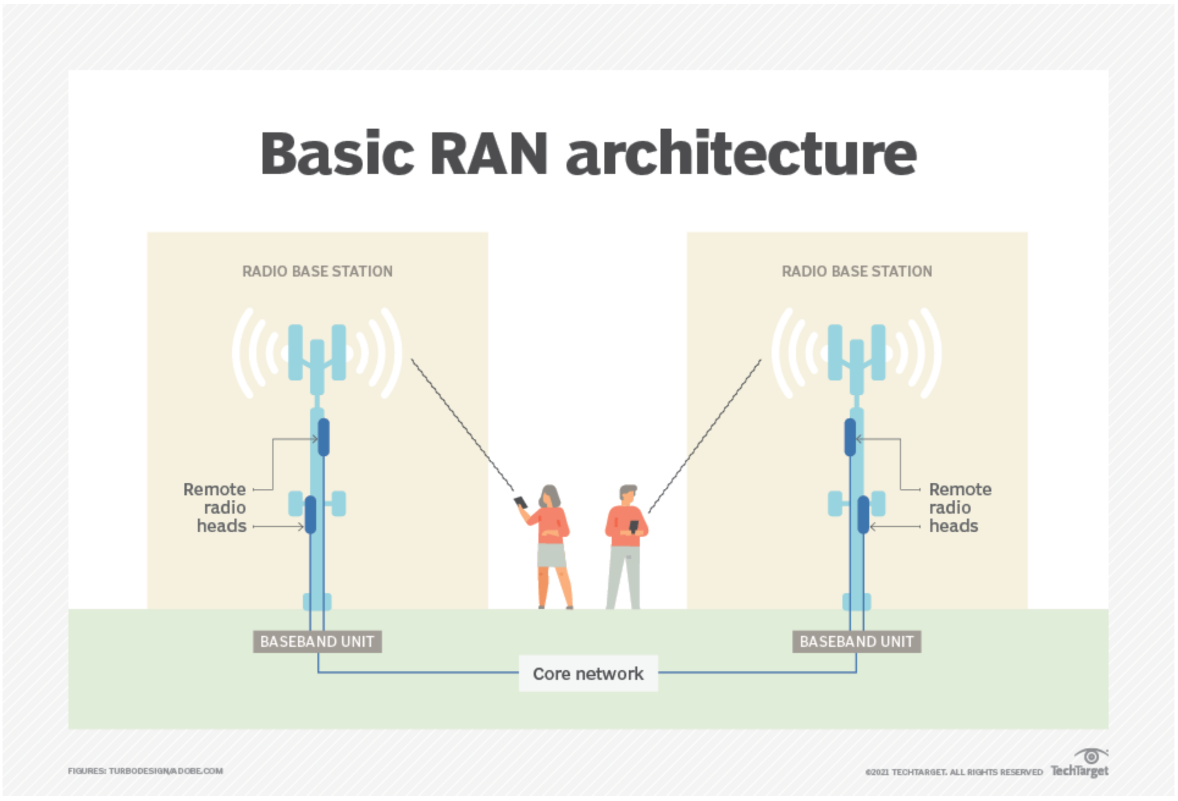

Figure 1.1. Basic RAN Architecture (taken from [4]).

The traditional approach to Radio Access Networks (RAN) involves distributed Baseband Units (BBUs) and Remote Radio Heads (RRHs), where each BBU is dedicated to its respective RRH. This setup leads to significant challenges, such as increased operational costs and inflexible resource management, which become particularly problematic as coverage areas shrink with newer generations of cellular networks. The move to 5G has prompted the introduction of Cloud-RAN, where BBUs are virtualized and centralized, allowing for more dynamic allocation of resources and reducing the burden of hardware maintenance. Unlike earlier work focused on virtual machine-based BBUs, our project leverages containerization for increased efficiency and scalability.

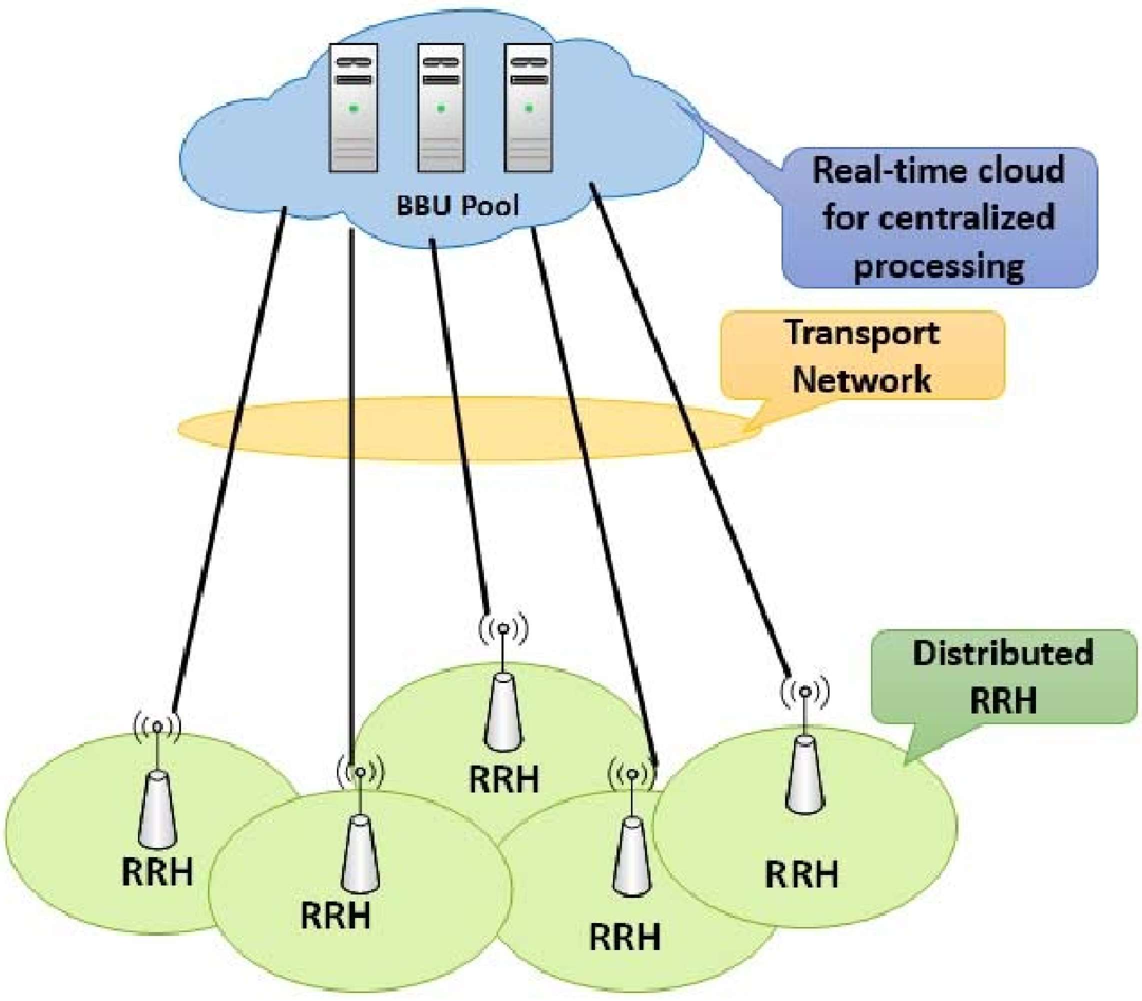

Figure 1.2. C-RAN Architecture (taken from [5]).

Our project aims to achieve live migration of server processes running inside containers between two virtual machines, minimizing downtime and ensuring high availability—critical aspects of achieving compliance with stringent 5G latency requirements. Instead of relying on third-party containerization tools, we have developed our own sandbox environment to ensure simplicity and tailor the functionality to our needs.

Initially, the project envisioned developing a custom containerization tool and novel migration tools to address the unique challenges of stateful server applications with high network dependency and frequent memory changes. However, the complexity and scope of implementing new migration technologies proved to be beyond the timeframe and resources of a single-semester project. Instead, the focus shifted to leveraging CRIU (Checkpoint/Restore in Userspace) for migration, integrating it into the custom containerization tool we developed. While the project remains incomplete, this report documents the work and research we have done so far, challenges encountered, and insights gained.

In this final report, we present the overall progress made in implementing the sandbox environment, preparation of sandbox environment to work with a process checkpoint and restore tool called CRIU[1] , implementation of UDP server and client applications, preparing the necessary infrastructure for test simulations, research of existing migration methodologies, and a new approach that might fit to our specific use case (which we couldn't implement and test yet).

1.1. Broad Impact

The impact of this project extends beyond the academic exploration of container migration. The telecommunications industry is currently undergoing a major transformation, with a push toward more flexible and cost-efficient solutions to meet the increasing demands of mobile data users. By focusing on container-based migration in Cloud-RAN, our project directly contributes to addressing some of the major challenges faced in rolling out 5G infrastructure, specifically by reducing costs, improving

[1]`https://criu.org/`

resource utilization, and minimizing network latency. These improvements are critical in delivering the high-speed, reliable connectivity promised by 5G and ensuring that network operators can meet user expectations without incurring prohibitive capital and operational expenditures.

The methodology of using lightweight containers instead of traditional VMs also offers significant potential benefits for the environment. With reduced resource overhead and improved efficiency, container-based Cloud-RAN can contribute to lower energy consumption in cellular networks, thereby supporting sustainability initiatives in the telecommunications industry. Additionally, the implementation of our own sandbox environment demonstrates the feasibility of customized containerization solutions that are specifically tailored to the needs of 5G and beyond, offering a pathway for future research and development in more targeted virtualization techniques.

Another important impact is on network reliability. By employing custom migration techniques, we hope that the downtime experienced by users during the migration of active network components will be reduced. This directly translates into better user experience and satisfaction, as well as a more robust network capable of handling the demands of emerging applications such as augmented reality, virtual reality, and autonomous systems, which all require consistent, low-latency connectivity.

1.2. Ethical Considerations

1.2.1. Access

All of the source code, documentation, and the guides are published as open to everyone's access in the project's GitHub repository[2] .

[2] `https://github.com/Simurgan/comicran`

1.2.2. Transparency and Accountability

Methodologies and important aspects of the project will be reported to ensure interested ones can reproduce the outcomes and evaluate the results.

2. PROJECT DEFINITION AND PLANNING

2.1. Project Definition

This project explores the implementation and evaluation of a container-based live migration mechanism within Cloud-RAN, focusing on migrating server processes between virtual machines with minimal disruption. Initially, the plan was to design and develop a custom containerization / sandboxing tool tailored to this purpose and implement a novel migration protocol specifically for the scenario of stateful server programs that heavily depend on network communications and experience frequent memory changes.

Due to the complexity and time constraints of the project, the scope was adjusted to utilize CRIU (Checkpoint/Restore in Userspace) for process migration, integrating it into the custom containerization tool we implemented in the first half of the project. The project now aims to find a more optimal process migration technique tailored to Cloud-RAN compared to existing techniques. The objective remains to achieve minimal downtime and maintain high availability during the live migration process, aligning with the stringent latency and reliability requirements of 5G networks.

Additionally, tools were developed to simulate test scenarios, including a simple UDP server-client application and SDN to mimic BBU server operations and the associated traffic. This setup provides a controlled environment for evaluating migration efficiency and the ability to adapt to high network traffic and dynamic memory changes.

2.2. Project Planning

So far, we have developed a custom containerization tool to create sandbox environments, implemented tools for test simulation and UDP server-client applications along with an SDN to mimic Cloud-RAN network traffic, and prepared the containerization tool for CRIU usage to enable process checkpointing and restoration. We accomplished to perform a cold migration between different containerized environments that are working on different virtual machines. We also conducted research on containerization tools such as LXC/LXD and explored their migration techniques. Moreover, we experimented with CRIU and achieved to checkpoint processes inside our sandbox tool which was working on a virtual machine and restored them in a sandbox on a different virtual machine.

2.2.1. Remaining Work and Future Planning

The semester has ended, and while significant progress has been made, the project remains incomplete, with several important tasks yet to be addressed. The remaining work includes conducting advanced research on utilizing CRIU through its C API for better integration and control. This involves implementing existing migration techniques such as pre-copy, post-copy, and hybrid approaches within the custom containerization tool's sandboxes. Efforts will then focus on minimizing downtime by modifying these methodologies to suit the specific requirements of the project. If necessary, novel migration techniques tailored to the scenario will be devised and implemented.

Testing these methods with realistic simulations will be critical. This includes evaluating the developed migration approaches on the custom containerization tool and comparing the results with the live migration techniques of existing containerization tools like LXC/LXD along with their migration techniques. The results should be analyzed to understand their efficiency and impact on downtime and overall performance.

2.2.2. Time Estimation

We estimate that the remainder of the project needs approximately 1 additional month to be completed.

2.2.3. Success Criteria

The success of this project will be measured based on two primary criteria: minimizing downtime and additional response delays during the live migration process. The downtime should be lower than existing solutions and should be eliminated if possible. The additional response delays should be acceptable. These criteria are essential to ensure that the migration process is seamless and that users experience no noticeable disruption in service.

2.2.4. Risk Analysis

Performing live migration poses several challenges, particularly in synchronizing memory changes between the source and destination machines. The frequent updates to memory during the migration process make it difficult to achieve zero downtime, as any discrepancies between the two instances can lead to inconsistencies and service interruptions. It is a considerable possibility that we may not achieve complete zero downtime, given the complexity of managing synchronization in real time. However, our efforts will focus on minimizing downtime as much as possible, aiming for an acceptable level of service continuity.

2.2.5. Team Work

As a student, I am working on this project alone. However, my project advisor Tuna Tuğcu supports me during our weekly meetings. He brainstorms with me on the challenges I encounter, suggests me critical techniques for the problems, and conveys his vision to me.

3. RELATED WORK

3.1. 5G/CloudRAN [1]

The 5G/CloudRAN project by Ahmet Bugrahan Tasdan and Salih Sevgican aimed to implement a Cloud-RAN architecture for 5G networks, focusing on the virtualization of BBUs using virtual machines. Their work primarily utilized OpenStack to create virtual BBUs, which were then migrated between different physical servers. This project demonstrated the feasibility of using virtual machines for live migration within a Cloud-RAN environment, successfully achieving migration without significant downtime. However, the use of virtual machines posed challenges in terms of resource overhead and memory footprint, which impacted the efficiency of the migration process. By using container-based virtualization, our project aims to address these limitations by providing a more lightweight and scalable solution.

Tasdan and Sevgican's work also included the use of Software Defined Networking (SDN) to manage network traffic during migration, ensuring that the network remained stable and consistent throughout the process. Their approach to SDN integration serves as a foundation for our own use of SDN in managing container migrations, particularly in ensuring minimal packet loss and maintaining service quality during the transition.

3.2. Container-Founded 5G vRAN Network [2]

The Container-Founded 5G vRAN Network project by Ömer Cihan Benzer and Yunus Emre Topal focused on implementing a virtual Radio Access Network (vRAN) using containerization instead of traditional virtual machines. Their work highlighted the advantages of using containers for vRAN, including reduced resource overhead and increased scalability. The project utilized Docker and Kubernetes to manage and orchestrate the containerized BBUs, demonstrating that container-based vRAN could achieve comparable performance to VM-based approaches while offering significant improvements in resource utilization.

Benzer and Topal also explored the challenges of live container migration, particularly in maintaining low latency and minimizing packet loss. They utilized Kubernetes' built-in tools to manage container orchestration and ensure that the migration process was as seamless as possible. Their project serves as a direct precursor to our own work, providing valuable insights into the benefits and challenges of container-based migration in a 5G context. By building on their findings, we aim to further optimize the migration process, focusing specifically on achieving zero downtime and meeting 5G latency requirements.

Additionally, Benzer and Topal's project referenced the work of Tasdan and Sevgican, noting the differences between VM-based and container-based approaches in terms of efficiency and scalability. Our project continues this line of investigation by directly comparing the performance of our custom sandbox tool with existing containerization solutions, aiming to provide a more tailored and efficient approach for Cloud-RAN migration.

3.3. Container Migration: A Perfomance Evaluation Between MIGrror AND Pre-copy [3]

This is a project carried out by Xinwen Liang at The University of Western Ontario. MiGrror is an innovative migration technique designed to minimize downtime during container live migration. Unlike traditional iterative methods such as pre-copy migration, which transfers memory states at fixed intervals, MiGrror employs an event-driven synchronization mechanism. Memory changes at the source trigger immediate synchronization events, allowing smaller, incremental updates to be transferred to the destination. This approach reduces the amount of data that needs to be transferred during the final migration phase, resulting in significantly lower downtime.

The MiGrror technique was evaluated in a realistic testbed, comparing its performance against pre-copy migration using Podman containers. The experiments revealed that MiGrror consistently achieved lower application downtime while maintaining comparable total network usage and migration time. By integrating event-driven synchronization with CRIU's checkpointing capabilities, MiGrror demonstrates a promising direction for optimizing container migrations in latency-sensitive environments such as Cloud-RAN.

The development and evaluation of MiGrror emphasize the importance of tailoring migration techniques to specific use cases. While pre-copy migration remains a widely used approach, MiGrror's adaptability to high-frequency memory changes makes it particularly suitable for scenarios involving stateful applications with dynamic memory requirements. This innovative technique contributes to the growing body of research on container migration, providing valuable insights for future advancements in the field.

4. METHODOLOGY

Our approach to container-based migration within Cloud-RAN is designed to address the unique challenges of achieving seamless, low-latency live migration of stateful server processes that heavily depend on network communications and experience frequent memory changes. Initially, our methodology was to implement a post-copy migration process tailored to these needs from scratch. However, due to the complexity and time limitations of the project, we adapted our approach to utilize CRIU (Checkpoint/Restore in Userspace) as the core tool for process checkpointing and restoration within our custom containerization tool.

This revised methodology focuses on leveraging CRIU to experiment with various migration techniques, including pre-copy, post-copy, hybrid, and custom methods, by integrating its capabilities into the sandbox environment. The custom containerization tool has been prepared to allow CRIU to work in it, enabling efficient checkpointing and restoration of processes. These capabilities allow us to explore and refine migration strategies to minimize downtime.

Furthermore, our methodology includes the development of a UDP server-client application and an SDN to simulate real-world network traffic scenarios and evaluate the impact of different migration approaches. By combining these tools with the custom sandbox environment and CRIU, we aim to conduct comprehensive testing and analysis to identify bottlenecks and optimize the migration process for Cloud-RAN use cases.

4.1. Custom Sandbox Environment: Jailor

One of the foundational elements of our methodology is the development of a custom sandbox tool called Jailor. Unlike traditional containerization tools such as Docker or Kubernetes, Jailor is designed specifically to meet the requirements of our project while avoiding unnecessary complexity. Jailor reads configuration files that define the container's parameters, including the root directory, executables, network configurations, and any dependencies. Using Linux features such as pivot_root, unshare, and network namespaces, Jailor isolates server processes in a lightweight, secure environment. This custom approach allows us to focus on core functionalities relevant to Cloud-RAN without the overhead associated with generic containerization tools.

4.2. CRIU: Checkpoint/Restore in Userspace [3]

Checkpoint/Restore in Userspace (CRIU) is a powerful open-source tool that enables the checkpointing and restoration of processes in Linux. By capturing the state of a running process, including its memory, CPU state, file descriptors, and network connections, CRIU allows the process to be stopped and later resumed on the same or a different system. This makes it an essential tool for implementing live migration[4] , as it can preserve the operational continuity of stateful applications.

CRIU provides a flexible C API[5] that allows developers to programmatically manage the migration process. This API facilitates advanced control over checkpointing and restoration operations, enabling the integration of CRIU into custom tools and workflows. Its command-based architecture allows for fine-grained management of the migration process, which is crucial for scenarios requiring minimal downtime.

Many existing containerization platforms, including Docker and LXC/LXD, utilize CRIU to perform live migration of containers. These tools leverage CRIU's capabilities to transfer containerized applications between hosts while maintaining their state, demonstrating its reliability and efficiency. By incorporating CRIU into our custom containerization tool, we aim to harness its robust features to explore and refine migration techniques tailored to the unique requirements of 5G networks.

[3]`https://criu.org/`
[4]`https://criu.org/Live_migration`
[5]`https://criu.org/C_API`

4.3. Migration Process

Our migration process will involve incrementally copying the memory of a running server process from the source virtual machine to the destination virtual machine while maintaining the server's availability. The process in the source destination machine will continue to respond to requests while its memory pages are being transferred to the process in the destination virtual machine. During the migration process, both instances (source and destination) will be kept in sync by re-sending the changed memory pages to the destination process. This can be done by using the pre-copy migration approach, a hybrid of pre-copy and post-copy migration approach, or offering a new approach derived from the existing ones which is probably more suitable to our use case. For example, placing the destination node between the SDN and source node might help us a lot. During the migration, the SDN sends requests to the destination node and the destination node forwards requests to the source node. Then, the process in the source node sends the responds to the destination node along with some changed memory pages. The destination node updates its memory accordingly and sends the respond to the SDN. With this approach, the synchronization and hand-off can be easier since the UDP packages are controlled by the destination node before and after they are sent to the source node. We are not sure about this derived technique, however, we think that it is worth to try.

4.4. Communication Between Components

The architecture of our solution comprises several components that work together to achieve live migration. The main components include:

4.4.1. Jailor

Responsible for setting up and managing the sandbox environment, including setting up network interfaces and isolating namespaces. Jailor ensures that each server process runs in a secure, isolated environment with the necessary configurations for migration.

4.4.2. SDN Controller

Our Software Defined Networking (SDN) controller is responsible for managing network traffic, ensuring that incoming requests are routed to the correct instance (source or destination) during migration. The SDN controller helps us minimize packet loss and ensures consistent connectivity throughout the migration.

4.4.3. Server and Client

The server is a simple UDP-based server that receives numerical requests and returns their squared values. The client sends requests to the server, simulating real-world network traffic that the system must handle during migration, and records timestamps that each UDP package is sent and their reply is received. The server runs inside the sandbox, and its migration is the focal point of our project.

4.4.4. Sdeamon and Smanager

The sdeamon is responsible for managing sandboxes within the virtual machines, listening on a TCP port for commands to start or terminate containers. Smanager acts as an interface for the user to manage these sandboxes, sending commands to sdeamon to control server instances as needed.

We hope that the combination of these components allows us to create a flexible and efficient migration system that minimizes downtime and ensures that network performance is not compromised during migration. By developing custom tools and leveraging SDN for dynamic routing, we are able to address the specific needs of 5G Cloud-RAN, providing a tailored solution that is both efficient and scalable.

4.5. Testing and Evaluation

To validate our methodology, we will design and conduct a series of tests that simulate real-world scenarios, including changing memory utilization and heavy server loads. We will also monitor the network traffic between virtual machines to measure network traffic statistics, such as packet loss, latency, and throughput; to ensure that our solution meets the required latency and reliability standards. The success of these tests will be determined based on metrics such as downtime, packet loss, and response latency, with the goal of achieving minimized downtime and low additional delay in the response times.

Overall, our methodology combines a custom sandbox environment, SDN-based network management, and comprehensive testing to achieve efficient and reliable container migration within Cloud-RAN. This approach not only ensures compliance with 5G standards but also offers a scalable solution for future network deployments.

5. REQUIREMENTS SPECIFICATION

5.1. Functional Requirements

- **Container Management**: The system must provide a mechanism to set up and manage sandbox environments for running isolated server processes. It should automate configuration, dependency resolution, and management of container lifecycles.
- **Live Migration**: The system must support live migration of server processes between virtual machines with minimized downtime. The migration must maintain service availability with acceptable latency.
- **Network Traffic Routing**: An SDN-based approach should manage and route incoming requests to appropriate server instances during the migration, ensuring seamless service without packet loss.
- **User Control**: The system must provide a command line interface for managing sandboxes, allowing users to start, terminate, and migrate server processes across virtual machines.
- **Monitoring Capabilities**: The system must include monitoring tools to track network performance metrics, such as packet loss and latency, to evaluate the efficiency of the migration process.

5.2. Non-functional Requirements

- **Performance**: The system must minimize downtime and latency during migration. Scalability should allow for managing multiple sandboxes across multiple virtual machines.
- **Security**: Processes running inside sandboxes must be securely isolated from the host system.
- **Usability**: The system must simplify container management through automation of configurations and provide an intuitive interface for end-users to control the

migration and sandbox processes.

- **Reliability**: The system must guarantee continuous availability of server processes during migration.

5.3. System Requirements

- **Hardware**: The implementation requires at least two virtual machines with sufficient resources (CPU, memory, storage) to support multiple sandbox instances.
- **Software**: The system must run on a Linux-based OS with support for namespaces, pivot_root, and other isolation features. A compatible C compiler (e.g., GCC) is also required.

6. DESIGN

You can find diagrams for system design and information flow below. In the diagrams, Server-2 within Jailor-2 is illustrated as migrating processes from VM-1 to VM-2. Also, the black arrows represents data flow between the components whereas the blue ones represents actions.

In the Figure 6.1 you can see how SDN manages the network traffic between the Servers and the Client. In this figure, Server-2 is currently being migrated from VM-1 to VM-2. See how SDN handles the traffic whose destination is a migrating server. This is the system design that SDN forwards requests to both source and destination VMs during migration.

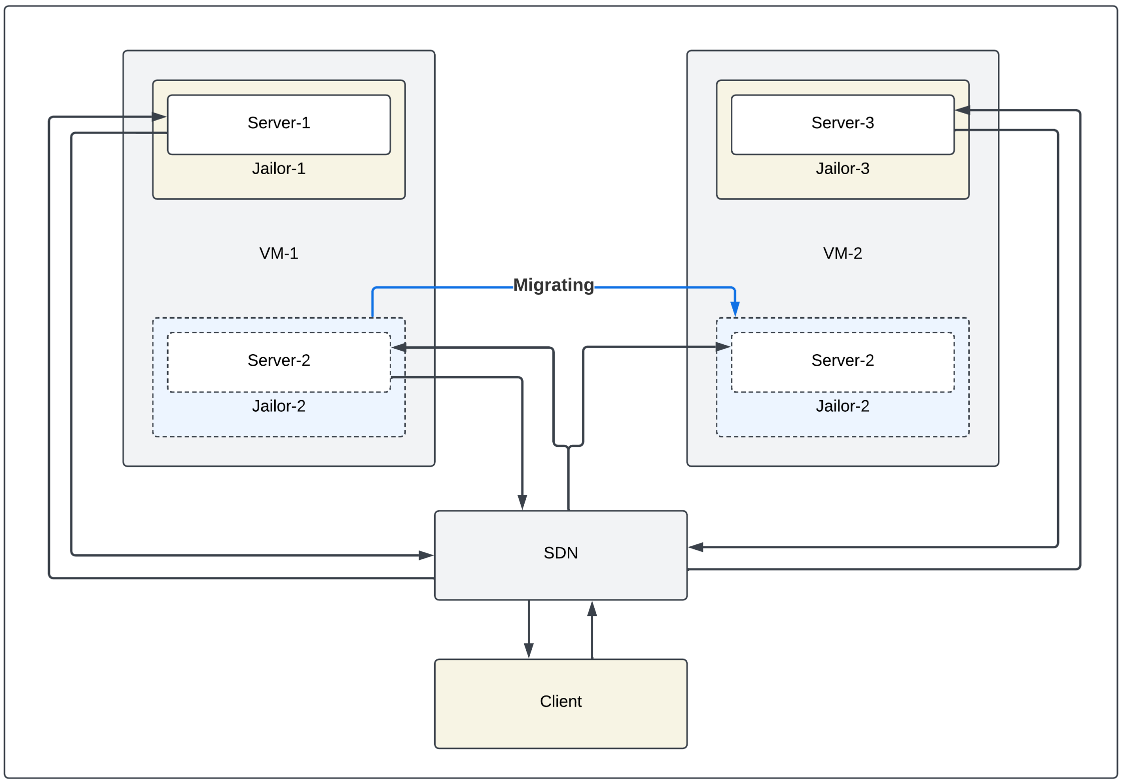

Figure 6.1. Data flow and system design for Server - Client communication.

In the Figure 6.2 you can see how Smanager communicates with the Sdeamons in the VMs and how Sdeamons manage the sandboxes (Jailors) and Servers inside their VM.

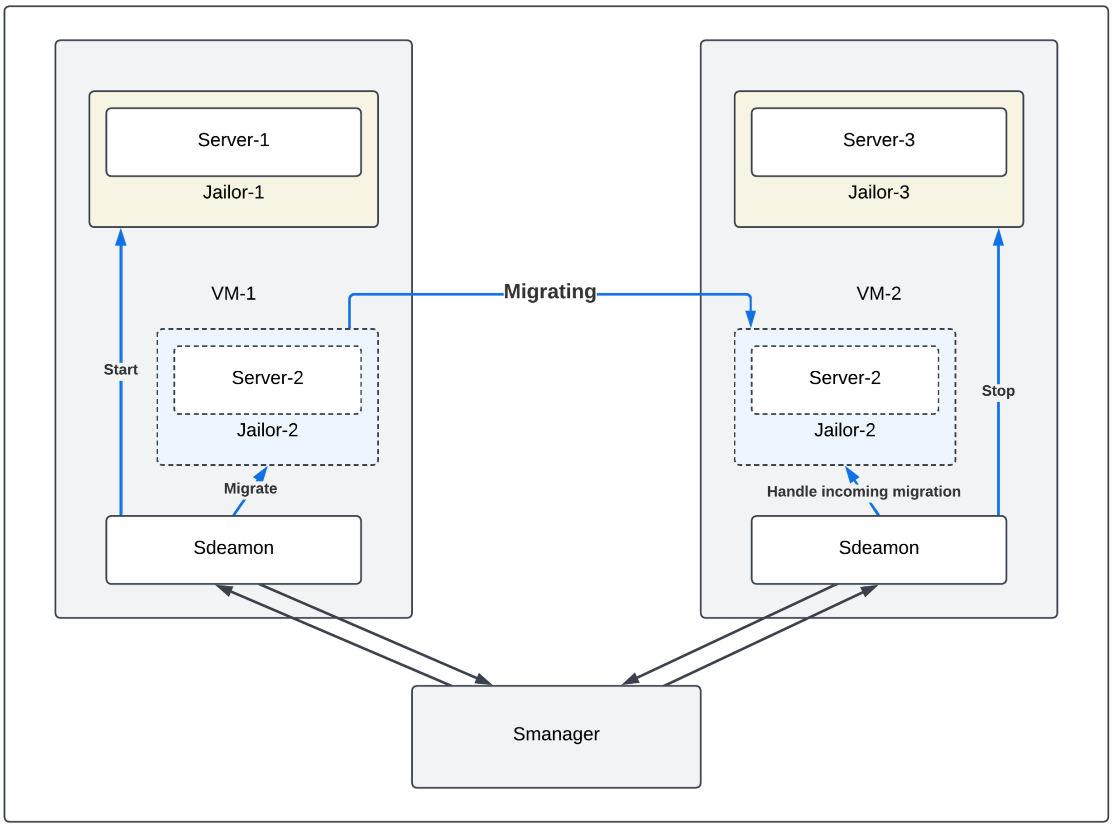

Figure 6.2. Data flow and system design for Smanager - Sdeamon communication.

In the Figure 6.3, you can see the current overall design of the system that SDN forwards requests to both source and destination VMs. This was the first intended system design.

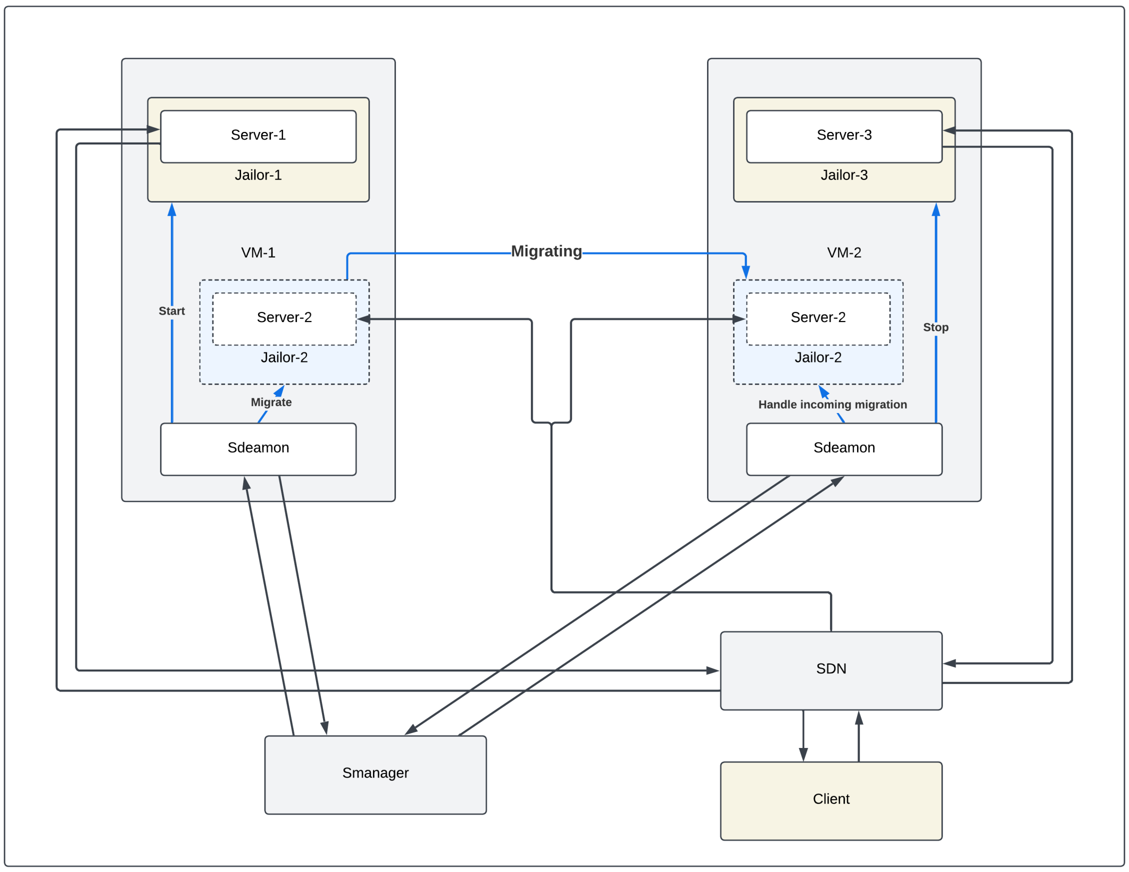

Figure 6.3. Overall system design.

In this Figrue 6.4, you can see an overall desing of the system that SDN forwards requests to only the destination virtual machine which then forwards to the source machine. Then, source machine responds back the result of the request along with the new memory updates. This is a custom approach which can be tested and be suitable for our use case.

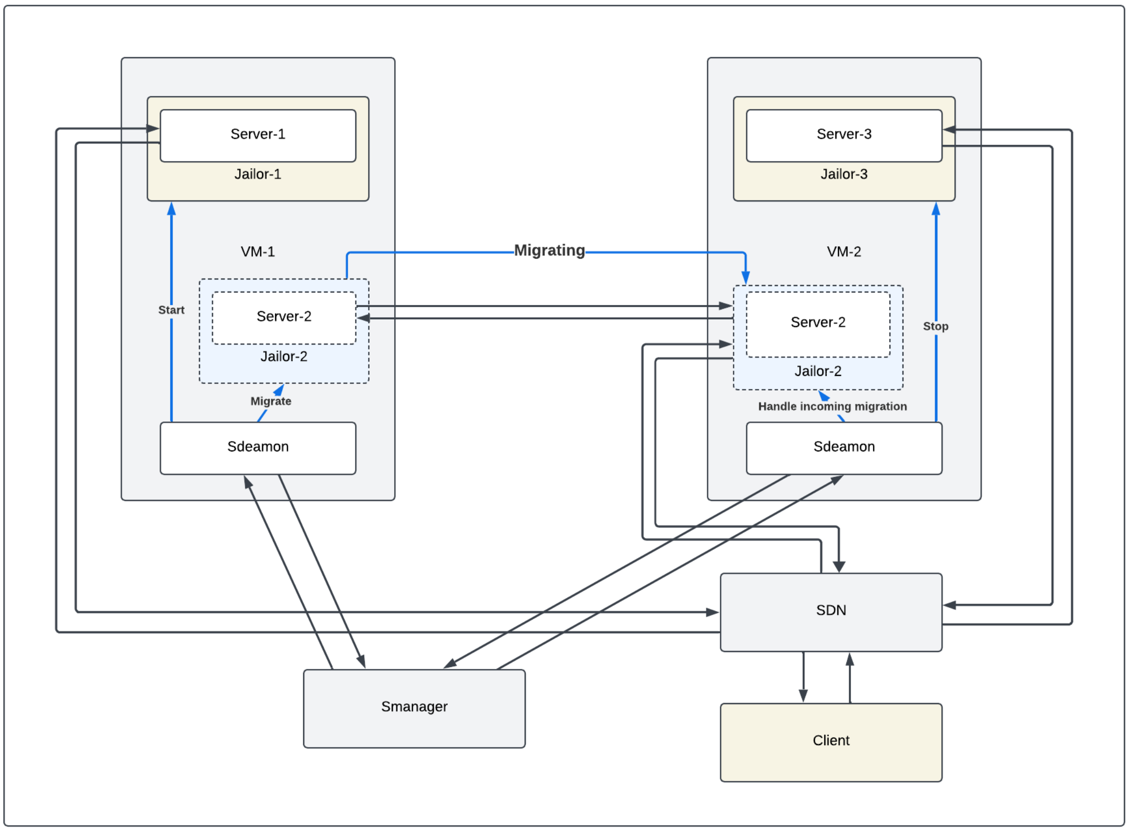

Figure 6.4. System design of the custom approach to migration.

7. IMPLEMENTATION

The implementation of our container-based migration system involves a suite of custom programs that facilitate the setup, management, and migration of server processes across virtual machines. All of these components have been implemented in C programming language, providing a lightweight and efficient solution tailored to the specific requirements of our project. The following sections describe each of these components in detail.

7.1. Configer

The configer program plays a critical role in automating the setup of sandbox environments. It takes a base configuration file as input, which includes details such as the sandbox ID, the path to the root directory, the path of the executable for the root process, command-line arguments for the root process, IP configurations for connectivity, a list of additional executables, required files, and symlinks. Configer is responsible for:

- **Dependency Resolution**: It reads the base configuration file to determine all the dependencies that the root process and additional executables will need within the sandbox. This includes resolving symbolic links to find their actual paths and ensuring all necessary files are listed for inclusion.
- **Output Config Generation**: It generates an output configuration file that includes a comprehensive list of files, executables, and symlinks required for the sandbox. This is especially helpful in eliminating the need for manual dependency management, significantly reducing the time and effort involved in setting up sandbox environments.

The output generated by configer serves as the input for the jailor program, ensuring that the sandbox has all the resources it needs to function correctly.

7.2. Jailor

The jailor program is responsible for creating the sandbox environments based on the configuration file provided by configer. It performs the following tasks:

- **Root Directory Setup**: Jailor creates the root directory for the sandbox and copies all the required files listed in the configuration file to this directory.
- **Network Setup**: It creates a virtual Ethernet (veth) pair to establish connectivity between the sandbox and the outside network. It also configures the network settings to enable communication.
- **Namespace Isolation**: The program clones (forks) a new process while separating it from the host system by using Linux namespaces. This includes isolating the network, process, and mount namespaces to create a fully isolated environment.
- **Pivot_root and Execution**: The child process is assigned the root directory specified in the configuration file using pivot_root, ensuring that it operates within the sandbox environment. Necessary symlinks are then created inside the sandbox, and the root process specified in the configuration file is executed with the given command-line arguments.
- **Cleanup**: Once the root process completes execution, the parent process ensures that the sandbox's root directory is cleaned up, removing all files and directories recursively.

Jailor is essential for ensuring that the server processes run in a controlled and isolated environment.

7.3. Sdeamon

The sdeamon program is designed to manage sandboxes on virtual machines. It runs as a daemon process and listens on a specified TCP port for incoming commands. Its main responsibilities include:

- **Container Management**: Sdeamon can start and terminate sandboxes based on commands received from the smanager program.
- **Port Forwarding**: It sets up necessary port forwarding rules to ensure that the sandboxes can communicate with the external network, enabling the UDP servers inside the sandboxes to receive requests.
- **External Control**: By running as a daemon, sdeamon allows sandboxes to be managed remotely, making it possible to initiate or stop server processes as needed during the migration process.

Sdeamon serves as the interface between the virtual machines and the sandbox management commands, providing the necessary control for managing server processes within each VM.

7.4. SDN

The sdn component is responsible for managing incoming client requests and directing them to the appropriate server instances. It plays a vital role in ensuring seamless communication during the migration process. The key functions of sdn include:

- **Request Management**: It accepts UDP requests from clients and uses a lookup table to determine which virtual machine and container should handle each request.
- **Threaded Request Handling**: When a request is received, sdn creates a new thread to handle it, allowing multiple requests to be processed concurrently without blocking.
- **Dynamic Routing**: During migration, sdn will update its lookup table to direct client requests to the appropriate instance, ensuring that responses are returned to the clients correctly and without interruption.

The sdn component is crucial for managing network traffic during the migration, ensuring that requests are consistently routed to the correct server instance, even as the server process is being migrated.

7.5. Smanager

The smanager program acts as a user interface in the host system for managing the sandboxes within the virtual machines. It interacts with sdeamon to initiate or terminate server processes based on user input. The key features of smanager include:

- **User Interaction**: It prompts the user for actions such as starting or stopping server processes and sends the appropriate commands to sdeamon running inside the virtual machines.
- **Centralized Control**: By providing a centralized way to manage sandboxes, smanager simplifies the process of controlling multiple server instances across different VMs.

7.6. Server and Client

- **Server**: The server is a simple UDP server that listens for incoming numerical requests from clients. When it receives a number, it calculates the square of the number and sends the result back to the client. The server runs inside the sandbox environment, which is managed by jailor.
- **Client**: The client is a basic UDP client that sends numerical requests to the server and waits for the response. The client simulates real-world network traffic, providing a way to test the migration process.

7.7. Component Interaction and Communication

The components described above work together to achieve seamless container-based migration within Cloud-RAN. The configer program sets up the necessary configuration files, which are then used by jailor to create isolated sandbox environments.

Sdeamon manages these sandboxes within each virtual machine, while smanager provides centralized user control. The sdn component ensures that network requests are routed correctly during migration, and the server and client programs simulate network traffic to test the system's performance.

The communication between these components is designed to be efficient and reliable, with sdeamon and smanager enabling remote management, and sdn ensuring that the migration process does not disrupt ongoing communications. Together, these components form a cohesive system that allows for the live migration of server processes with minimal downtime, meeting the stringent requirements of 5G networks.

8. RESULTS

The project remains incomplete due to its inherent complexity and the time constraints of a single semester. Further research and practical experience since the midterm report have revealed that achieving zero downtime during live migration is highly challenging, particularly under heavy traffic conditions where memory change frequency is exceptionally high. While zero downtime appears infeasible, it is highly possible that downtime can be significantly minimized with tailored techniques and optimizations.

Despite the incomplete status, the project has provided valuable insights. The integration of CRIU into the custom containerization tool allowed us to experiment with process checkpointing and restoration. The research conducted on existing containerization tools like LXC/LXD has highlighted their reliance on CRIU for migration and their respective strengths and limitations. These findings pave the way for future advancements in container-based live migration solutions.

9. CONCLUSION

This final report outlines the progress and challenges faced during the implementation of a container-based migration system for Cloud-RAN. Throughout the project, we covered a big amount of domain research and practice, the development of a custom containerization tool, the utilization of CRIU within this containerization tool for process checkpointing and restoration, and the implementation of a simulation environment with UDP server-client applications and SDN to mimic real-world traffic.

However, the project remains incomplete. Key tasks, such as implementing an efficient live migration protocol that minimizes downtime, automating the migration process, and conducting extensive testing are still pending. The insights gained suggest that achieving zero downtime for live migration under high-frequency memory changes is unlikely, but significant downtime reduction is achievable with further research and experimentation.

One critical realization is that a project of this scale exceeds the scope of a single semester and requires a larger team. A minimum of two students, preferably more, is recommended to manage the workload effectively. Additionally, the field of containerization and live migration proved to be significantly more challenging than anticipated. Comprehensive research and hands-on practice are essential prerequisites for undertaking such a project.

In conclusion, while the project is incomplete, the foundational work and research conducted have laid the groundwork for future efforts. The lessons learned and the tools developed provide a solid base for advancing container-based live migration techniques tailored to Cloud-RAN and similar environments.

10. FUTURE WORK

To complete the project and achieve its goals, the following tasks must be addressed:

- **Advanced CRIU Utilization**: Conduct in-depth research on CRIU's C API[6] to leverage its full potential for managing migration processes programmatically. Also, its CLI is very flexible and it can be utilized in a bash script by leveraging its advanced command line arguments too[7] .
- **Implementation of Migration Techniques**: Develop and test migration protocols, including pre-copy, post-copy, hybrid approaches, and custom approaches if necessary. Modify these techniques to minimize downtime and adapt them to high-traffic scenarios.
- **Extensive Testing and Optimization**: Design realistic test scenarios to evaluate the developed migration techniques. Monitor metrics such as packet loss, latency, and downtime under varying traffic loads to analyze the solutions.
- **Comparison with Existing Tools**: Benchmark the implemented solution against existing containerization tools like Docker, Podman, and LXC/LXD along with their migration approaches to identify areas for improvement and validate its effectiveness.

By addressing these tasks, the project can achieve its objectives of developing a seamless live migration solution tailored to Cloud-RAN and similar latency-sensitive environments.

[6]`https://criu.org/C_API`
[7]`https://man.archlinux.org/man/criu.8.en`

REFERENCES

1. Taşdan, A. B. and S. Sevgican, "5G/CloudRAN", *Boğaziçi University, Computer Engineering Senior Projects*, 2018.

2. Ömer Cihan Benzer and Y. E. Topal, "Container-Founded 5G vRAN Network", *Boğaziçi University, Computer Engineering Senior Projects*, 2022, `https://github.com/ocebenzer/UDP-Simulation`.

3. Liang, X., "Container Migration: A Perfomance Evaluation Between MIGrror AND Pre-copy", *Western University, Electronic Thesis and Dissertation Repository*, 2024.

4. radiall.com, "What is RRU in Telecom?", , 2022, `https://www.radiall.com/insights/what-is-rru-in-telecom`.

5. Chabbouh, O., S. B. Rejeb, Z. Choukair and N. Agoulmine, "A novel cloud RAN architecture for 5G HetNets and QoS evaluation", *2016 International Symposium on Networks, Computers and Communications (ISNCC)*, pp. 1–6, 2016, `https://api.semanticscholar.org/CorpusID:21304415`.

APPENDIX A: DATA AVAILABILITY STATEMENT

Source code of the COMIC-RAN can be found in the repository:

https://github.com/Simurgan/comicran