Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

PhD Thesis

CPU Resource Allocation in SDN, SDR and MEC Integrated Networks

Author:
Beiran Chen

Supervisor:
Prof. Marco Ruffini

24th June 2025

(This Page Intentionally Left Blank)

Declaration

I declare that this thesis has not been submitted as an exercise for a degree at this or any other university and is entirely my own work.

I agree to deposit this thesis in the University's open access institutional repository or allow the Library to do so on my behalf, subject to Irish Copyright Legislation and Trinity College Library conditions of use and acknowledgement.

I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish (EU GDPR May 2018).

Signed:

Beiran Chen, 24th June 2025

(This Page Intentionally Left Blank)

Abstract

Software-Defined Networking (SDN) and Software-Defined Radio (SDR) are two key software-defined technologies for enabling flexible and open wired and wireless connections in next-generation networks. SDN separates the control plane from the data plane, allowing for centralized, programmable control over network functions, while SDR uses software to implement radio functions, providing adaptability in wireless communication systems. Network Function Virtualization (NFV) virtualizes network functions, making it possible to integrate SDN and SDR within the same cloud infrastructure, thus reducing the reliance on specialized hardware and enhancing scalability. Researchers have recently explored use cases for such integration, with cloud-based platforms like Mobile Edge Cloud (MEC) being one of these promising applications. MEC brings computation closer to end users by extending cloud capabilities to the edge, which helps meet the low-latency requirements of modern applications such as augmented reality, autonomous driving, and real-time data analytics.

Our investigation highlights that the cloud infrastructure used for SDN and SDR can also support MEC applications, offering an opportunity to integrate SDN, SDR, and MEC into a unified cloud infrastructure. Such integration allows for more dynamic and flexible resource utilization, leading to better performance, cost savings, and energy efficiency. In this thesis, we propose an integrated network architecture that leverages this integration, which, to the best of our knowledge, is a novel contribution to this research area.

Moreover, recent literature on SDN and SDR has primarily focused on the performance of communication aspects, such as bandwidth, delay, and packet loss, to meet the low-latency and high-bandwidth requirements of 5G and beyond. However, these studies often overlook computational resource requirements and related power consumption due to limited real data on computational resource usage in cloud environments and a lack of experimental testbeds. This thesis addresses this gap by focusing on CPU resource allocation

and energy-saving strategies for SDN, SDR, and MEC integration. We designed and conducted experiments using the Trinity College Dublin OpenIreland testbed with real SDN and SDR equipment to evaluate CPU and power consumption. Our results provide insights into computational resource savings when applying our approaches to an integrated MEC environment, examining the correlation between network bandwidth and CPU consumption for SDN and SDR. The findings are further applied to MEC use cases and experiments.

Additionally, we investigate algorithmic solutions for resource allocation in a shared SDN, SDR, and MEC structure. Existing literature mainly focuses on centralized control algorithms, where a central controller manages resource allocation for all elements in an MEC network. There is, therefore, a gap in distributed resource sharing and self-organization. This thesis focuses on distributed solutions for computational resource sharing through self-organization. Our approach shares spare resources between users to guarantee their satisfaction, employing a self-organization-based method for CPU resource sharing while considering users' personality traits. The resource-sharing steps are game-theory-based and utilize distributed solutions. Furthermore, we simulated real MEC use cases to demonstrate the benefits of our proposed algorithms, showing improvements in resource allocation and power savings over baseline algorithms.

In summary, this thesis covers the theoretical, simulation, and experimental investigation of CPU resource allocation for SDN, SDR, and MEC in a shared cloud-based environment. Our key contributions beyond the state-of-the-art are as follows: 1) We propose a softwarized network architecture that integrates the computation and processing of SDN, SDR, and MEC within a unified cloud-based infrastructure, allowing for enhanced flexibility and dynamic resource allocation. 2) We optimize CPU and power consumption for this integrated infrastructure, supported by both testbed experiments and simulation results, demonstrating significant improvements in resource utilization and energy efficiency compared to existing approaches.

Acknowledgements

Words cannot fully express my gratitude to my supervisor, Prof. Marco Ruffini. I could not have undertaken this journey without his invaluable support. From theory development and experiment setup to paper writing, his immense knowledge and extensive experience have been a constant source of encouragement throughout my academic research.

I would also like to thank my co-supervisor and collaborators, Prof. George Iosifidis, Prof. Mingming Liu, and Prof. Noel E. O'Connor, for their guidance and contributions to my publications.

This endeavor would not have been possible without the generous support of Dr. Yi Zhang, Dr. Diarmuid Collins, and Dr. Frank Slyne, whose technical expertise greatly assisted my studies.

I am also grateful to all the members of the CONNECT administrative staff, whose kind help and support made my time at TCD truly enjoyable.

Lastly, I want to express my heartfelt gratitude to my family. Their unwavering understanding and encouragement over the past few years made it possible for me to complete my studies.

(This Page Intentionally Left Blank)

Contents

(This Page Intentionally Left Blank)

List of Figures

List of Tables

(This Page Intentionally Left Blank)

1 Introduction

(This Page Intentionally Left Blank)

Introduction

1.1 Overview

In next-generation 5G and 6G networks, flexibility and programmability are crucial for building an integrated wireless and wired network architecture. These networks are driven by application needs, particularly mobile applications. Bandwidth- and computation-intensive applications, such as virtual reality (VR) and autonomous driving, demand substantial computational resources for both communication and processing. Mobile Edge Cloud (MEC) provides a suitable platform to meet these requirements by distributing computational resources from end users to the cloud. To enable high-bandwidth and low-latency communication for these applications, Open Radio Access Network (Open RAN) technology has emerged.

Virtualization technologies are extensively used for both networking and computing in Open RAN. Network virtualization enables network functions to be implemented in cloud-based data centers, reducing the need for specialized hardware and enhancing scalability. Similarly, the computational demands of mobile applications can be integrated into the same cloud architecture to improve cost efficiency and reduce energy consumption. In this thesis, we focus on integrating computational and communication resources within MEC to optimize performance and resource utilization.

In the communication domain, we investigate Software-Defined Networking (SDN), a key technology for Network Function Virtualization (NFV), and Software-Defined Radio (SDR), a key technology for Cloud Radio Access Network (C-RAN). Most recent literature primarily focuses on traditional networking performance metrics, such as throughput, signal-to-noise ratio, and latency, without thoroughly examining the computational resources required to

support SDN and SDR. Addressing this gap - how to optimize computational resources - forms a core part of this thesis. The Trinity College Dublin OpenIreland testbed provides the necessary hardware and software for our research, allowing us to design and implement experiments to explore these computational requirements.

This thesis proposes an integrated architecture involving SDN, SDR, and MEC, where cloud-based CPUs are shared across these network functions and services. This architecture allows for flexible allocation and control of CPU resources. Specifically, CPU resources are containerized, enabling them to be dynamically deployed and scaled based on application requirements. The CPU resources are divided into two categories: communication CPU resources and computation CPU resources. Communication resources handle tasks related to SDN and SDR components, ensuring efficient data transmission and network management, while computation resources are allocated to MEC applications, providing processing power for tasks such as VR and autonomous driving.

Figure 1.1 illustrates the integrated network architecture, including SDN and SDR for wired and wireless connections, and MEC for executing tasks generated from vehicular networks and other demanding applications. This thesis emphasizes the efficient integration of CPU resources for both communication (SDN and SDR) and computation (MEC), with the goal of minimizing computational resource usage and reducing power consumption.

Furthermore, we have developed simulations using use case applications of this integrated architecture. Most existing literature on MEC resource allocation focuses on optimizing task migration between end users and MEC, with task completion rate and completion time as the main optimization targets. However, with the introduction of this integrated softwarized architecture incorporating SDN and SDR, there is a research gap regarding the coordination of CPU resources for both network and computational functions. In this thesis, we address this gap by proposing a self-organizing algorithm for resource coordination. We present an end-to-end resource-sharing scheme that incorporates user preferences, considering interactions between users and the system. We employ a game-theory-based self-organization method to solve the CPU resource-sharing problem between end users, as well as between end users and the mobile edge cloud. Our optimization goals include minimizing overall CPU resource and energy consumption, while ensuring user satisfaction by executing tasks

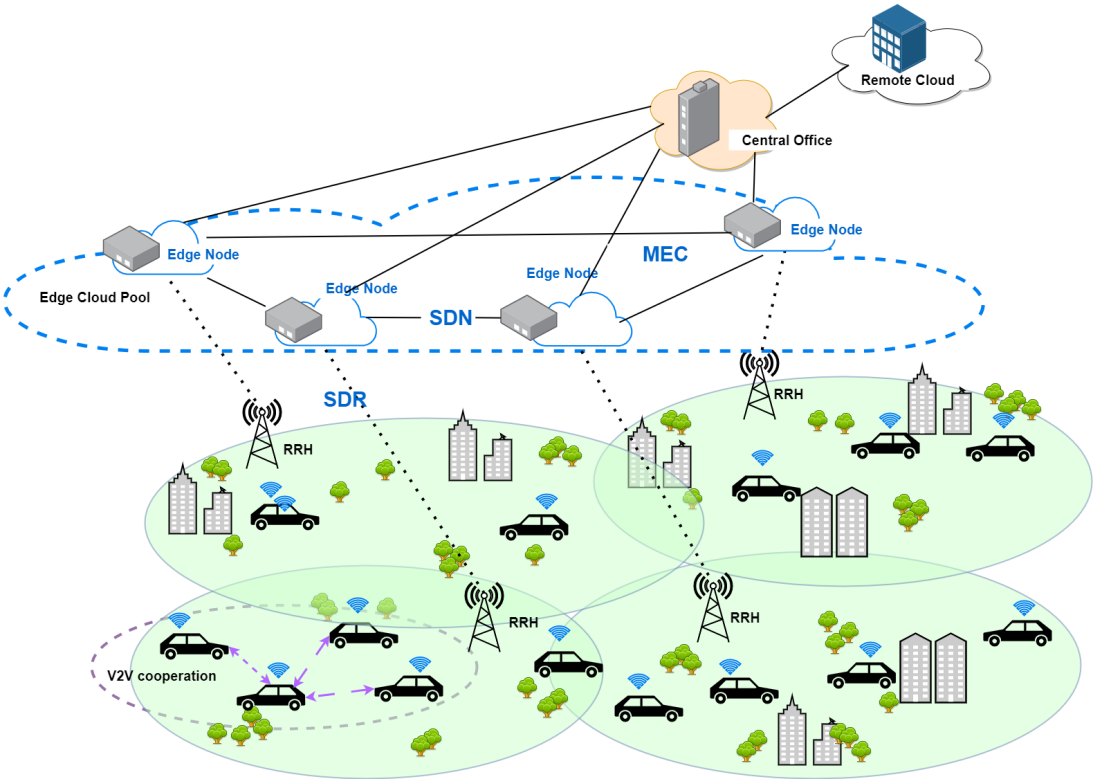


Figure 1.1: Proposed integrated MEC architecture.

within the required time constraints.

1.2 Research Problems Definition

Existing literature on MEC task offloading primarily focuses on task execution and resource allocation within the edge cloud. The communication aspect - such as wireless and wired communication supporting the MEC network architecture - is often treated as a constraint, impacting data rate and communication delay in task offloading optimization. In this thesis, we propose leveraging SDN and SDR for programmable wireless and wired networks, allowing the CPU processing of SDN and SDR to be integrated into the same edge server as MEC. This co-location presents opportunities for significant CPU resources and energy savings.

Moreover, this co-location and integration approach is novel and extends beyond the current state of the art. In addition, for resource sharing in this integrated architecture, we incorporate user behaviour and preferences - an aspect not considered in recent related

work.

After reviewing recent literature, we identified several research gaps, which led us to define and investigate the following research questions in this thesis:

- What are the benefits of integrating SDN, SDR, and MEC within the same cloud infrastructure?
- How to design and optimize a CPU resource-sharing scheme using this integrated network architecture?
- What is the relationship between network performance (e.g., bandwidth, data rate) and CPU utilization for SDN and SDR, and how does this affect CPU resource sharing?
- How do user behaviour and preferences influence CPU resource-sharing outcomes?
- How much energy can be saved through the integration of SDN, SDR, and cloud infrastructure with CPU resource sharing?

1.3 Our Contribution

As mentioned earlier, our work addresses the CPU resource allocation problem within the proposed architecture that integrates SDN, SDR, and MEC technologies through comprehensive theoretical analysis, experimental testbed evaluations, and simulations.

The contributions of this thesis can be summarized as follows:

1. We proposed an integrated SDN, SDR, and MEC architecture that shares CPU resources in the cloud for both networking/communication processing and task computation.

2. We developed a game-theory-based self-organization algorithm to optimize overall CPU and energy consumption within the integrated architecture, taking into account user preferences in the network.

3. Through testbed experiments, we demonstrated that SDN and SDR exhibit different CPU usage and processing parallelization patterns, leading to energy savings when integrated on the same cloud servers.

4. We showed that integrating SDN, SDR, and MEC in the same edge cloud infrastructure with our resource-sharing strategy reduces power consumption compared to a non-resource-sharing scheme in mobile edge cloud task-offloading scenarios.

1.4 Thesis Organization

This thesis is organized as follows:

Chapter 1 introduces our research motivation, research questions, and a summary of our contributions.

Chapter 2 provides a state-of-the-art review of related work. It explores the background and research efforts on SDN, SDR, and MEC integration, as well as MEC resource allocation. This chapter identifies research gaps and highlights our contributions beyond the existing state of the art. We also summarize the similarities and differences between related work and our research.

Chapter 3 presents our proposed self-organizing algorithm for CPU resource sharing. We incorporate the concept of Personality Traits to design a game-theory-based algorithm for resource sharing in the cloud system. Our main algorithm, EMP with Asymmetric Nash Bargaining strategy (EMP-A), is introduced, along with three other algorithms - EMP with Fixed step strategy (EMP-F), Single Evolution Multi-robots Personality (SEMP), and Nash Bargaining Solution Sharing (NBSS) - for benchmarking. We also compare our results with a related published algorithm, Cloud Cooperative-Federation Sharing (CCFS).

Chapter 4 discusses an energy-efficient approach to SDN and SDR joint adaptation of CPU utilization. Using testbed measurements with real cloud-based virtual machines, we analyze the power consumption of NFV processes and investigate CPU utilization and parallelization under different configurations. We also developed an energy-saving scheme based on the collected data.

Chapter 5 describes the simulation we implemented using EdgeCloudSim for SDN, SDR, and MEC integration. Leveraging the data collected in Chapter 4 and the algorithm from Chapter 3, we developed an energy-saving scheme and conducted a case study involving vehicular networks with task execution and offloading.

Chapter 6 concludes the thesis and discusses potential directions for future work.

1.5 Published Works

(1) **B. Chen**, F. Slyne and M. Ruffini. Energy Efficient SDN and SDR Joint Adaptation of CPU Utilization Based on Experimental Data Analytics. IEEE International Conference on Communications (ICC 2023). Available: https://arxiv.org/pdf/2302.01558.pdf

(2) **B. Chen** and M. Ruffini. Resource Cooperation in MEC and SDN based Vehicular Networks. To appear in IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC 2023). Available: https://arxiv.org/abs/2308.04564

(3) **B. Chen**, Y. Zhang, G. Iosifidis, and M. Liu, "Reinforcement Learning on Computational Resource Allocation of Cloud-based Wireless Networks," 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), Internet of Things (WF-IoT).June 2020:1-6. doi:10.1109/WF-IoT48130.2020.9221234

(4) **B. Chen**, M. Liu, Y. Zhang, Z. Chen, Y. Gu and Noel E. O'Connor, "An intelligent multi-speed advisory system using improved whale optimisation algorithm," In: 2021 IEEE 93rd Vehicular Technology Conference - Spring, 25 - 28 Apr 2021, Helsinki, Finland (Online). doi: 10.1109/VTC2021-Spring51267.2021.9449030

(5) **B. Chen**, Y. Zhang, and G. Iosifidis, "Resource Sharing in Public Cloud System with Evolutionary Multi-agent Artificial Swarm Intelligence," Service-Oriented Computing - ICSOC 2020 Workshops, December 2020. doi: 10.1007/978-3-030-76352-7_25

2 Background and Related Work

(This Page Intentionally Left Blank)

Background and Related work

This chapter presents an in-depth review of the background of SDN, SDR, and MEC, along with their integration, and includes a comprehensive literature review on CPU resource allocation algorithms. Throughout the literature review, we emphasize that our core contribution lies in the "*SDN, SDR, and MEC integration*" and that our research is fundamentally built upon this proposed network architecture. In this chapter We then summarize the technologies and mathematical models employed in related published studies, highlighting their limitations. Additionally, we outline the similarities and differences between the existing research and our work.

We organize this chapter as follows: Section 2.1 covers the background and related work on SDN and SDR integration. Section 2.2 discusses the background and related work on SDN and MEC integration. Section 2.3 focuses on related work concerning CPU resource allocation algorithms in MEC. Finally, Section 2.4 summarizes this chapter and highlights the contributions of our work in this thesis.

2.1 SDN and SDR integration

2.1.1 C-RAN Architecture and Virtualization

When discussing SDN and SDR integration, it is essential to consider the broader context of C-RAN and NFV. C-RAN (Centralized Radio Access Network) is a mobile network architecture where the Base-Band Units (BBUs) of a set of base stations are moved to the cloud from the Remote Radio Heads (RRHs). Thus, the entire Radio Access Network (RAN) processing is performed in a cloud/data centre environment. A generic C-RAN architecture consists of two main components: 1) the cellular site, which comprises antennas and RRHs

that convert wireless signals to and from digital baseband signals, and 2) the data centre, which houses a large number of BBUs responsible for RAN processing.

In C-RAN, the connection between BBUs and RRHs is known as the mobile fronthaul [1], which can be implemented using a wireless-optical integrated network. The baseband signals are transmitted over high-speed links (typically optical fibres) to the BBUs. The C-RAN approach eliminates the need for expensive on-site equipment and real estate, providing significant advantages such as reduced costs and advanced cooperative processing capabilities by incorporating NFV [2]. The centralization of BBUs offers the flexibility to control and manage virtualized resources efficiently.

The entire processing chain in C-RAN, from baseband processing to packet processing, is carried out in the cloud, where general-purpose servers are used, a mature and cost-optimized technology [3]. The use of General-Purpose Processors (GPPs) allows for a unified processing platform, eliminating the need for separate platforms for physical and other layers. The GPP platform breaks the boundary between cellular infrastructure providers and cellular network service providers, thereby elevating all resources to meet evolving demands.

SDR plays a critical role in C-RAN. By softwarizing baseband processing, SDR provides foundational technologies for functional splits, allowing part of the wireless signal processing to be offloaded to the C-RAN cloud. This approach offers significant potential for further integration in cloud environments.

We investigate Software Defined Radio (SDR) and Cloud Radio Access Network (C-RAN) technologies in our thesis by implementing multiple prototypes within our OpenIreland testbed, which has been established at Trinity College Dublin. These prototypes aim to demonstrate the practical aspects and challenges of deploying SDR and C-RAN in a cloud-based environment, highlighting the critical technical aspects involved in achieving optimal performance in such settings. To facilitate these experiments, an OpenStack-based Cloud infrastructure [4] has been meticulously deployed in our testbed, which not only supports the virtualization requirements but also provides us with various opportunities to experiment with and fine-tune CPU resource allocations in a software-defined virtualization architecture. This cloud-based environment is ideal for experimenting with virtualized network functions, dynamic resource management, and scalability, which are critical components of

modern telecommunications systems.

Furthermore, open-source SDR software platforms such as OpenAirInterface [5] and srsRAN [6] have been leveraged to implement 4G and 5G programmable network solutions within our testbed. These platforms enable the development, deployment, and evaluation of software-defined mobile base stations and core network elements in a cloud computing environment that is driven by CPU resources rather than specialized hardware. By utilizing these tools, we were able to conduct extensive tests on both the performance and scalability of virtualized mobile networks, examining the potential of software-based implementations for future telecommunications.

Our research focuses on the practical implementation of these open-source SDR solutions to create a realistic cloud-based mobile networking environment. In doing so, we aim to bridge the gap between theoretical research on SDR and C-RAN technologies and their real-world deployment challenges. The OpenStack-based virtualization infrastructure allows us to simulate realistic scenarios by dynamically adjusting CPU resource allocation to meet the performance requirements of different network functions, enabling us to analyze the impact of resource contention and other factors on the quality of service (QoS) delivered by SDR systems. This experimental setup provides valuable insights into the effectiveness of using commodity hardware to support mobile network infrastructure and highlights the benefits and trade-offs of cloud-based RAN deployments.

Moreover, our testbed's flexibility enables us to explore various configurations and optimizations, allowing us to investigate the feasibility of integrating SDR and C-RAN into future 5G and beyond networks. This includes studying aspects such as network slicing, multi-tenant environments, and efficient resource utilization. By employing open-source software and an adaptable cloud platform, we contribute to the growing body of research that seeks to understand and advance the adoption of virtualized, software-defined networking technologies in the telecommunications domain.

2.1.2 NFV and SDN as Enabling Technologies

NFV addresses the virtualization of network equipment, including routers, switches, and other network processing devices. NFV aims to reduce costs and power consumption while

facilitating resource sharing across operators [7]. The European Telecommunications Standards Institute (ETSI) has presented standardized NFV for wireless virtualization [8], focusing primarily on core network functions and enabling networking functions to be implemented as virtualized entities. While NFV for C-RAN is not yet standardized, it shows critical benefits due to its flexibility. ETSI, in its white paper No. 23 [9], indicates that NFV can be an effective tool for implementing the collocation of C-RAN.

As one of the main enabling technologies of NFV, SDN provides softwarization and the decoupling of the control plane and data plane. This capability helps virtualize network functions, enabling network slicing and orchestration across multiple server providers.

In reviewing the enabling technologies and related work on SDN, SDR, and MEC integration (which is the primary focus of this thesis), it is clear that SDN and SDR integration has received increasing attention from the research community.

For example, authors in [10] adopt SDR, SDN, and NFV to enable flexibility and efficiency in various operational areas, such as in-network processing and data storage, fine-grained radio control and monitoring, distributed enforcement of QoS, and network resource management. This work utilizes the concept of Mobile Network Virtual Operators (MNVO) and network slicing for virtual network functions, providing inspiration for SDN and SDR network architecture integration.

Authors in [11] study the provision of NFV technologies in the Internet of Things (IoT). They propose an architecture for an IoT network based on virtualization technologies, suggesting the virtualization of full-stack IoT functions using generic devices, including radio processing with SDR and network functions with SDN and NFV concepts. This architecture provides more flexible and reconfigurable solutions for deploying customized on-demand virtualized IoT services.

This thesis focuses on analyzing the CPU resource consumption of virtualized Software-Defined Networking (SDN) functions by integrating some of the most popular SDN controllers currently used in the research community, such as RYU [12] and OpenDaylight [13], into our testbed environment. By employing these controllers, we aim to explore their effectiveness and efficiency in a cloud-based network setting, particularly in terms of resource utilization and scalability.

In our approach, the control plane and data plane of the SDN architecture are separately virtualized into distinct cloud-based Virtual Machines (VMs). This separation allows us to accurately measure and analyze the CPU and power consumption of each plane independently. By isolating these functions, we gain a clearer understanding of their individual performance characteristics and resource requirements, which is crucial for optimizing SDN deployments in cloud environments. Such detailed measurements provide insights into the challenges of resource allocation and the performance trade-offs involved, ultimately helping to identify strategies for improving efficiency and reducing overhead in virtualized SDN networks. Moreover, the virtualization of the control and data planes enables us to simulate realistic network scenarios with varying traffic loads and resource constraints. This allows us to assess the scalability and reliability of the SDN controllers under different conditions, providing valuable information for designing more robust and efficient cloud-based SDN solutions. By thoroughly analyzing the CPU resource consumption and exploring optimization opportunities, this research aims to contribute to the development of best practices for deploying SDN in modern cloud environments, particularly for emerging applications such as 5G networks and beyond.

2.1.3 SDN and SDR Integration for 4G and 5G Networks

SDN and SDR integration has received increasing attention in the context of 4G and 5G networks. The authors in [14] propose using NFV to virtualize SDN and SDR in the network architecture for 4G and 5G wireless networks. This work provides an overview of standardization efforts for SDN and SDR by various organizations, discusses future technological challenges, and examines key research issues. Authors in [15] propose an SDR and SDN integration framework with cloud-based control, monitoring, and management functions, simulating the end-to-end delay of networks. Their results provide insights into how such integration can be beneficial for delay-sensitive applications, such as MEC task execution in vehicular networks, which is a focus of this thesis.

Authors in [16] introduce an SDN/NFV-based SDR by integrating SDR and SDN for Sparse Code Multiple Access (SCMA) using asymmetric encryption to enhance security. This work focuses on physical layer transmission. Authors in [17] discuss SDN and SDR

integration using functional split and a virtualized Remote Aggregation Unit (RAU), proposing a master-slave SDN controller architecture with the integration of a virtualized BBU from SDR. This work is a conceptual proposal for an SDN-SDR integrated controller aimed at guiding future research in this area.

Authors in [18] implement C-RAN with a separated control plane, centralized unit user plane, and distributed unit networking controls. They decouple the control and data planes using SDN, focusing on networking performance (throughput, delay) with DPDK integration. We adopt this concept but focus on CPU resource consumption. Authors in [19] present a method for integrating SDR into network environments using containerization, minimizing CPU consumption overheads and setup latency. This work, which focuses on network management to control SDR and wireless network performance, provides a theoretical foundation for our research on CPU and power consumption for SDN and SDR integration.

In this thesis, we adopt the concept of integrating SDN and Software-Defined Radio (SDR) to create a more flexible and efficient network architecture. By separating the control plane from the data plane, we propose two distinct layers of integration that work in harmony to achieve seamless network control and operation.

The first layer, the integrated control plane, includes both the SDN controller and the wireless controller for the SDR. This integrated control plane is responsible for managing and orchestrating the overall network operations, providing a centralized and unified point for controlling both wired and wireless network components. With the integrated control plane, we achieve a holistic approach to network management, allowing for dynamic control of not only the wired infrastructure but also the radio resources. This integration brings increased adaptability and scalability, which is particularly important for next-generation networks such as 5G, where multiple types of network components need to operate in synergy.

The second layer, the integrated data plane, consists of SDN switches, SDR Baseband Units (BBUs), and various wireless networking components. This layer is responsible for the actual forwarding of data and the physical transmission of signals across the network. By integrating SDN switches with SDR BBUs, we create a programmable data plane that can dynamically adapt to changing network conditions and requirements. The combination of

these components allows for efficient handling of network traffic, enabling enhanced quality of service (QoS) and optimal resource allocation. The SDR BBUs handle the signal processing tasks while SDN switches manage data flow, providing a flexible and programmable environment that is essential for supporting diverse applications and services in modern telecommunications networks.

2.1.4 Experimental and Testbed Studies

In addition to these theoretical studies, there are experimental/testbed-based studies on resource allocation that are similar to our work in this thesis. SDN experiments can be conducted using simulators, emulators, or actual testbeds, often implemented using the OpenFlow protocol [20]. The most popular emulator is Mininet, which uses OpenFlow switches to conduct SDN experiments on a single PC. EstiNet is another popular emulator; authors in [21] present several benchmarks for EstiNet.

Authors in [22] designed a service using the XenServer hypervisor OpenStack platform and OpenDayLight SDN controller, demonstrating capabilities to dynamically adjust resources. Their experiment results showed that CPU resource usage increased linearly with the number of switches, CPUs, and memory. In our experiments, we used both OpenDayLight and Ryu controllers for SDN.

Authors in [23] propose a Network Hypervisor (NH) for efficient CPU resource provisioning with performance guarantees. They take three steps to achieve their goal: (a) determine the minimum CPU resources needed, (b) reveal the key virtual network properties affecting CPU utilization, and (c) design a CPU usage prediction model. Their experiments, using Mininet to emulate physical infrastructure and running tenants controlled by Ryu [12], assess six topologies to predict CPU resource requirements for an NH.

Authors in [24] evaluated SDN performance with different network resource allocations (e.g., bandwidth, CPU). They used Mininet to conduct experiments on three topologies with the same controller and the same number of hosts, employing the "iperf" tool to measure data transmission. We also use this tool in our testbed experiments.

Authors in [25] built five topologies and used Mininet to test various SDN experimental platforms, measuring metrics such as topology setup and destruction time, CPU usage

during setup and experiments, RAM demand, ping delays, and resource sharing fairness. The results show that (a) setup time was highly affected by the number of switches, (b) available RAM influenced Mininet's RAM usage, (c) failed ping packets increased with the number of links, and (d) load balancing between CPU cores improved as the number of nodes increased. From this experimental setup, we learned that varying the number of switches and topologies is crucial for obtaining realistic results in SDN CPU utilization tests.

All of these studies used Mininet to investigate network performance across diverse topologies. In our experiments, we also use Mininet but separate SDN switches and controllers, measuring CPU utilization for controllers and switches independently.

For SDR experiments, we reviewed studies investigating power consumption in legacy base stations (BSs), mainly focusing on RF output, power amplifiers, and baseband processing. Authors in [26] examine the effect of bandwidth, while authors in [27] study packet length effects on CPU power consumption under different SDR settings.

Some works on virtualized base stations (vBSs) have also been conducted. Authors in [28] presented an experimental study of vBS power consumption, investigating its relationship with performance. Authors in [29] examined energy consumption in vBSs, showing the complex relationship between power consumption, performance, and vBS control policies using a srsLTE-based vBS testbed [30]. They proposed an online learning framework for balancing power consumption and performance and maximizing performance in power-constrained vBS scenarios.

Authors in [1] proposed a dynamic method for allocating processing resources based on NFV technologies in C-RAN. They virtualized BBU resources for LTE mobile networks into a BBU pool using Linux Container technology, deploying their experiments on a real testbed with SDR-based LTE functionality. In our work, we also use Linux Containers to host SDR processing, allowing flexibility in adding or reducing CPU resources for the containers.

Figure 2.1 is the block diagram illustrating the integration of SDN and SDR, including the key function blocks for each technology. The SDN components (SDN Controller, SDN Applications, and SDN Data Plane, and APIs) are shown on the left, while the SDR components (SDR Applications, SDR Layers, and SDR RF Front End) are on the right. The arrows indicate the flow of information and integration between these blocks, highlighting

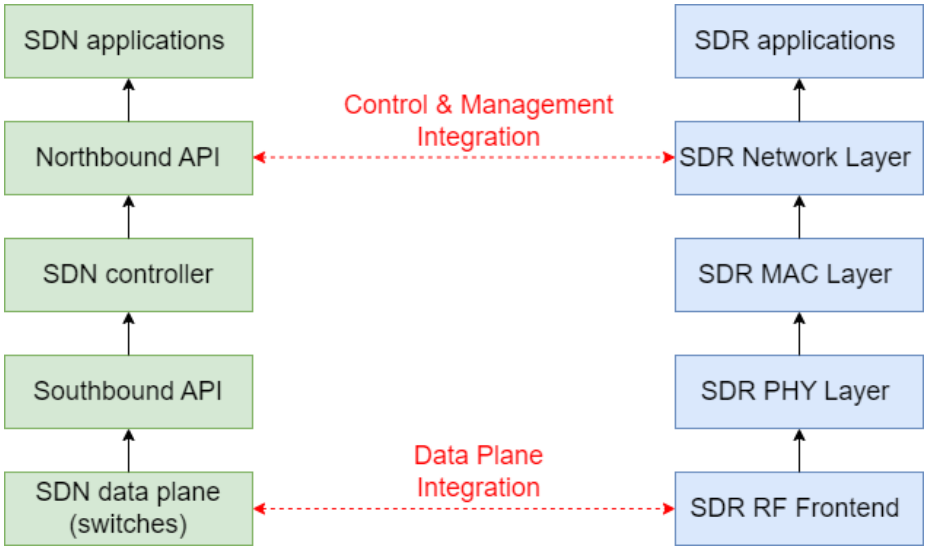


Figure 2.1: SDN and SDR integration block diagram

their interaction.

As mentioned earlier, SDR and SDN are important for next-generation wireless and wired networks, as they provide unprecedented flexibility in capacity allocation and service differentiation [31]. However, SDR and SDN are conventionally deployed separately (i.e., in different servers), leading to suboptimal resource utilization. The aforementioned experiments studied SDR or SDN separately and did not examine SDN and SDR together in terms of CPU utilization or power consumption.

In this thesis, we explore the CPU resource allocation of an experimental setup that deploys both SDR and SDN on the same limited CPU resources in the cloud. We analyze the power consumption of SDN and SDR, proposing power-saving methodologies based on their coexistence on the same group of CPU cores. We conduct extensive testbed measurements and collect data from a real cloud-based system for SDN and SDR in the Trinity College Dublin OpenIreland testbed [32].

2.2 SDN and MEC integration

2.2.1 Mobile Edge Cloud and Task Offloading

Cloud computing is a mature technology in the modern technology world, providing scalable and flexible resources for both businesses and individuals. There are many available

commercial cloud computing services, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure, which offer diverse solutions for computing, storage, and networking. These services have significantly reduced the need for on-premises infrastructure, providing pay-as-you-go models that help reduce costs for users while enabling access to vast computational power.

Mobile device users or vehicular network users can offload computation-intensive tasks to a remote cloud (i,e. the abovementioned commercial cloud services) to benefit from the cloud's large processing capacity and storage capabilities. This offloading can effectively reduce local computational time and power consumption, leading to prolonged battery life and improved device performance [33]. However, remote cloud offloading can sometimes result in higher latency due to the physical distance between users and cloud data centres, making it less suitable for applications with strict real-time requirements.

In addition to remote cloud solutions, Mobile Edge Computing (MEC) has emerged as an attractive alternative for users, offering edge services that are much closer to the end-users compared to traditional cloud computing. MEC provides the opportunity to offload tasks to servers at the edge of the network, which can significantly reduce task transmission delay on backhaul links [34, 35]. By bringing computational resources closer to the user, MEC can support latency-sensitive applications and enhance the overall user experience, especially in environments where rapid responses are critical.

MEC offloading is particularly well-suited for intelligent transportation use cases. The increasing growth of connected vehicles has led to an explosion in the number and variety of computing requests required for intelligent transportation systems. This surge in demand has put considerable pressure on data processing, as many intelligent transportation tasks have stringent latency requirements. Therefore, ensuring efficient processing and low latency is critical for vehicular systems to operate effectively. Network Function Virtualization (NFV) offers a promising solution to address these challenges by utilizing virtualization technology to flexibly program service functionalities. These software instances, known as Virtual Network Functions (VNFs), can be deployed at edge servers without incurring significant additional costs [36].

As previously mentioned, this thesis leverages the concept of Network Function Virtu-

alization (NFV) for the virtualization of network functions, enabling a more flexible and efficient infrastructure. One of the most prominent use cases in this area of research is Mobile Edge Cloud (MEC) task offloading. MEC task offloading is increasingly popular due to its ability to reduce latency and enhance the overall performance of network services by bringing computation closer to the network edge.

Task offloading, however, is a complex and non-trivial problem that involves several intricate processes. It requires the division and parallelization of tasks during the computational process, which enables the distribution of workload across multiple edge nodes to achieve higher efficiency. In addition, it involves the seamless downloading and uploading of task data, results, and intermediate states as part of the communication process between mobile devices, edge servers, and potentially even the cloud. This interplay of data exchange and computation makes task offloading a unique challenge, as it combines both network data transmission and computational task execution into a single coordinated effort.

To address these challenges, our research focuses on optimizing the split and parallelization of tasks to ensure efficient resource usage and minimal latency. We consider various parameters such as network conditions, available computational resources, and the nature of the tasks themselves. By doing so, we aim to identify the most effective strategies for task allocation and offloading, taking into account the trade-offs between processing time, energy consumption, and network resource usage. Furthermore, we explore different approaches to enhance the cooperation between edge servers and cloud infrastructure to improve the scalability and robustness of MEC deployments.

2.2.2 NFV and SDN Integration with MEC

By integrating NFV with MEC, computation-oriented service provisioning at the network edge can be realized, offering a flexible and cost-effective way to support applications with varying computational needs. This integration allows for the rapid deployment of new services and the dynamic scaling of resources in response to changing demand. Furthermore, combining SDN with NFV has the potential to achieve network-level resource allocation and flexible service provisioning. This integration enhances the performance of network systems in terms of end-to-end quality-of-service (QoS) guarantees and enables service customization,

which is crucial for meeting the diverse requirements of modern network applications [37].

The authors in [37] conducted a comprehensive survey of NFV technologies in transportation scenarios and applications, highlighting the role of NFV in enabling efficient and scalable service deployment. Some studies have presented different Internet of Vehicles (IoV)-integrated architectures that combine MEC/cloud computing with SDN/NFV technologies [38, 39]. These integrated architectures aim to provide a holistic approach to managing the computing and networking needs of vehicular systems.

Authors in [40] conduct an in-depth investigation of ETSI and 3GPP standards for SDN and MEC, respectively, and propose an SDN-based MEC framework that complies with both ETSI and 3GPP architectures. A testbed experiment with OpenAirInterface eNodeB and MEC servers is established, and an SDN controller is integrated into the framework. Low latency is achieved as the goal of the experimental setup. This paper is one of the first to involve an experimental prototype with SDN and MEC integration, which guides us in establishing our testbed, although we use more flexible open-source SDR software with srsLTE [41] in our setup.

Authors in [42] propose a new architecture called Edge-based SDN-enabled 5G network (ES-5G). This work aims to develop a mechanism to facilitate seamless handover procedures by pre-establishing forwarding paths. An OpenFlow controller and SDN switches are used to support handover routing, and end-to-end latency is evaluated. Although this work is not directly related to MEC, the authors mention that MEC could benefit from this framework's low-latency features, improving task offloading performance.

Authors in [43] propose an SDN-enhanced MEC architecture to facilitate the management of Mobile Edge Hosts (MEH). They implement a multi-host shared persistent storage concept using Docker containers. The authors integrate features of Mininet-WiFi and Containernet to evaluate the proposed architecture experimentally. Latency and bandwidth are measured in the experiment. This work demonstrates that SDN-enhanced signalling and control bring better latency and bandwidth performance, thus proving the effectiveness of integrating MEC and SDN.

Authors in [44] evaluate SDN technology applicability at MEC by adopting OpenFlow switches in MEC, focusing on data plane switch size and control plane latency performance.

In addition to these measurements, our work measures on CPU utilization as well.

In summary, this thesis adopts the integration of NFV, SDN and MEC to create a comprehensive and cohesive network architecture. This integration facilitates the virtualization of network functions and establishes a highly adaptable framework for dynamic task offloading and efficient resource management. By leveraging the combined strengths of NFV, SDN, and MEC, we are able to address the diverse and evolving requirements of modern applications that demand low latency, high reliability, and scalability.

The integration of these technologies allows to support a wide range of use cases, such as augmented reality (AR), autonomous driving, industrial automation, and other emerging technologies that require near-instantaneous data processing and seamless connectivity. By virtualizing network functions, we reduce the dependency on specialized hardware, enabling greater flexibility and scalability. SDN plays a crucial role by providing centralized control and programmability, allowing for dynamic network reconfiguration and optimized routing of data flows. MEC brings computational capabilities closer to the end users, minimizing latency and improving the overall quality of experience for latency-sensitive applications.

2.2.3 SDN for MEC Task Offloading

To enhance the communication networking aspect of MEC task offloading, SDN can be employed to tailor communication bandwidth and latency according to the needs of MEC task execution. SDN's centralized control and programmability allow for dynamic optimization of network resources, ensuring that MEC tasks receive the appropriate quality of service to meet application requirements. By leveraging SDN, CPU and processing resources, as well as power consumption, can be effectively managed and optimized, which is the key research question we investigate in this thesis. Our focus is on how SDN, MEC, and NFV can be integrated to create a more efficient and sustainable vehicular network system, with particular emphasis on minimizing power consumption and maximizing resource utilization.

Some recent works have been published related to this integration. Authors in [45] propose a Mobile Edge Computing Framework on SDN-based architecture, utilized in Vehicular Network scenarios for 5G Services. This work also proposes a fog computing framework based on fog computing as well. The paper focuses on a conceptual discussion about the

pros and cons of these two frameworks and identifies research challenges in this area.

Authors in [46] focus on security when integrating SDN into MEC task offloading. A secure service offloading method, named SOME, is designed and implemented by the authors. A technique called locality-sensitive hashing (LSH) is realized to achieve private and secure services. Performance indicators like load balancing and the success rate of task offloading are proven to be effective as well. In this thesis, our focus is on CPU resource and power savings; therefore, the overhead of making secure connections can be avoided. Additionally, we aim to consolidate loads on CPUs to shut down idle CPUs for power savings rather than balance the load, which differs from the target in this paper.

Authors in [47] aim to enhance service quality and optimize overall resource utilization by using a centralized control module (an SDN controller) to gather extensive network information, including vehicle density, mobility, location, traffic load, and resource allocation policies. With this information, similarly, the SDN controller we design is able to make network-level decisions for access control, resource allocation, and routing.

Authors in [48] propose an SDN-based computation offloading method for MEC, optimizing the overall response time using Mixed Integer Linear Programming (MILP) to enhance task execution. In contrast, our work focuses on overall CPU consumption and power consumption, including SDR integration, providing a broader perspective on energy efficiency alongside computational metrics.

Authors in [38] propose a conceptual SDN and MEC integrated framework for urban intelligent transportation networks, simulating latency, reliability, and throughput to evaluate network performance. In addition, authors in [49] propose an SDN-based control scheme with MEC offloading, named Lifetime-Based Network State Routing (LT-NSR). These two works focus on maximizing throughput and minimising network latency, whereas our study aims to minimize CPU consumption and power consumption, allowing us to analyze the impact of integration on resource efficiency and offering novel insights into the trade-offs between performance metrics and energy use.

The authors in [50] propose a three-tier Edge Computation (EC) framework that provides elastic processing resources and dynamic routing to edge servers based on real-time vehicle monitoring. They integrate EC and SDN to develop a software-defined networking edge

framework (SDN-EC) for vehicular network resource allocation. A vehicular network with Vehicle-to-Vehicle (V2V) communication is included in their use case, similar to our use cases in this thesis. However, the paper does not investigate the benefits and resource overhead of the SDN control plane functions for the framework, which we address in this thesis.

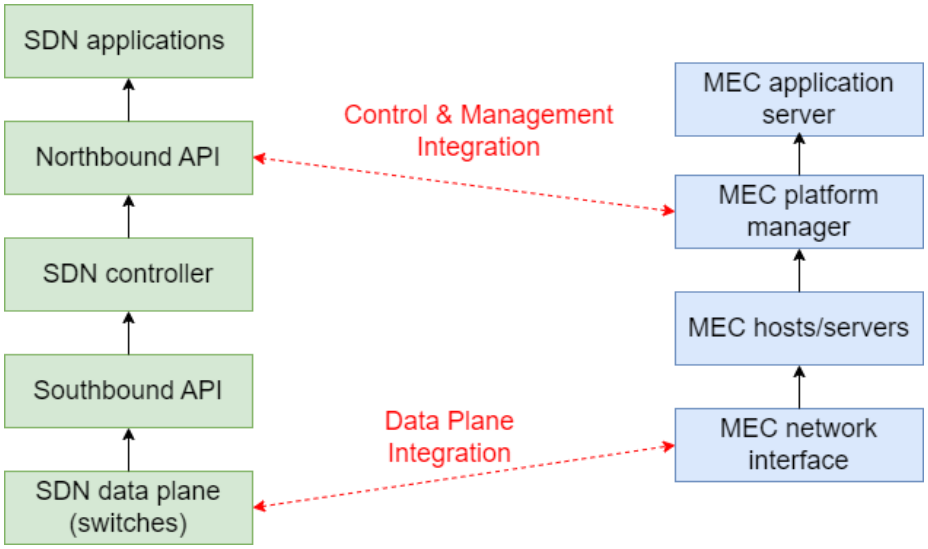


Figure 2.2: SDN and MEC integration block diagram

In summary, the integration of SDN and MEC in the related work provides additional control and management flexibility to the network infrastructure. This integration allows for the decoupling of the control plane and data plane, as well as the implementation of network slicing to create multiple virtual networks over a shared physical infrastructure. These capabilities are crucial for meeting the diverse needs of modern network applications, especially those requiring customizable and low-latency services. Figure 2.2 illustrates the block diagram of SDN and MEC integration, highlighting the key functional blocks for both SDN and MEC, including control and data processing units.

In addition, our research in this thesis builds upon these foundational concepts by integrating Software Defined Radio (SDR) into the SDN-MEC framework, thereby extending its capabilities to include dynamic radio resource management. This integration allows us to explore how SDR impacts various performance metrics, such as CPU usage, power consumption, and the overall efficiency of both the control and data planes. By incorporating SDR, we add an extra layer of flexibility to the network, allowing for programmable control

of the radio interface, which is essential for enabling adaptable and efficient communication in next-generation wireless networks.

Moreover, our research provides a unique contribution to the field by addressing an important gap in understanding the interplay between SDR and SDN in MEC systems. Specifically, we investigate how the integration of SDR affects the separation of control and data planes. By examining these aspects, our work aims to provide a deeper understanding of the potential benefits and challenges associated with using SDR in conjunction with SDN and MEC, ultimately contributing to the development of more adaptable, efficient, and scalable network architectures for future telecommunications systems.

2.3 CPU resource allocation algorithms in MEC

2.3.1 Self-Organizing Multi-Robot Systems

Self-organizing multi-robot systems are decentralized systems designed to optimize group-based user behaviors [51]. In these systems, robots collaborate without a centralized controller, relying instead on local interactions and rules to achieve a desired group outcome. The concept of personality updates and their impact on user collaboration has been extensively explored in this context, highlighting the significance of adaptive behaviours and dynamic role assignments in optimizing performance.

For example, the authors in [51] used personality attributes to develop a self-organized algorithm to address a problem involving multiple robots leaving a room in a coordinated manner. Their approach demonstrated how personality traits such as cooperation, persistence, and adaptability could influence the group's success in completing a shared task. This research illustrated the power of personality-driven coordination in achieving efficient group behaviours without relying on a centralized control mechanism.

Similarly, the authors in [52] introduced a game-theory-based approach for motive robots, which enabled them to collaborate effectively in pursuing a target. By applying principles of game theory, the robots were able to make strategic decisions based on the actions of other agents, thereby maximizing their chances of success. This work demonstrated the

potential of game-theory-inspired mechanisms to enhance collaboration and adaptability in decentralized multi-robot systems. The authors in [53] explored the evolution of control for a group of robots, focusing particularly on the management of signalling and connectivity for effective communication. By evolving control strategies, they were able to enhance the robots' ability to maintain connectivity and adapt their signalling behaviours to changing environmental conditions. This research highlighted the importance of flexible control mechanisms that can evolve over time to improve the efficiency and robustness of multi-robot systems.

Furthermore, the authors in [54] employed an artificial intelligence-based platform to measure the personalities of human groups, which provided insights into how individual traits impact group dynamics and collaboration. By quantifying personality attributes and analyzing their effect on group interactions, they were able to identify key factors that contribute to effective teamwork. This study provided a valuable perspective on the role of personality in group-based systems, further emphasizing the importance of adaptive and context-aware behaviours.

Despite these advances in self-organization concepts and their successful application to multi-robot systems, there remains a significant gap in the literature regarding the application of these methods to resource allocation in cloud-based systems. Specifically, limited research has explored how self-organizing principles can be used to optimize resource distribution in cloud environments, where tasks must be allocated dynamically and efficiently to meet evolving user demands. Cloud resource allocation is inherently complex, involving diverse workloads, fluctuating resource availability, and the requirement to maintain quality of service (QoS) under constantly changing conditions. This gap presents an opportunity to adapt successful techniques from self-organizing multi-robot systems to improve the adaptability, efficiency, and resilience of resource management in cloud-based environments.

In this thesis, we aim to fill this gap by applying self-organization methods to the MEC resource allocation problem. By leveraging the principles of self-organization, we seek to develop a framework that enables distributed and adaptive resource allocation in MEC systems, leading to more efficient utilization of computational resources and improved service delivery. MEC, as an extension of cloud computing at the network edge, introduces

additional challenges, such as the need for ultra-low latency, localized processing, and the management of heterogeneous devices. These challenges necessitate innovative and flexible approaches to resource management that can operate effectively in a decentralized and dynamic environment.

Our approach will draw inspiration from the successes of multi-robot systems, adapting these concepts to address the unique challenges of managing resources in a cloud-based, edge computing context. Specifically, we will explore how decentralized decision-making, adaptive behaviour, and local interactions can be applied to create a robust, self-organizing resource allocation strategy for MEC. By utilizing these principles, we aim to build a system that can autonomously adjust to workload changes, ensure optimal resource utilization, and maintain service quality without relying on centralized coordination. Our research aims to significantly contribute to the advancement of self-organizing systems in new domains, ultimately enhancing the scalability, flexibility, and robustness of MEC resource management. By extending the application of self-organization to cloud and edge computing, we hope to provide a novel solution that addresses the critical challenges faced by modern computing infrastructure.

2.3.2 Personality Updates and Game Theory in Resource Sharing

In this thesis, we introduce the concept of personality updates to solve resource-sharing challenges in cloud systems. Additionally, we employ game theory to define rewards for users, allowing them to make optimal decisions in a decentralized system. This approach presents opportunities for researchers to optimize resource allocation in cloud environments. Our approach leverages self-organizing algorithms, which enable individual users to update their preferences dynamically based on the environment and the actions of other users. This results in a more flexible and adaptive resource allocation process, particularly well-suited for distributed and highly dynamic cloud systems.

Game theory has been extensively employed in the literature to distribute optimization tasks among users in a decentralized manner. Similarly, in our thesis, we utilize game theory to facilitate distributed optimization, allowing individual users to make optimal decisions while contributing to overall system efficiency. For example, authors in [55] propose an

optimization framework that allocates MEC computational resources with the aim of minimizing power consumption while satisfying latency constraints. The authors use a convex optimization model combined with game-theoretic approaches to solve the resource allocation problem for multiple users in a multi-cell environment. The model optimizes resource usage by considering user-specific demands and network conditions, thus providing an efficient balance between energy savings and system performance. Authors in [56] introduce a Stackelberg game-based approach for multi-resource allocation and pricing in MEC systems. The authors model the interaction between service providers and users as a leader-follower game where service providers set the pricing levels for computational resources, and users respond by deciding their offloading strategies. The optimization problem is solved using game theory concepts to ensure an equilibrium between resource pricing and user satisfaction. The proposed model not only addresses resource allocation but also provides an economic mechanism for fair resource usage in edge computing environments. Authors in [57] investigate computation offloading and resource allocation for vehicular networks using a game theory approach. The problem is formulated as a mixed-integer programming model to minimize the total latency by jointly optimizing offloading decisions and allocating CPU resources. Game theory is applied to manage competitive relationships among vehicles for offloading strategies. The solution involves iteratively solving sub-problems, i.e., offloading strategy and CPU allocation until reaching Nash Equilibrium. This approach enables an efficient distribution of CPU resources among multiple tasks while ensuring latency constraints are met. Authors in [58] propose a game theory-based approach to jointly optimize task offloading decisions and resource allocation in MEC systems. The method treats offloading decisions, CPU capacity adjustment, and power control as parts of a game. The model aims to minimize energy consumption and latency while managing inter-cell interference of the mobile network. Each user optimizes their utility using a best-response approach, resulting in a Pareto-efficient Nash equilibrium. The paper also investigates the algorithm's convergence, computational complexity, and Price of Anarchy, showing its effectiveness in improving system performance through simulation analysis.

The models proposed in our thesis integrate game theory concepts for negotiation and apply them within a self-organizing multi-robot framework. By incorporating these con-

cepts, our approach facilitates efficient resource sharing and dynamic coordination among users. Specifically, we leverage game theory to guide negotiation and cooperation, allowing individual users to make decisions that benefit both themselves and the overall system, while embedding this mechanism into the self-organizing structure. This dual approach ensures that tasks and resources are allocated in an optimized manner, ultimately leading to increased overall user satisfaction and system efficiency.

The integration of game theory into the self-organizing framework enables each user to participate in a negotiation process where decisions are made based on their preferences, resource availability, and the actions of other users. By employing concepts such as Nash equilibrium and cooperative bargaining, the proposed model ensures fair resource distribution and conflict resolution in a decentralized environment. This enhances resource utilization and promotes fairness and adaptability, which are crucial for managing dynamic and heterogeneous network conditions. The integration of game theory and self-organization provides a decentralized yet cohesive strategy for achieving efficient resource allocation, optimizing overall system performance, and improving user experience. This represents one of the key contributions of this thesis, as it offers a novel approach to managing complex systems where centralized control is either impractical or inefficient.

2.3.3 Distributed Self-Organization in MEC task offloading

With the proliferation of mobile devices, there has been a rapid increase in resource-intensive mobile applications, including media processing, online gaming, augmented reality (AR), and virtual reality (VR). These applications are now integral to both businesses and daily life. However, the limited computing power of mobile devices poses challenges in executing such demanding applications. Task offloading has emerged as an effective solution to address this computational burden. Offloading tasks to MEC, which is closer to the user equipment (UE), reduces communication latency compared to offloading to remote cloud servers. As a result, MEC offloading has become a prominent research topic. The proximity of MEC servers to end-users allows for real-time processing and quicker responses, which is crucial for latency-sensitive applications such as AR/VR and autonomous vehicles.

MEC offloading approaches involve transferring resource-intensive tasks to edge servers

to improve the efficiency of mobile device applications. The primary goal is to alleviate the burden on local devices by utilizing edge resources. Different offloading methods have been developed to achieve resource optimization, including mathematical models, heuristic approaches, and machine learning algorithms [59–61]. These methods can adapt to changes in network conditions, user demands, and available resources, making them effective in optimizing resource utilization.

The concept of social trust has also been incorporated into resource-sharing schemes. For example, the authors in [62] introduce a social trust scheme for resource utilization by integrating in-network caching, D2D communications, and MEC. Social trust mechanisms help ensure cooperation among users by encouraging honest behavior and discouraging selfish actions, which is crucial in decentralized environments where users may have conflicting objectives. Trust-based systems can significantly enhance resource-sharing efficiency by promoting collaboration and reducing resource hoarding.

In our study, we expand these concepts by adopting a distributed self-organization approach for CPU resource allocation and sharing in integrated networks involving SDN, SDR, and MEC. Our model considers a multi-user self-organizing system where users exhibit different preferences and personality traits. We propose a scheme in which users share their CPU resources by offloading tasks to neighboring users or edge cloud servers when necessary, thereby optimizing overall CPU resource consumption. Additionally, we evaluate energy efficiency within this integrated network. Our algorithm allows individual users to make autonomous decisions regarding task offloading while considering the impact of their actions on other users. This collaborative decision-making process leads to an overall improvement in system performance, including reduced latency and lower energy consumption.

In terms of use cases, we focus on the use of MEC in vehicular network scenarios. Vehicles today are equipped with onboard units (OBUs) that include multiple sensors, processing units, localization systems, and radio transceivers. These technologies enable the establishment of vehicular ad hoc networks (VANETs) [63]. However, the limited processing power of these devices makes it difficult to execute computationally demanding tasks, necessitating the offloading of tasks to Edge Server (ES) or cloud environments for enhanced computational resource availability. By leveraging MEC, vehicles can offload tasks to nearby edge

servers, thereby reducing the latency associated with data transmission to remote cloud servers. This is particularly important for applications such as driving assistance, collision avoidance, and traffic management, where real-time data processing is critical for ensuring safety and efficiency.

2.3.4 Optimization Methods in MEC CPU Resource Allocation

Existing literature on optimization has been applied to the CPU resource allocation problem in MEC. For instance, authors in [64] propose a multi-objective resource allocation approach for robotic workflows in edge cloud systems designed for smart factories of IoT. The method uses a hybrid algorithm to optimize objectives such as task latency and energy efficiency, incorporating the Non-dominated Sorting Genetic Algorithm II (NSGA-II) and a fuzzy logic system to handle conflicting objectives and uncertainty in resource management. This paper emphasizes balancing energy consumption with task completion time to enhance the performance of robotic workflows in smart factories.

Authors in [65] introduce a joint optimization strategy focusing on reducing energy consumption and latency in MEC systems for IoT. They model the problem as a mixed-integer nonlinear programming (MINLP) problem and solve it using a heuristic-based approach with a multi-objective optimization framework to address the trade-off between minimizing energy consumption and latency. The model integrates power control of transmission and optimal CPU frequency scaling, providing a comprehensive solution to energy-efficient edge computing in IoT environments.

Authors in [66] discuss multi-objective computation offloading for workflow management in cloudlet-based mobile cloud environments. They use the NSGA-II algorithm to handle competing objectives such as energy consumption, task completion time, and bandwidth utilization, incorporating a workflow dependency model that accounts for inter-task data flows. This ensures optimal resource allocation across the cloudlet network, enabling efficient resource distribution while minimizing computational costs in MEC systems.

Authors in [67] focus on a joint resource allocation strategy that integrates both communication and computational resources in Non-Orthogonal Multiple Access (NOMA) enabled MEC systems. They formulate an optimization problem to minimize energy consumption

while ensuring quality of service (QoS), employing Lagrangian Dual Decomposition and a convex optimization approach to solve the problem efficiently. The proposed model addresses interference management and power allocation, providing an energy-efficient solution for MEC systems.

However, the aforementioned optimization methods are centralized algorithms that lack the self-organizing capabilities and user negotiation features integral to our approach. In our thesis, we take a decentralized perspective, emphasizing collaboration and dynamic resource allocation among users. By incorporating personality traits, we introduce an adaptive mechanism that accounts for individual user preferences, enabling negotiation and cooperation. This leads to more balanced resource sharing and ultimately enhances overall system performance. Additionally, the use of personality traits allows us to optimize user satisfaction, making our system more responsive and user-centric compared to traditional centralized models.

Authors in [68] applied a semi-Markov decision model for optimizing computational resources in RSUs and vehicles in a vehicular cloud, but they did not consider negotiation between individual vehicles. Our model incorporates user-level resource sharing, leveraging personality traits to facilitate cooperation among users. By allowing vehicles to negotiate resource-sharing agreements based on their individual preferences and available resources, our model ensures equitable distribution of computational load, reducing bottlenecks and improving system performance.

In addition, existing methods for resource allocation in MEC primarily focus on computational resource allocation for task execution on edge servers. In contrast, our work considers both computation and communication resource allocation by integrating SDN and SDR technologies into MEC, thereby improving resource efficiency and energy savings for the entire integrated network. Our model enables the integration of communication and computation components in the same MEC server, improving overall CPU resource utilization efficiency. Moreover, our approach introduces user-specific preferences and personality traits to facilitate resource sharing, providing a new perspective on optimizing MEC-integrated cloud systems. This holistic approach ensures that both network and computational resources are utilized efficiently, leading to significant improvements in system

performance metrics such as latency, throughput, and energy consumption. The proposed integration of SDN, SDR, and MEC offers unprecedented flexibility for capacity allocation and service differentiation in next-generation networks. We adopt NFV characteristics to enable parallel resource allocation in our integrated model, considering both testbed measurements and real-time simulation results. Our experimental results demonstrate that the integrated SDN-SDR-MEC architecture significantly reduces power consumption while maintaining high performance. By decoupling the control and data planes using SDN, we can dynamically adjust resource allocation in response to network conditions, optimizing both computational and communication resources.

In conclusion, the integration of self-organized resource sharing with SDN, SDR, and MEC offers significant potential for improving cloud system efficiency. By incorporating personality-based user preferences and decentralized decision-making, our model optimizes CPU resource allocation, enhances energy efficiency, and ensures effective use of both computation and communication resources. These contributions provide a new direction for future research in cloud-integrated vehicular and edge networks. Furthermore, the use of game theory and social trust mechanisms introduces an additional layer of optimization by encouraging cooperation among users, leading to more balanced and efficient resource utilization. Our thesis combines the concept of self-organizing multi-user resource allocation algorithms with game theory to facilitate negotiation among users, aiming to achieve optimal user satisfaction in CPU resource allocation.

2.4 Summary of the related work

With the technology stacks rapidly evolving in the research areas of SDN, SDR, and MEC, researchers have begun to explore the potential integration of these technologies. The initial integration between SDN and SDR emerged due to the programmable nature of both technologies. SDN provides centralized control and management at the network layer, which has been effectively combined with the flexible functional splits at the physical (PHY) layer offered by SDR. This integration allows for enhanced adaptability and programmability across both network and physical layers, resulting in more efficient and flexible communication systems. Our work in this thesis draws inspiration from methodologies in the existing

literature but takes a different approach by focusing on CPU utilization and power consumption. We develop our testbed experiments to explore these aspects, providing insights into the parallelization of multiple SDN and SDR processes running co-located. This perspective allows us to investigate the efficiency gains from co-executing SDN and SDR processes, emphasizing the benefits of resource sharing and optimization that arise from this integrated setup.

The control plane technologies provided by SDN, such as OpenFlow, have increasingly been incorporated into MEC networks to enable flexible control, resource management, and dynamic adaptation for MEC task offloading and migration. By combining SDN's control plane capabilities with MEC's edge computing environment, this integrated approach offers a promising solution for managing dynamic workloads, reducing latency, and optimizing resource utilization across heterogeneous network environments. This integration makes MEC well-suited for applications that require low-latency and real-time processing, such as augmented reality (AR), virtual reality (VR), and vehicular networks, where responsiveness is crucial. However, the full potential of integrating SDR, which provides the flexibility to control the PHY layer, has not yet been fully explored in conjunction with MEC and SDN. SDR could play a pivotal role in further optimizing radio resource management, improving spectral efficiency, and enhancing communication reliability, which is essential in dynamic environments such as mobile edge computing. In this thesis, we incorporate SDR to bring additional flexibility to PHY layer control, thus allowing a more comprehensive approach to resource management and improving the overall performance of the integrated SDN, SDR, and MEC architecture.

Resource allocation algorithms, encompassing both centralized and distributed approaches, have gathered significant interest in the field of MEC research due to their potential to enhance efficiency and performance. Distributed approaches, particularly those involving self-organization and game theory-based algorithms, serve as conceptual cornerstones for the design and implementation of the optimization algorithms presented in this thesis. These approaches facilitate scalable and adaptive resource management, particularly in dynamic network environments. Personality traits have also been considered in our work to fine-tune user preferences and enhance satisfaction in the resource-sharing process. By accounting

for individual user characteristics, we aim to personalize the resource allocation procedure, ensuring that the system aligns more closely with the needs and behaviours of different users. This approach contributes to improved efficiency and fairness in resource sharing, ultimately leading to a higher overall satisfaction rate across the user base. Moreover, our proposed integration of SDN and SDR into MEC facilitates a comprehensive resource allocation strategy that takes into account both computational and communication resources, thereby achieving our goal of optimizing the entire system's CPU resource utilization.

Therefore, our proposed SDN, SDR, and MEC integrated network architecture involving these three elements goes beyond the current state-of-the-art and addresses this gap. This integration is one of the main contributions of this thesis. By bringing together the capabilities of SDN, SDR, and MEC, we create a comprehensive framework that allows for seamless coordination between computation and communication resources, leading to improved system efficiency and adaptability. Based on this integrated network architecture, we propose self-organizing and game-theory integrated algorithms for optimizing CPU resource allocation and power consumption in both the communication components (SDN and SDR) and the computation component (MEC). This represents another major research contribution of this work. Our algorithms consider the interdependencies between communication and computation resources, ensuring that the overall resource allocation is optimized for both performance and energy efficiency.

In conclusion, this thesis contributes to the enhancement of network and edge computing technologies by addressing a critical gap in the integration of SDN, SDR, and MEC. Our proposed integrated network architecture and resource optimization algorithms pave the way for future research on resource-efficient, scalable, and sustainable edge computing solutions. The work presented here provides a comprehensive framework that can be adapted and extended to various use cases, enabling a new generation of intelligent, responsive, and energy-efficient network systems.

3 Self-Organization Algorithms for CPU Resource Sharing

(This Page Intentionally Left Blank)

Self-Organization Algorithms for CPU Resource Sharing

In this chapter, we propose a resource-sharing scheme between cloud users to minimize resource utilization while guaranteeing Quality of Experience (QoE) of the users. The definition of users here is cloud users, i.e., can be Mobile Edge Cloud (MEC) users or public cloud users like Amazon AWS, Microsoft Azure, etc. As the real-time resource utilization of each user keeps on changing, the sharing system needs a dynamic resource-sharing strategy. We design a dynamic self-organizing sharing system based on the personality of the users. In addition, with the variation of real-time resource utilization, the users share their spare resources with each other according to their needs and their Personality Traits (PT). In this section, we first propose and implement an Evolutionary Multi-robots Personality (EMP) model, which considers the constraints from the environment (resource usage states of the users) and the update of two users' PT at each sharing step. We then implement a Single Evolution Multi-robots Personality (SEMP) model, which only considers the evolving user's PT and neglects the resource usage states. For benchmarking, we also implement a Nash Bargaining Solution Sharing (NBSS) model, which uses game theory but does not involve PT or risks of usage states. We use game theory to set the reward of each user's actions during the sharing procedure. The objective of our proposed models is to provide all the users with sufficient resources while reducing the total amount of excessive resources. We adopt the cloud service CPU resource as a use case to evaluate the performance of EMP. The results show that our EMP model performs the best, with the least iteration steps leading to the convergence and best resource savings. Based on the investigation of related works, our contribution in this chapter can be summarized as follows:

- we build a self-organized EMP model for user resource sharing in cloud system considering personality traits;
- we apply Personality Traits (PT) and game theory for the EMP model to optimize the sharing policy;
- we conduct a comprehensive analysis of the performance of our algorithm by comparing our EMP model (including EMP with Asymmetric Nash Bargaining strategy (EMP-A) and EMP with Fixed step strategy (EMP-F)) with three other baseline models, SEMP, NBSS, and CCFS).

In cloud services, cloud hardware resources, such as CPU, memory, disk, networking, etc., are virtualized before being provided to the customers. These virtualized resources are abstracted as one layer above the physical infrastructure layer and exposed to the customers. With this resource virtualization, cloud users/customers are designated to share the same pool of resources in the cloud. In the cloud system, customer usage patterns are highly dynamic and asynchronous. Therefore, there exist opportunities for cloud operators (IT operation teams) to reallocate resources between users dynamically to save physical resource usage [69, 70]. From the service provider's perspective, a business model that encourages sharing excessive resources between customers helps to save the overall cloud resources for the service providers. In addition, since these shared resources are excessive, the QoE of users is not compromised. This concept of resource sharing provides a new option for the IT operation team to optimize the resource utilization of the cloud. A successful sharing scheme includes the following key points: 1) to give enough reward to the users who share; 2) to make sure that the users do not experience service disruptions while sharing; and 3) to give resources back to users when they need extra resources. In cloud systems, the major challenges for designing an optimal sharing scheme are the following: 1) the real-time user demands for cloud resources, e.g., CPU, memory, and network bandwidth, are asynchronous, dynamic, and hard to predict, which makes the amount of excessive resources quite uncertain; 2) the users have different personalities, either conservative or generous, which affect their choices of sharing or requesting resources; 3) traditional centralized resource allocation algorithms are inefficient in solving this problem due to the difficulty of tracking and grouping large-scale users.

To cope with the resource-sharing problem of large-scale cloud users, we bring in the concept of a Self-organizing multi-robot system [51]. The goal of this concept is to provide solutions with robots to imitate human/animal behaviours/personalities and make optimal decisions in a decentralized way. With this concept applied in the cloud resource allocation system, the cloud system is capable of achieving optimal resource utilization without compromising the QoE of the customers. In this chapter, we design an EMP model for our sharing system, which is a model originated from the self-organizing multi-robot system. The goal of this sharing procedure is to dynamically allocate spare resources from one user to another who needs that resource in the meantime. Besides, we define that each user has PT, which leads them to have different preferences in different situations. In addition, the update of their PT during the sharing procedure causes them to make different actions at each iteration step to achieve the goal of sharing, i.e., all users have sufficient resources, and QoE is guaranteed. During the sharing procedure, we use the Nash Bargaining method in game theory to obtain optimal policies.

3.1 Introduction and Background

Self-organizing multi-robot system is a decentralized system for the optimization of group-based user behaviours [51]. The update of personalities and the corresponding effects on the collaboration of users have also been brought to this research. For example, the authors in [51] used personality to build a self-organized algorithm to research a problem of multiple robots leaving a room in a self-organized way. The authors in [52] proposed a game-theory-based approach to swarm robots to collaborate with each other to chase a target. The authors in [53] deal with the evolved control of a swarm of robots to decide the signalling and connectivity of communications with each other. The authors in [54] use an artificial-intelligence based platform to measure the personalities of human groups. However, there is no application for this self-organization concept in the literature that focuses on resource allocation in cloud-based systems. In this chapter, we focus on this research gap and apply the self-organization system method to the MEC resource allocation problem. In this chapter, we bring the concept of the update of the personality of users to solve our resource-sharing problem in cloud systems. In addition, we use game theory when defining

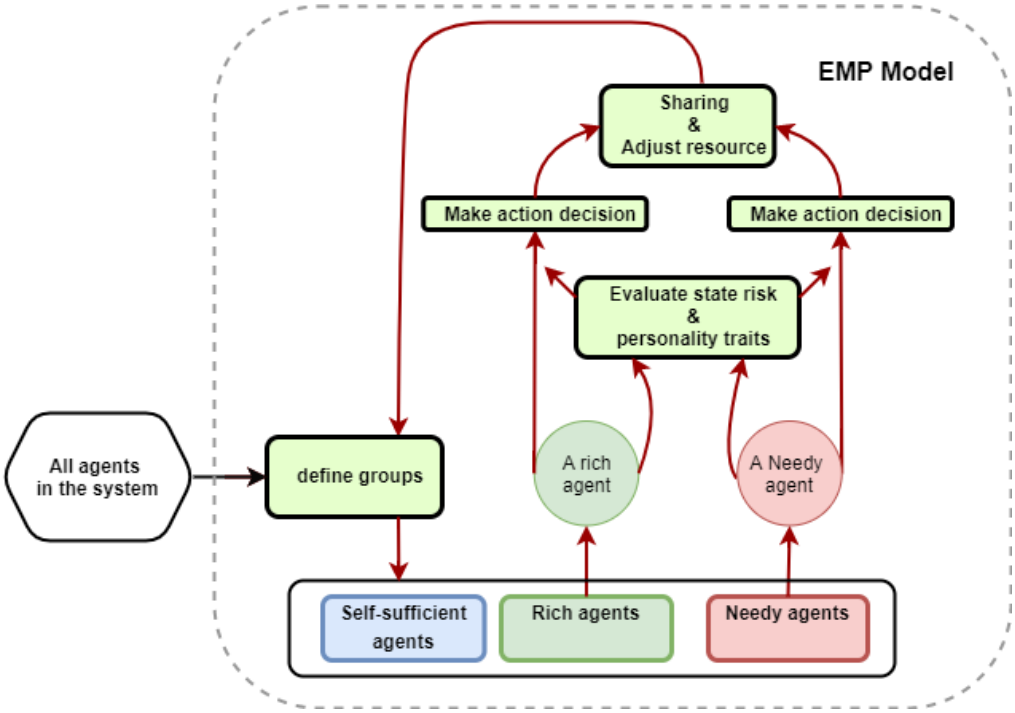


Figure 3.1: EMP model diagram

rewards in the system for the users to take optimal actions. Our work brings a novel scope for researchers to optimize resource allocation when operating the cloud system.

3.2 EMP Model

In this section, we design the EMP model, which is based on a self-organized multi-robot system that considers the personalities of the users. In the EMP model, we design two algorithms EMP-A and EMP-F, considering both PT and the risks of running short of resources. After that, we design 2 algorithms as control groups for benchmarking: one is SEMP, in which every user only cares about its own PT at each iteration step without considering the risk of shortage of resources; the other one is NBSS that uses Nash Bargaining Solution (NBS) for users without involving the users' PT in the sharing.

In this EMP model shown in Figure 3.1, we focus on the cooperation between the users without considering the competition between them, which aligns with the service scenario in Infrastructure as a Service (IaaS) of the cloud where users should not realize any competition between them when sharing resources. To quantify the personality of users in EMP, we

introduce PT in section 3.2.3, which describes the users' tendency to share. It affects the decision of whether to share and how much to share. We set all users in our system as homogeneous users. The only difference between them is in the parameter settings of the users based on their preferences. To describe the users' different preferences, we use one pair of numerical values named PT [71], which is also called the orientation of the users [51]. After that, when calculating the resources of sharing conducted by the users, we adopt Game Theory [52] and Asymmetric Nash Bargaining Solution (ANBS) [72] to derive mathematical formulation.

3.2.1 The System Definition

We use a multi-user robotic system imitating cloud users to give suggestions for sharing. Each user i in the system has the same resource value R_{fixed} at the beginning and has a flexible extra resource quota $Re(t)$, which denotes the resources it would get from other users at time t, with the upper boundary Re_{max} restricting the greediness of each user. The resource usage set is denoted as $U = \{U_i | U_i \in [0, R_{fixed} + Re_{max}], i = 1, 2, 3....n\}$, and the real-time resource at time t holds $Rr_i(t) = R_{fixed} + Re_i(t)$.

When users get real-time resource usage information (e.g., the system measures users' usage amount in a time interval), the users start to evaluate whether they have enough resources for their current usage. Then, the EMP model will guide the users to adjust their resource allocation in the system. Besides, the total amount of resources in the system stays the same. For each user i, the spare resource is denoted as $Rs_i(t) = Rr_i(t) - U_i(t)$. According to the real-time resource, we defined all users in three groups: 1) needy users ($Rs_i(t) < 0$); 2) rich users ($Rs_i(t) > 0$); 3) self-sufficient users($Rs_i(t) = 0$).

3.2.2 The Action and Payoff Definition

In every sharing step, we pick up two users: one rich user and one needy user. In principle, both users have two action choices: giving out resources, denoted as a_0, or getting resources, denoted as a_1, the action space can be defined as $A = [a_0, a_1]$. The payoff matrix for users is designed in Table 3.1.

Table 3.1: Payoff for individual users

Ma		Payoff for needy user	
		give a_0	get a_1
Payoff of rich user	give a_0	0 , -X	Y, Y
	get a_1	-Z ,-Z	-X ,0

$M\theta$	State risk	
	high risk θ_0	low risk θ_1
Payoff of user for state	-A	B
	C	-D

The payoff matrices (i.e., X, Y, Z, A, B, C, and D) shown in Table 3.1 represent the values used in our algorithm. We define these with empirical numbers to simulate real sharing scenarios in the cloud network and to ensure fast convergence of the algorithm during the sharing process. We assume that X, Y, Z, A, B, C, and D are all positive, representing rewards. However, we assign negative values (using "-") to some of them to indicate penalties. The overall concept is to encourage resource-rich users to share with resource-needy users, provided there is sufficient willingness from the former.

3.2.3 The User PT and State Definition

Our EMP model defines the users' PT, which makes the users imitate human emotions. These emotions act as an internal motivation for their behaviors. In our case, the vector PT, $\beta_i(t) = [\beta_{i,0}(t), \beta_{i,1}(t)], \quad (\beta_{i,0}(t) \geq 0, \beta_{i,1}(t) \geq 0)$, describes the user i's cooperation willingness at a given time t, where the $\beta_{i,0}(t)$ represents 'generous' trait and $\beta_{i,1}(t)$ represents 'eager' trait, and $\beta_{i,0}(t) + \beta_{i,1}(t) = 1$.

Other than the PT, the resource availability state s is another factor that influences the users' actions that we have to take into account together with the PT. The user in a safe resource state means a user has enough resources for its usage and is more likely to share. Otherwise, it is more likely to ask for help from the other users. We describe the high-risk state at time t by $P_{\theta i}(\theta_0|\theta, t)$, and low risk state as $P_{\theta i}(\theta_1|\theta, t)$ in Equation (3.1).

$$P\theta_{i0}(t) = normalize(\frac{U_i(t)}{Rr_i(t)} + (\beta_{i,0}(t) - \beta_{i,1}(t))); \;\; P_{\theta i}(\theta_1|\theta, t) = 1 - P_{\theta i}(\theta_0|\theta, t) \qquad (3.1)$$

Besides, we describe the value function for users when they are in sharing step as the

following:

$$\begin{aligned} V_i\left(s,a,t\right) &= E\left\{J_i\left(s,a,t\right)\right\} = M\theta \cdot P\theta_i^T \cdot J_i(s,a,t) \\ &= \begin{bmatrix} M\theta_{11}, M\theta_{12} \\ M\theta_{21}, M\theta_{22} \end{bmatrix} \cdot \begin{bmatrix} P_{\theta i}(\theta_0|\theta,t) \\ P_{\theta i}(\theta_1|\theta,t) \end{bmatrix} \cdot J_i(s,a,t) \end{aligned} \tag{3.2}$$

where,

$$J_i(s,a,t) = P_{ai} \cdot Ma \cdot P_{aj}^T = [P_{ai}(a_0|s,t), P_{ai}(a_1|s,t)] \cdot \begin{bmatrix} Ma_{11}, Ma_{12} \\ Ma_{21}, Ma_{22} \end{bmatrix} \cdot \begin{bmatrix} P_{aj}(a_0|s,t) \\ P_{aj}(a_1|s,t) \end{bmatrix} \quad (i,j = 1,2....n \quad i \neq j) \tag{3.3}$$

The payoff matrices $M\theta$ and Ma are defined in Table 3.1. Once the user recognizes its state situation and its value function, its action selection strategy uses randomized strategy [73], which considers the user is 'exploring' new action as well as 'exploiting' learned action. The probability of action selection holds:

$$P_{ai}(a_0|s,t) = \frac{k^{V_i(s,a_0,t)}}{k^{V_i(s,a_0,t)} + k^{V_i(s,a_1,t)}};\ P_{ai}(a_0|s,t) + P_{ai}(a_1|s,t) = 1 \tag{3.4}$$

where the coefficient k represents how often the user would like to 'explore' rather than 'exploit'. In our application, we set it to be $e = 2.718$.

3.2.4 The Sharing Strategy

At each sharing step, assuming both users are rational and intend to maximize their spare resource utility in the bargain, the EMP model considers the two-user cooperation as a bargain problem. The set of spare resource utility function can be described as $\Gamma = \{\gamma_i\,|i = 1,2\}\}$, where $\gamma_i = \{(Rr_i(t) - U_i(t))|i = 1,2\}$, which is a nonempty compact convex set with boundary ([74, 75]). When the real resource is equivalent to their usage, get the status point for

each user. In this case, the sharing problem can be described as [72]:

$$\begin{aligned} \Gamma^* = \underset{Rr_i(t)}{argmax} \prod_i (Rr_i(t) - U_i(t))^{\lambda_i(t)} \\ s.t. \qquad \sum_{i=1}^{2} Rr_i(t) = \Phi \\ \sum_{i=1}^{2} \lambda_i(t) = 1 \\ Rr_i(t) \geq U_i(t), \quad i = 1, 2 \\ Rr_i(t) \geq 0, \quad i = 1, 2 \end{aligned} \tag{3.5}$$

$$\lambda_i(t) = \frac{\beta_{i,1}(t)}{\beta_{i,1}(t) + \beta_{j,1}(t)} \quad (i, j = 1, 2, 3...n) \tag{3.6}$$

where Φ is the total real-time resource of those two users. $\lambda_i(t)$ denotes the bargaining power of user. At each allocation step, the user gets their real-time resource as $U_i(t) + \lambda_i(t) \cdot \sum_{i=1}^{2} (Rr_i(t) - U_i(t))$.

3.2.5 The PT Update Strategy

After the sharing, the real-time resource held by every user involved in the sharing step has been changed, leading their PT to evolve. This will affect their action decisions in the next sharing steps. We define the updating rule as the following [76]:

$$\beta_{i,m}(t) = \beta_{i,m}(t-1) + \alpha \Delta\beta_{i,m}(t), \quad (m = 0, 1), \quad \alpha \in (0, 1) \tag{3.7}$$

where, α is the learning rate and the $\Delta\beta_{i,m}(t)$ holds as :

$$\begin{aligned} \Delta\beta_{i,m}(t) = \frac{\Delta J_i(s, a_m, t)}{\Delta J_i(s, a_m, t) + \Delta J_i(s, a_l, t)}, \\ \Delta J_i(s, a_m, t) = J_i(s, a_m, t) - J_i(s, a_m, t-1), \\ (m, l = 0, 1 \quad m \neq l) \end{aligned} \tag{3.8}$$

Then we implement the aforementioned EMP model by Algorithm 1 with two different sharing policies, i.e. EMP-A using ANBS, and EMP-F using fixed-value sharing strategy

during the sharing (i.e., all the users share a fixed value of resources during a step of sharing).

Algorithm 1 for EMP, including EMP-A and EMP-F

```
Initialisation:
Initial$\beta = [0.5, 0.5]^T$,$Pa = [0.5, 0.5]^T$
Split users into three groups according to $Rs_i(t)$
Sharing step:
while there are needy users in the system do
    for pick one rich user do
        for pick one needy user do
            1) evaluate state risk by equation (3.1)
            2) calculate expected value by equations (3.2-3.3)
            3) calculate Pa by equation (3.4)
            4) select and execute actions by Pa
            5) share spare resource:
            if sharing by ANBS strategy(EMP-A) then
                sharing value by equation (3.5)
            if sharing by fixed value strategy(EMP-F) then
                sharing with fixed value (e.g.5 units).
            6) update PT by equation (3.7-3.8)
            7) update the groups
            if no needy user in the system then
                break
```

3.3 Other Models

3.3.1 The SEMP Model

We build an SEMP model to be one of the baseline models to compare with our EMP model in Section 3.2. Unlike the EMP model that involves both users in state evaluation before sharing and updates PT after sharing, the SEMP model, implemented as Algorithm 2, only considers the users' PT when making actions and neglects the evaluation of the state risk probabilities. During the sharing steps, the SEMP algorithm also adopts the ANBS strategy and updates the rich user's PT at the end of sharing. Without considering the state risk

probability, the PT update functions are simplified as:

$$P_{ai}(a_0|s,t) = \frac{e^{\beta_{i,0}(t)}}{e^{\beta_{i,0}(t)} + e^{\beta_{i,1}(t)}},$$
$$\beta_{i,m}(t) = \beta_{i,m}(t-1) + \alpha \cdot \ Ma$$
$$(m = 0, 1), \quad \alpha \in (0, 1) \quad (3.9)$$

Algorithm 2 for SEMP

```
Initialisation:
Initialize β = [0.5, 0.5]^T, Pa = [0.5, 0.5]^T
Split users into three groups according to Rs_i(t)
Sharing step:
while there are needy users in the system do
    for pick one rich user do
        for pick one needy user do
            1) calculate Pa by equation(3.9)
            2) select and execute actions by Pa
            3) share spare resource by ANBS strategy
            4) update PT by equation(3.9)
            5) update the groups:
            if no needy user in the system then
                break
```

3.3.2 The NBSS Model

We build an NBSS model as another baseline algorithm to compare with our EMP model and SEMP model. In the NBSS model, without considering the state influence, we assume all users have the same personality and the needy-rich users pairs share their total spare resource by NBS as $U_i(t) + 1/2 \cdot \sum_{i=1}^{2} (Rr_i(t) - U_i(t))$.

3.3.3 The Cloud Cooperative-Federation Sharing (CCFS) model

Apart from the four models we have developed (EMP-A, EMP-F, SEMP, and NBSS), we have also reviewed similar models in the literature to compare their performance with ours. Since our integrated network architecture (i.e., the combination of SDN, SDR, and MEC) is novel, there are no directly comparable algorithms available in the literature that are designed for exactly the same network scenarios. Therefore, we have identified the most

similar approaches and modified them to suit our specific integrated architecture.

One such model is introduced by the authors in [77], where they present an optimization scheme for resource sharing in cloud environments. Their work proposes an efficient method for handling resource allocation by leveraging game theory and queuing models to optimize resource distribution. Specifically, they propose the Cloud Cooperative-Federation Sharing (CCFS) algorithm, which operates under the assumption that participants in the sharing scheme are fully cooperative. In their model, a central broker is responsible for determining the best possible request transfer policy among participants involved in resource sharing, thereby ensuring that resources are allocated optimally.

In the following sections, we compare the performance of our proposed algorithms with the CCFS algorithm to evaluate how our distributed, self-organizing approaches stack up against a more centralized and cooperative method. By doing this comparison, we aim to highlight the advantages and potential trade-offs of using distributed game-theory-based resource sharing in our newly proposed integrated SDN, SDR, and MEC network architecture. Our analysis will demonstrate how factors like network dynamics, user behaviour, and cooperation levels impact the efficiency of resource allocation and overall system performance.

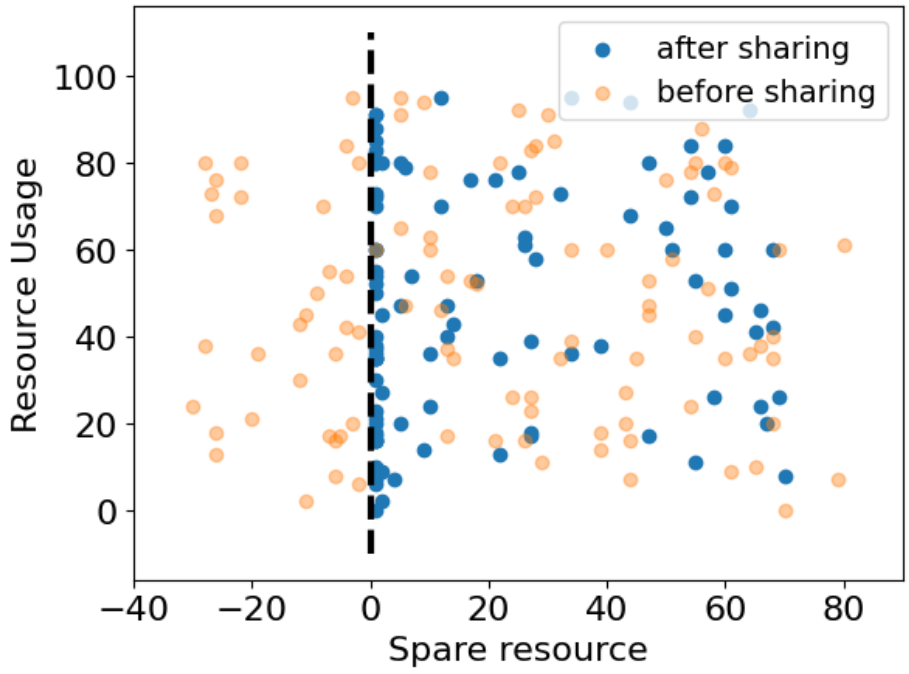
3.4 Simulation Results

In this section, we present the experimental results of our sharing models. We simulate a 100-user system for sharing resources in a cloud system. To initiate our experiment, we assume that each user has a fixed resource of 70 units at the beginning, and the maximum extra resource it could get from the other users is 30 units. The real-time resource usage for each user is a set of numbers in the range of $[0, 100]$ units. We define one 'resource usage measuring round' as the time window between two consecutive instances of measuring the users' resource usage when the resource sharing between all users should start and finish. During each 'resource usage measuring round', each user conducts multiple 'sharing steps' with other users to achieve the goal of eliminating all needy users. We investigate two use-case scenarios: 'independent sharing' and 'continuous sharing'. 'Independent sharing' means the real-time resource usage in each measuring round is randomly generated, and the

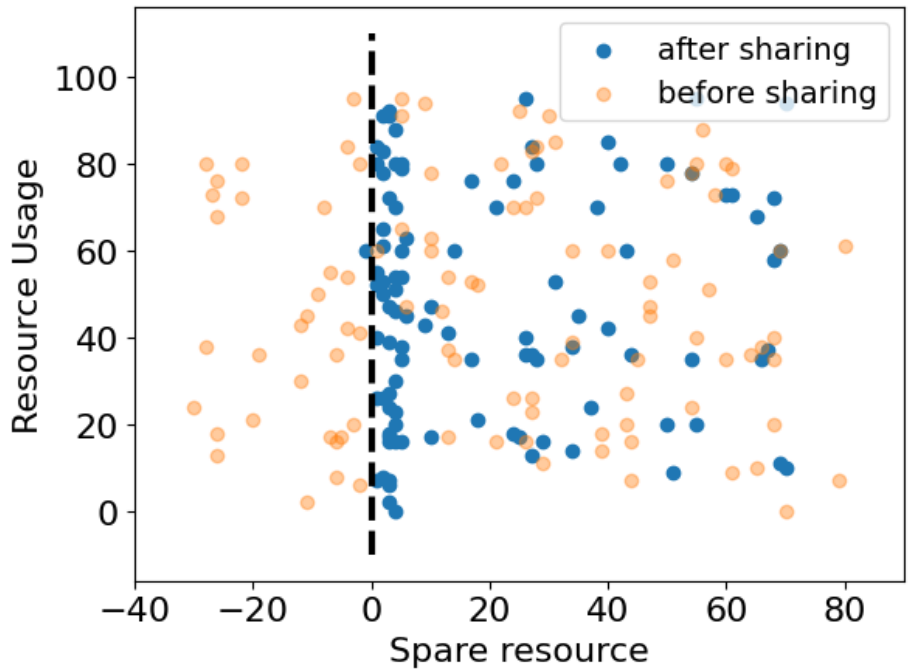
remaining resource from the previous round is not rolled over to the next round. However, 'continuous sharing' means that real-time resource usage follows a sinusoidal pattern during the time of the day, and the remaining resource is rolled over to the next round. We investigate the performance of the four aforementioned models: 1) The EMP-A model; 2) The EMP-F model; 3) The SEMP model; 4) The NBSS model; 5) The CCFS model.

3.4.1 Resource sharing results

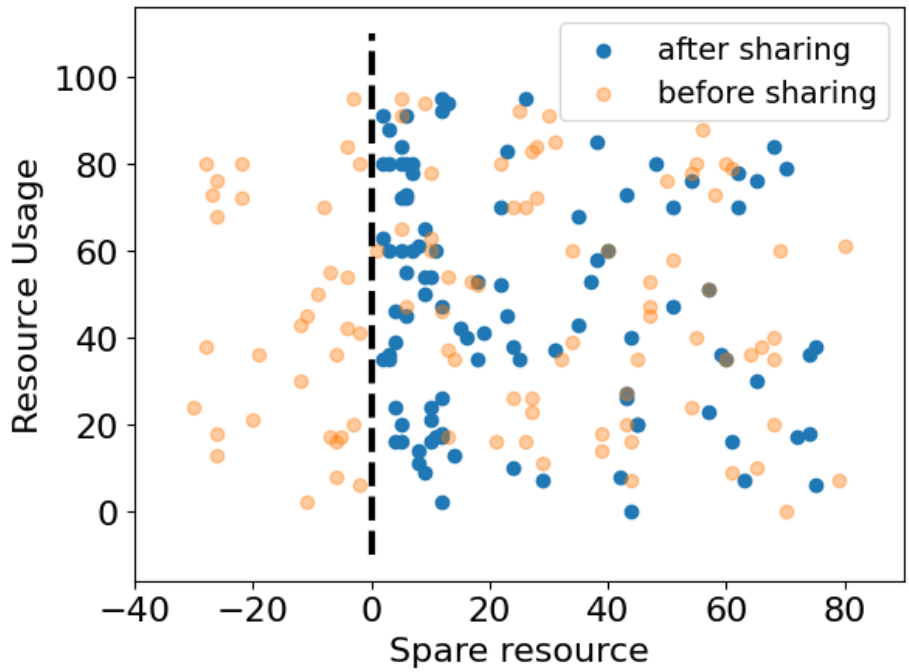
Figure 3.2 presents the simulation results for the algorithms in terms of the amount of spare resources before and after resource sharing during a usage measurement round. Each dot in the figure represents an individual user. On the x-axis, a negative value of spare resources indicates that a user's real-time resources are insufficient to meet their needs, while a positive value indicates that the user has excess resources. The results demonstrate that all five sharing algorithms (including the four we proposed and the CCFS algorithm from related work) successfully provide each user with sufficient resources, as evidenced by all users having non-negative spare resources after sharing (represented by the blue dots on the right side of the figure). The orange dots represent resource availability before sharing, and the overlapping brown dots are a result of coinciding orange and blue dots.



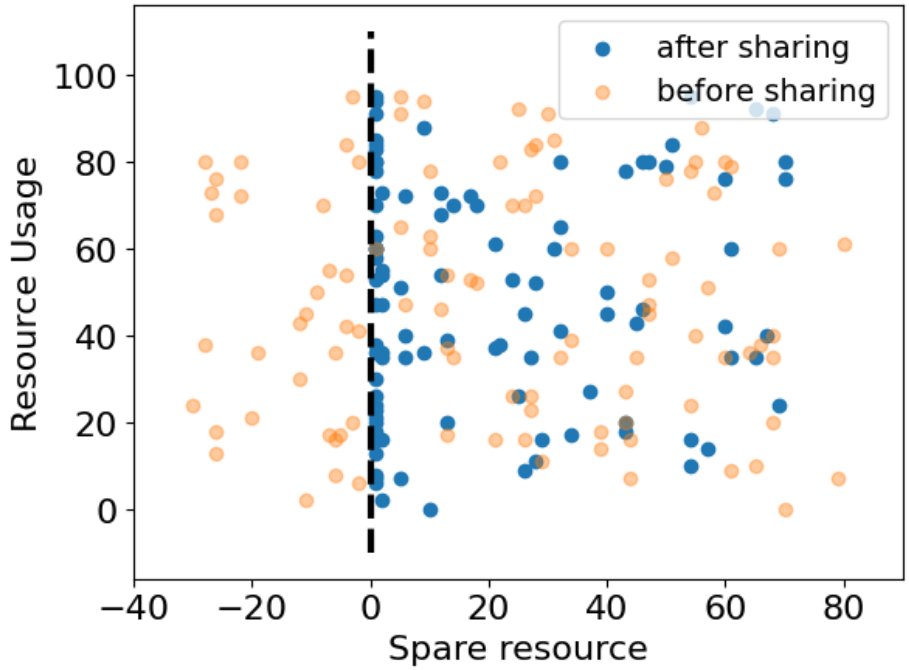
[a] EMP-A


after sharing
before sharing
Resource Usage
Spare resource
100
80
60
40
20
0
−40
−20
0
20
40
60
80

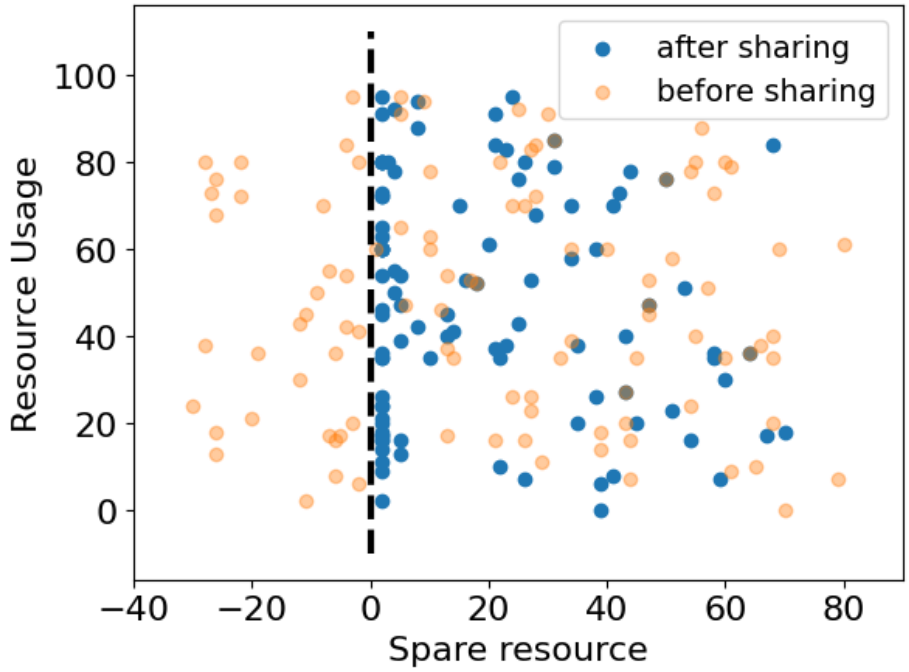
[b] EMP-F


after sharing
before sharing
Resource Usage
Spare resource
100
80
60
40
20
0
−40
−20
0
20
40
60
80

[c] SEMP



[d] NBSS



[e] CCFS

Figure 3.2: 100-user distribution of independent sharing in one resource usage measuring round by different algorithms (orange dots show the users' distribution before sharing; blue dots show the users' distribution after sharing): [a] EMP-A; [b] EMP-F; [c SEMP; [d] NBSS; [e] CCFS

We begin our simulation with a randomly generated usage pattern across all users. Figure 3.3 illustrates the resource-sharing process for a single needy user within one usage

measurement round. Through sharing, the user gradually receives additional resources until becoming self-sufficient. Among the five sharing algorithms, EMP-A and SEMP demonstrate the best performance.

For the overall system, Figures 3.4, 3.5, and 3.6 depict the reduction in the number of needy users, the total lacking resources, and the total spare resources across the entire system during the sharing steps within one usage measurement round. These figures compare the performance of the four algorithms with the scenario where no sharing takes place. From the results, EMP-A outperforms the other algorithms, including the CCFS algorithm from the literature.

To understand the reasons behind the varying performances of these algorithms, we can examine their underlying principles. The EMP-A algorithm incorporates personality traits and defines the resource exchange between needy and resource-rich users through negotiation based on these traits, resulting in greater flexibility and higher satisfaction for all users. EMP-F, on the other hand, has a fixed resource exchange amount between needy and rich users without considering personality traits, which limits its adaptability.

The SEMP algorithm considers the personality traits of only the resource-rich users who provide resources, but not those of the needy users requesting resources. The NBSS algorithm takes an equalization approach, always attempting to equalize resources between rich and needy users during each sharing round. The CCFS algorithm assumes full cooperation among users, with a central broker managing the sharing process. While CCFS is slightly more flexible than NBSS, it does not incorporate personality traits, which limits its adaptability.

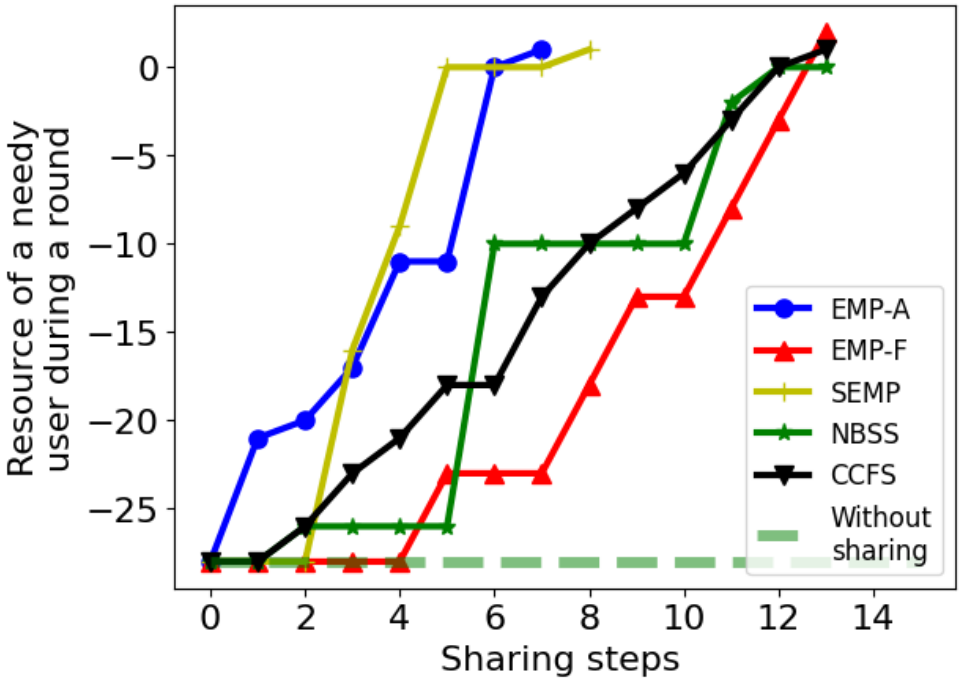


Figure 3.3: Resource of a needy user during one independent usage measuring round vs. sharing steps, with random usage pattern.

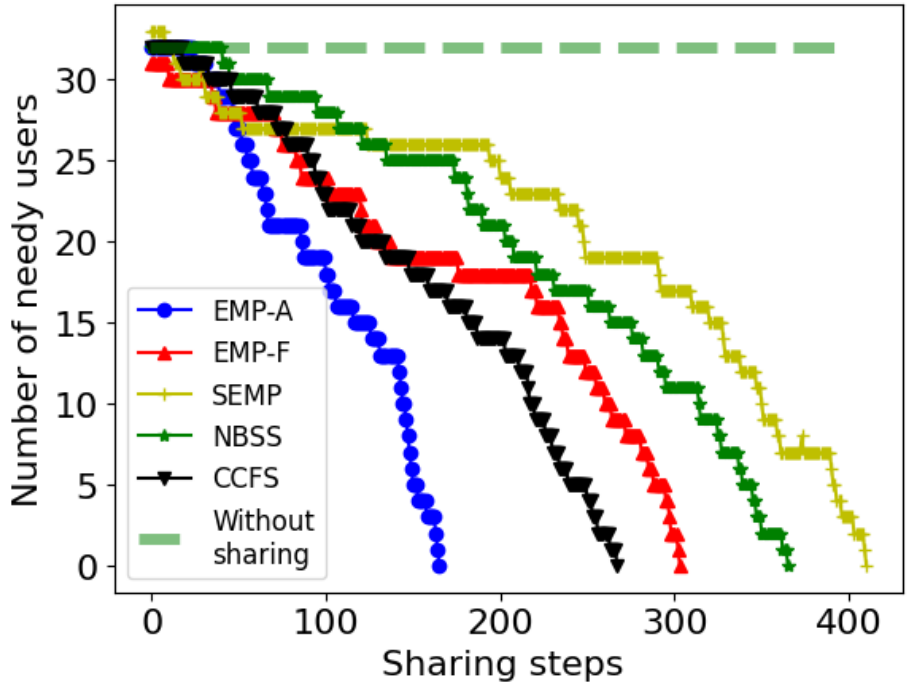


Figure 3.4: Number of needy users during one independent usage measuring round vs. sharing steps, with random usage pattern.

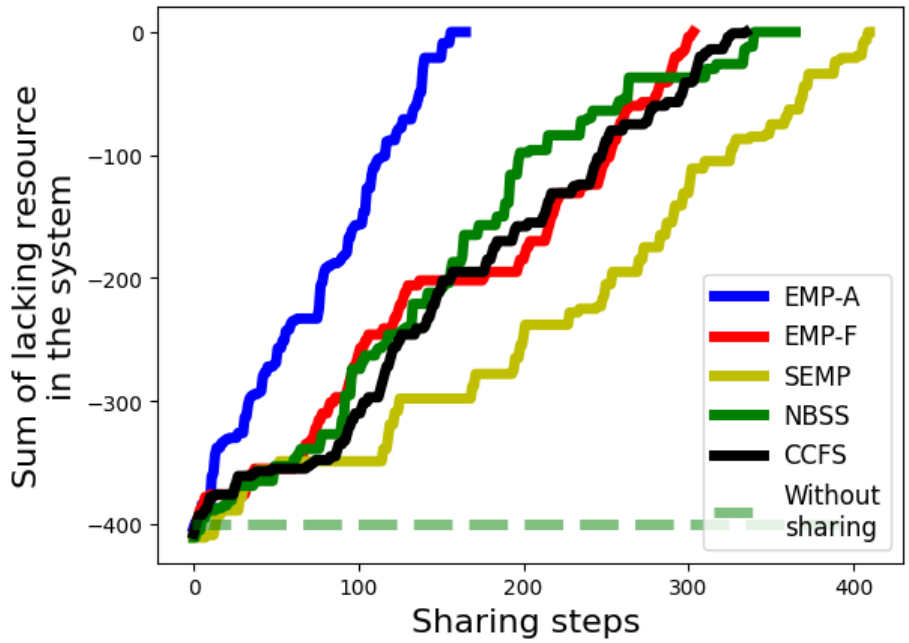


Figure 3.5: Sum of total lacking resource during one independent usage measuring round vs. sharing steps, with random usage pattern.

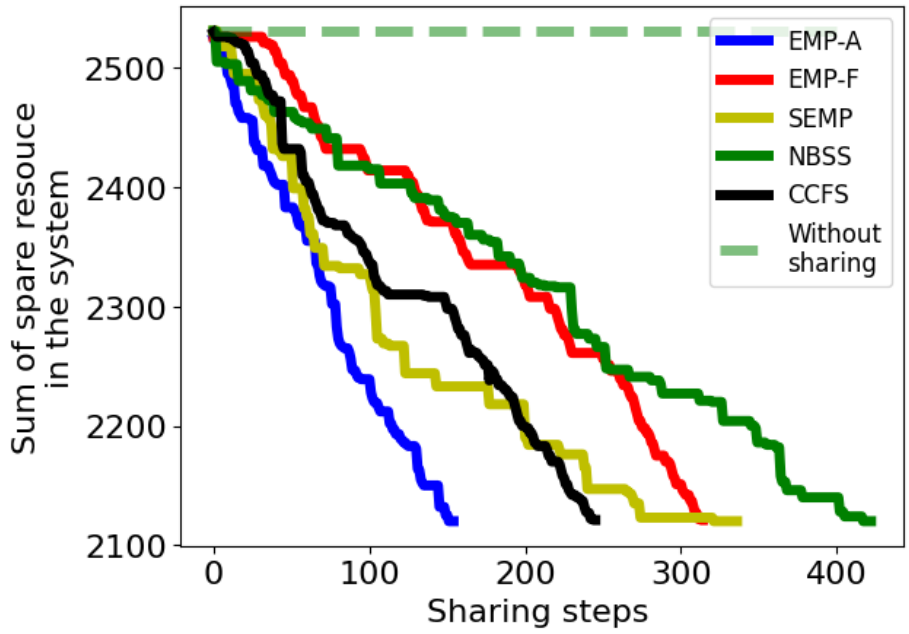


Figure 3.6: Sum of total spare resource during one independent usage measuring round vs. sharing steps, with random usage pattern.

With resource sharing, needy users in the system receive sufficient resources from resource-rich users. As a result, the amount of spare resources and the number of needy users decrease during the sharing process. Figure 3.7 illustrates the number of sharing steps required to

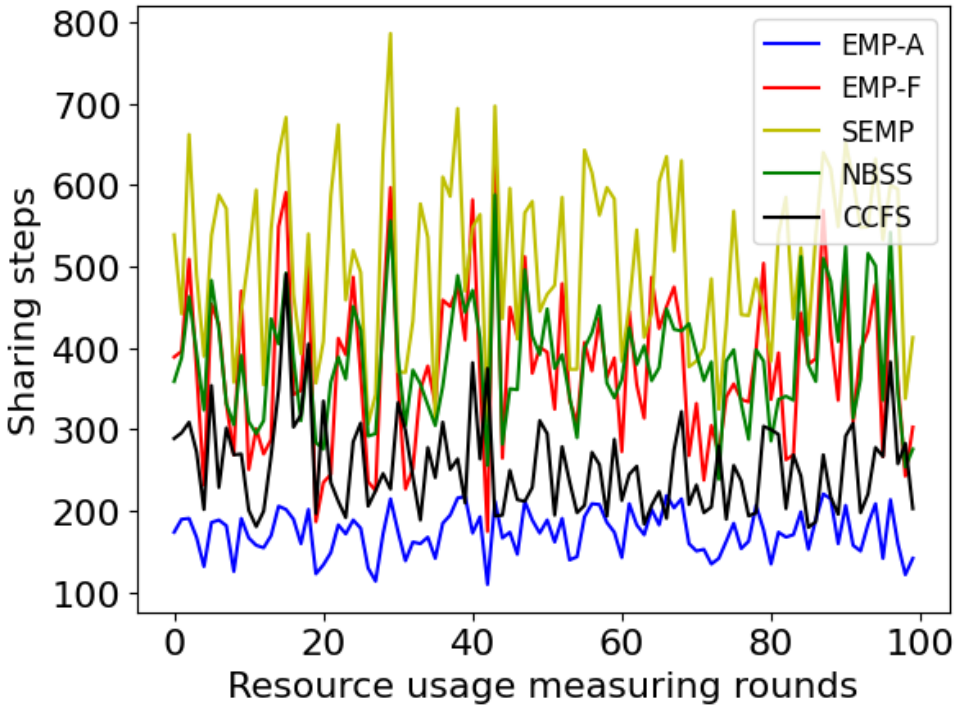


Figure 3.7: Number of sharing steps in 100 independent rounds with random usage pattern.

reach zero needy users over 100 rounds of individual experiments. The fewer steps it takes to reach zero needy users, the better the performance of the algorithm. All sharing algorithms efficiently achieve this goal, with the EMP-A algorithm performing the best. While CCFS outperforms the other three methods, it still does not match the performance of EMP-A.

In summary, the EMP-F, SEMP, NBSS, and CCFS algorithms have certain limitations and are less flexible and optimized compared to EMP-A. These differences in design are the primary reasons behind the varying performances observed in the simulation results.

3.4.2 Continuous sharing with sinusoidal usage pattern

In this subsection, we consider continuous resource sharing with a sinusoidal usage pattern. The resource usage follows a sinusoidal function over the course of the day, reflecting the peak and off-peak hours of network traffic. This type of traffic pattern is commonly used in cellular networks, particularly in energy-efficient networking research [78]. As mentioned earlier, in this scenario, any remaining resources from the previous round are rolled over to the next round. Additionally, we assume that each user has a different phase for their daily sinusoidal usage pattern, meaning that peak usage times vary for each user throughout the

day. This setup emulates different user groups, such as residential and business users, who have distinct usage patterns. For example, residential users typically experience peak usage in the evenings, while business users' peak times are generally in the mornings or afternoons.

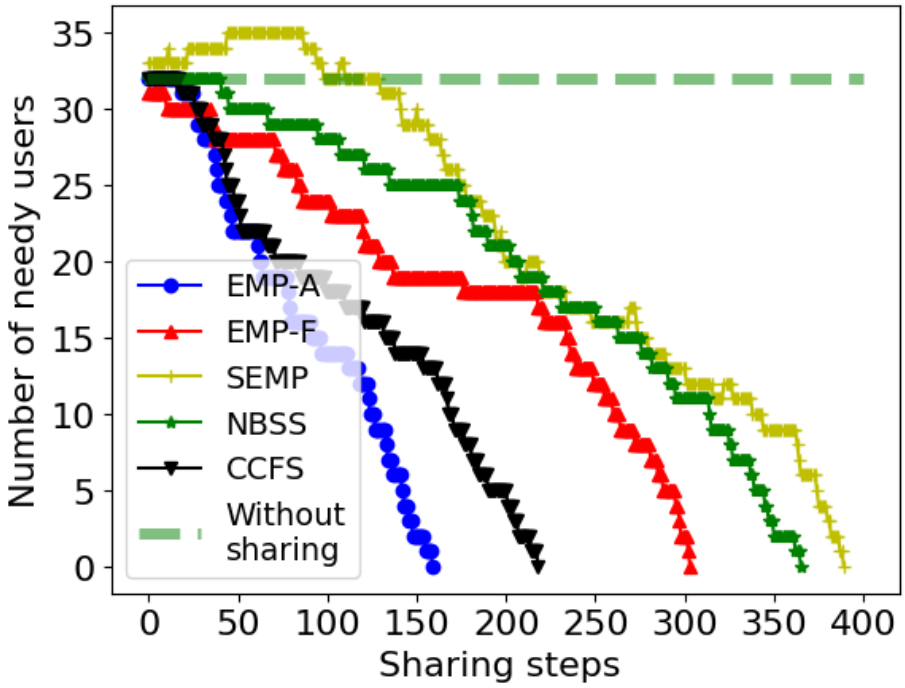


Figure 3.8: Number of needy users during one independent usage measuring round vs. sharing steps, with sinusoidal usage pattern.

In this scenario, the performance of the four algorithms is shown in Figure 3.8 and Figure 3.9. Similar to the previous case with random usage, the EMP-A algorithm requires fewer sharing steps than the other three algorithms to reduce the number of needy users to zero. The performance of the other four algorithms is also comparable to each other. However, in this continuous sharing scenario with a sinusoidal usage pattern, the advantage of EMP-A over the other four algorithms is more pronounced compared to the independent sharing case. This is expected, as the continuous sharing scenario is more stable and better reflects real-world conditions.

3.4.3 Average percentage of satisfaction time of users

We assume that, in a cloud system, the system gets real-time usage information measured every 15 minutes and schedules one sharing step every 1.5 seconds, i.e., the system can schedule up to 600 sharing steps in 15 minutes. As shown in previous Figures(3.8, 3.7, and

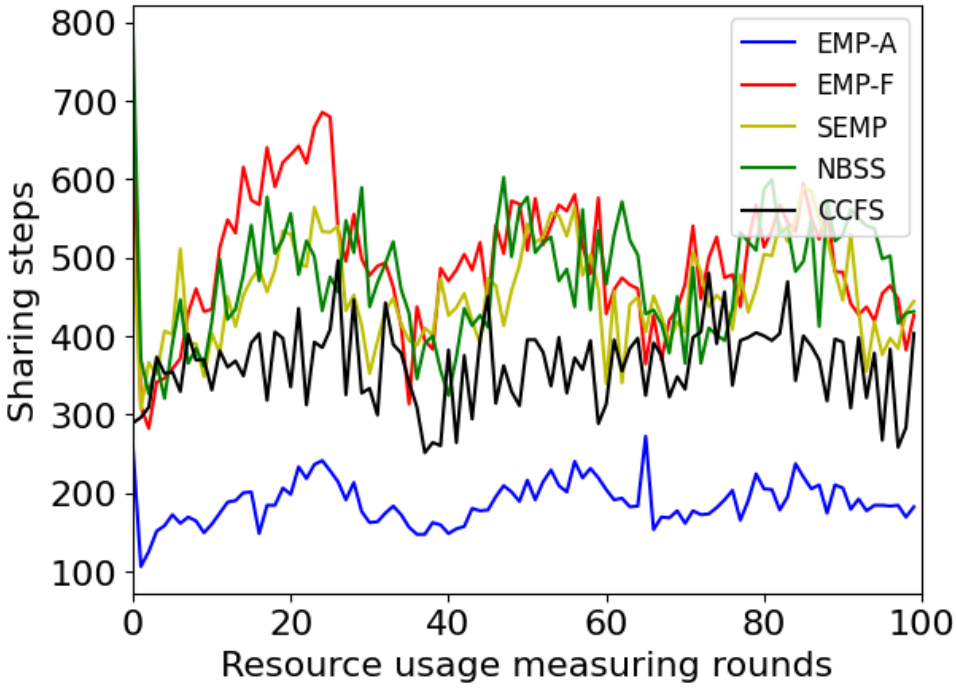


Figure 3.9: Number of sharing steps in 100 continuous resource usage measuring rounds with sinusoidal usage pattern.

Table 3.2: Average percentage of satisfaction time of users

	Without sharing	With sharing				
Usage Pattern		EMP-A	EMP-F	SEMP	NBSS	CCFS
100 rounds of independent sharing with random usage pattern	69%	95%	87%	87%	87%	90%
100 rounds of continuous sharing with sinusoidal usage pattern	63%	95%	85%	88%	84%	89%

3.9), most of the time, all 4 algorithms can make the number of needy users equal to 0 (all users reaching satisfaction) within 600 sharing steps. However, for different algorithms, the needy users spend different sharing steps to get sufficient resources before reaching satisfaction. We calculate and show the average percentage of satisfaction time for all users in Table 3.2. The EMP-A has the highest percentage of satisfaction time as 95% during the sharing. Without sharing, the percentage is only 69% in the random usage pattern and 63% in the sinusoidal usage pattern. The other four algorithms (including the CCFS from the literature) have similar performance of around 85%-90% during the sharing, which is higher than the case without sharing but not as good as the EMP-A.

3.5 Conclusion

In this chapter, we integrate the concept of Artificial Swarm Intelligence and Personality Traits to design a multi-user system on the resource sharing of the cloud system. We have designed and implemented our main algorithm, i.e., EMP-A, as well as three other algorithms, EMP-F, SEMP, and NBSS for comparison. All algorithms are capable of re-allocating spare resources to the needy users through the sharing procedure between the users, without adding external resources. To evaluate the performance of the 4 algorithms, we simulated a 100-user system and executed 100 usage measuring rounds. The results showed that EMP-A performances much better than the other three algorithms (SEMP, EMP-F and NBSS) in terms of fewer sharing steps as well as higher user satisfaction rate.

(This Page Intentionally Left Blank)

4 Energy Efficient SDN and SDR Joint Adaptation of CPU Utilization

(This Page Intentionally Left Blank)

Energy Efficient SDN and SDR Joint Adaptation of CPU Utilization

In this chapter, we investigate the Software-Defined Radio (SDR) and Software-Defined Networking (SDN) integration and parallelization characteristics experimentally in our Trinity College Dublin OpenIreland testbed. We analyze CPU utilization and power consumption in the OpenIreland testbed for different parameter settings and use case scenarios of this integrated architecture. The experiment results show different behaviours between SDN data plane switching and SDR in terms of CPU utilization and parallelization, which provides insights for processing aggregation and power savings when integrating them together in a cloud-based system. We study three use cases that have different settings for the process computing loads used as input for the SDN data plane and SDR. Each use case differs with respect to the number of processes that need to be allocated to SDN and SDR and CPU utilization for each process. Based on the investigation of related works, the contribution of our work in this chapter can be summarized as follows:

- We propose an SDN and SDR integrated Network Function Virtualization (NFV) architecture that is cloud-based architecture. We investigate the power consumption of virtual switches and SDR by real testbed measurement.
- We propose a power optimization algorithm with SDN and SDR integrated CPU deployment and compare it to the case where these two functions are deployed separately.

This chapter addresses the research gap related to experimental measurements of CPU and power consumption for the integrated SDN and SDR architecture. Additionally, the measurement results presented in this chapter will be used in Chapter 5 to represent CPU

consumption for the communication components. Chapter 5 combines these results with those of the computation component (MEC) to form a use case study on CPU and power consumption optimization for our proposed SDN-SDR-MEC integrated architecture.

4.1 Introduction and Background

NFV technologies are being proposed to be used broadly in next-generation networks, especially in cloud-based network architecture, e.g., Mobile Edge Cloud (MEC), Internet of Things (IoT), etc. [79, 80]. The softwarization and virtualization of the network functions enable the opportunity for cloud technologies to be embedded in the NFV. SDR and SDN play important roles in next-generation wireless and wired networks, as together they can enable unprecedented flexibility in the capacity allocation and service differentiation [31]. However, these are conventionally deployed separately (i.e., in different servers), which can lead to suboptimal use of resources. In this chapter, we investigate the power consumption of the two technologies and propose power-saving methodologies based on the coexistence of the two paradigms on the same group of CPU cores. We go deep into the power consumption and CPU utilization by carrying out extensive testbed measurements, collecting and analyzing data of a real cloud-based system for SDN and SDR in the Trinity College Dublin OpenIreland testbed [32].

There are other works related to our work. For instance, authors in [28] presented an experimental study of power consumption of virtualized base stations (vBSs), investigating its relationship with performance. However, they do not investigate SDN or power-saving schemes. Authors in [81] proposed a dynamic resource allocation scheme in Software-Defined-Radio Access Networks based on statistical evaluations. However, they focus on the allocation of radio resources rather than CPU computational resources. Authors in [82] studied several types of functional splits in the NFV of a dual-site network in Virtualized Radio Access Networks with SDR. The paper investigated the combined optimization of the power consumption and the mid-haul bandwidth of the functional splits. However, there were no real testbed measurements involved in the paper. Authors in [83] solved an optimization problem to minimize the power consumption of SDN by Integer Linear Programming and routing constraints. However, there was no testbed measurement or experiment involved

either.

The results of our work show different CPU utilization features for SDN and SDR, which enables the feasibility of power savings of an integrated CPU deployment of these two important NFV functions. Our power saving scheme can save up to 20% power consumption compared to the approach where SDN and SDR are deployed separately. We propose an MEC as an example for a use case of SDN and SDR integrated architecture, as shown in Figure 4.1. This architecture can be, for example, adopted in systems for inter-vehicle communications [84]. The architecture is all software-defined, and network functions are virtualized as the two major network functions for this architecture, SDN and SDR functions, are integrated into the edge servers (i.e., MEC nodes). Vehicular networks are illustrated on the end-user side. Inside the MEC nodes, SDN switches, eNodeB Baseband Units (BBUs), 4G/5G core function, and computational resources for MEC tasks utilize the same group of servers. Here, we focus on the integration of the communication part, i.e., SDN and SDR components.

4.2 CPU Utilization in NFV

In this section, we investigate the CPU utilization for two Virtual Network Functions (VNFs), SDN and SDR through testbed measurements and data analysis.

4.2.1 CPU Utilization in SDN

In order to characterize the CPU utilization of SDN functions, we have set up a control plane and data plane in our testbed for SDN-based data transmission. The measurements focus on the data plane, i.e., the CPU consumption of Open vSwitches (OVS switches).

Figure 4.2 shows our experimental setup of SDN CPU utilization measurement. We use Mininet [85] for SDN and two separate virtual machines (VMs) for hosting the control plane (SDN controller) and the data plane (OVS switches and hosts), respectively. The *htop* command is used for monitoring the CPU utilization at the data plane. We use *iperf* UDP data transmission tool integrated with the Mininet to produce packet data streams. We measure the CPU utilization of the OVS switches and hosts for the data plane (in the

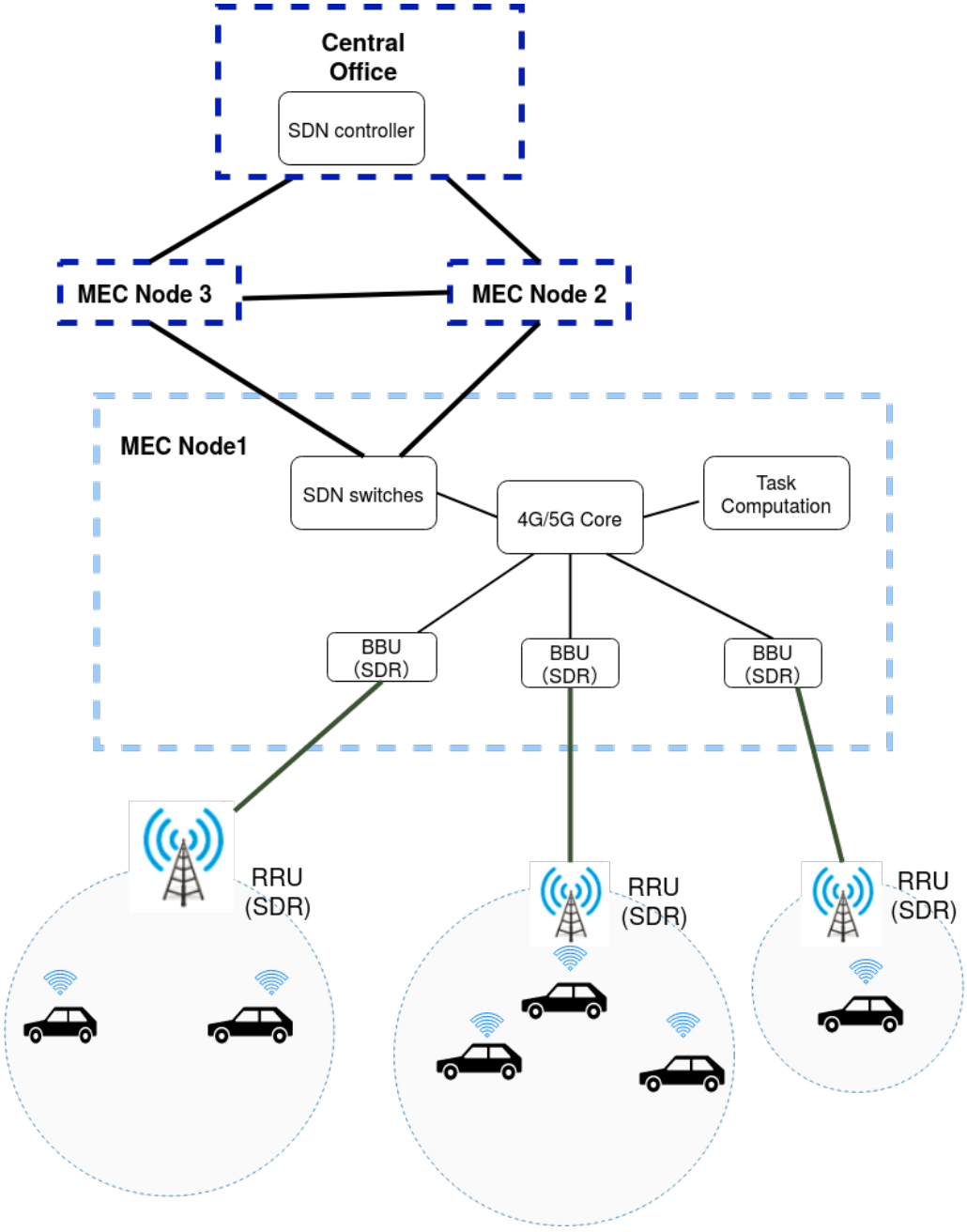


Figure 4.1: Proposed SDN/SDR integrated architecture with MEC as a use case example

Mininet virtual machine) in different data transmission rates. In the testing scenario of linear topology, we set up 5 OVS switches with one host attached to each switch, and the data transmission is from the host on the left end to the host on the right end, i.e., the route traverses 5 OVS switches. We use 2 CPU cores to host the data plane and use the *htop* program in Linux to measure the CPU utilization. The CPU model we use for the SDN experiment is *Intel Core i7-4810MQ@2.80GHz*. *htop* is a Linux command to monitor system resource utilization [86]. The results are shown in Figure 4.3, which is a boxplot

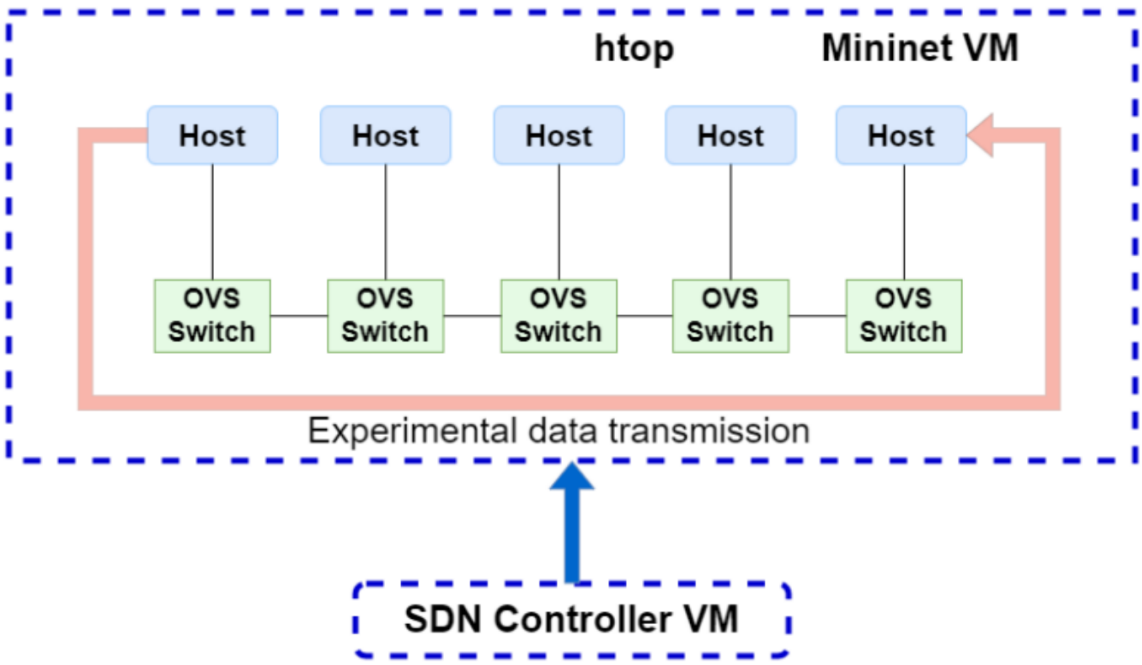


Figure 4.2: Experimental setup with linear topology

with data plots in red dots. For each data transmission rate, we measured 20 data points. As the transmission rate increases from 100kbps to 10Gbps (logarithmic scale), the CPU utilization appears to increase in a linear scale, with the mean value increasing from 20% to 175% (the maximum utilization for 2 CPU cores is 200%). Note that we are using virtual SDN switches here implemented by virtual machines. Thus, the CPU utilization is affected by the CPU processing capacity. The CPU utilization should be lower for the same data rate if we use more powerful CPUs.

4.2.2 CPU Utilization in SDR

Next, we measure the CPU utilization for an SDR with different parameter settings. We use srsRAN [6] to build a software LTE base station whose RF runs in a USRP B210. The OpenIreland testbed is located at the CONNECT center in Trinity College Dublin (whose SDR part is shown in Figure 4.4).

We have measured CPU utilization for scenarios involving the use of different bandwidths and Physical Resource Blocks (PRBs). Table 4.1 shows the results of 50 PRBs for the LTE eNodeB (our CPU is not powerful enough to carry out 100 PRB experiments without downsampling the wireless signals in the signal processing phase of SDR). We have used

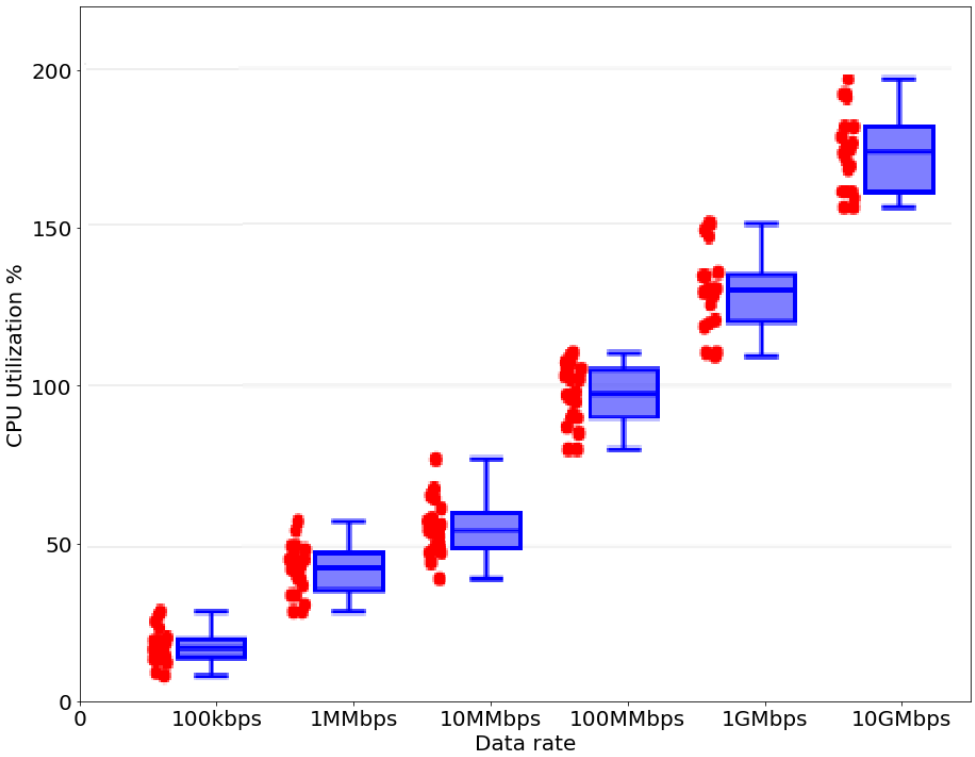


Figure 4.3: SDN CPU utilization results

the Linux application *iperf* for both UDP and TCP packet transmission tests. The CPU utilization in the table is the mean value taken from 20 experimental data points. The results show that CPU utilization increases when bandwidth settings for data transmission increase. The real throughput measured in the air is limited to around 20 Mbps, and the CPU utilization is limited to around 85%-90%. For this reason, also the packet loss for UDP and the number of retries for TCP both start to increase when the bandwidth setting increases beyond 20Mbps. Please note that we were using 16QAM as the modulation and coding schemes to carry out the experiments. The CPU model we use for the SDR experiment is *Intel NUC i7-8559U@2.70GHz*. Our CPU is not powerful enough for high modulation of 64QAM without downsampling the wireless signals in the signal processing phase of SDR.

Table 4.2 shows the results for the case of 25 PRBs. 16 QAM is again used in this scenario, and the real throughput caps at around 10 Mbps. The CPU utilization in the table is also the mean value taken from 20 experimental data points. We can observe that the CPU utilization caps at around 60% - 65% percent. This is lower compared to the 20Mbps throughput using 50 PRBs, which means that in this 25 PRB scenario, the full

Figure 4.4: Testbed picture for SDR

potential of the CPU processing is not reached. The data rate and CPU utilization are both constrained by the lower PRB numbers and low order of modulation and coding schemes.

4.3 Power Saving Methods

In this section, we investigate the power-saving methods for the proposed MEC and NFV architecture by dynamically allocating different numbers of CPUs to SDN and SDR processes.

4.3.1 Dynamic CPU Allocation and Process Parallelization

In our cloud system, we propose to use the same group of CPU cores for SDN and SDR. We investigate the parallelization features by switching from one CPU core to two CPU cores for one process of SDN or SDR, using the Linux system command in *echo* 1 >

Table 4.1: Testbed measurement results for SDR PRB 50

PRB 50	TCP			UDP		
Bandwidth setting (Mbps)	Real Throughput (Mbps)	CPU utilization (%)	Number of Retries	Real Throughput (Mbps)	CPU utilization (%)	Packet Loss
1	1	28.5	0	1	29.8	0
5	5	68.3	0	5	49	0
10	9.8	73.4	0	10	62.8	0
15	14.6	76.3	0	15	74.2	0
20	19.3	87.1	0	20	84.7	0
25	21.2	90.3	36	20.6	86.6	16
30	21.2	90.3	58	20.6	86.1	30
35	21.1	90.8	44	20.3	86	41
40	21.1	90.9	30	20.6	86.6	48
45	21.1	90.9	31	20.1	86	54
50	21.2	90.4	40	20.1	85.1	59
100	21.2	90	50	19.5	86.7	80

Table 4.2: Testbed measurement results for SDR PRB 25

PRB 25	TCP			UDP		
Bandwidth setting (Mbps)	Real Throughput (Mbps)	CPU utilization (%)	Number of Retries	Real Throughput (Mbps)	CPU utilization (%)	Packet Loss
1	1	42.1	0	1	37.4	0
5	4.99	50.3	0	4.99	51	0
10	9.46	61.1	54	9.13	62.3	8.1
15	9.46	64.4	58	9.03	60.1	39
20	9.46	63.1	57	8.92	63.3	55
25	9.46	64.1	54	8.82	63.1	64
30	9.46	63.7	58	8.74	63.1	71
35	9.46	65	53	7.71	63.3	78
40	9.46	61.7	59	7.48	62.9	81
45	9.47	59.5	58	7.21	62.1	84
50	9.46	63.6	58	6.54	63.3	86

$/sys/devices/system/cpu1/online$ and $echo\ 0 > /sys/devices/system/cpu1/online$ for adding/reducing CPU cores. We measure the CPU utilization for the same process running on one CPU core or on two CPU cores after one CPU is added to the virtual machine.

4.3.2 SDN Testing

We use Mininet to create an SDN-controlled experimental virtual network and use $iperf$ to transmit 500GB of data, with 10Gbps bandwidth in a single link on different CPU number

systems. We use the Linux command *htop* to measure the CPU utilization during our experiment. We monitor the CPU utilization rate during the transmission. When we use two CPUs to process the same SDN task, the task appears to be parallelized on the two CPUs, with a sample *htop* screenshot shown in Figure 4.5. We collect more samples (100 data points measured) of the CPU utilization to draw the box plots of the samples in Figure 4.6. The results show that SDN data plane switching processes are usually multi-threaded and can be parallelized.

Figure 4.5: CPU Utilization htop measurement for SDN

4.3.3 SDR Testing

In this subsection, we use srsRAN to create an experimental virtual network for SDR and use *iperf* to transmit with 10Mbps bandwidth. We monitor the CPU utilization rate during the transmission. When we use two CPUs to process the same SDR task, the task does not appear to be parallelized on the two CPUs, with the *htop* sample screenshot in Figure 4.7, and more samples in box plots shown in Figure 4.8. Among these two CPUs, the higher utilized CPU has almost the same utilization from the one CPU case and the lower utilized CPU has very low utilization. The results show that SDR processes are usually single-thread processes that can not be parallelized.

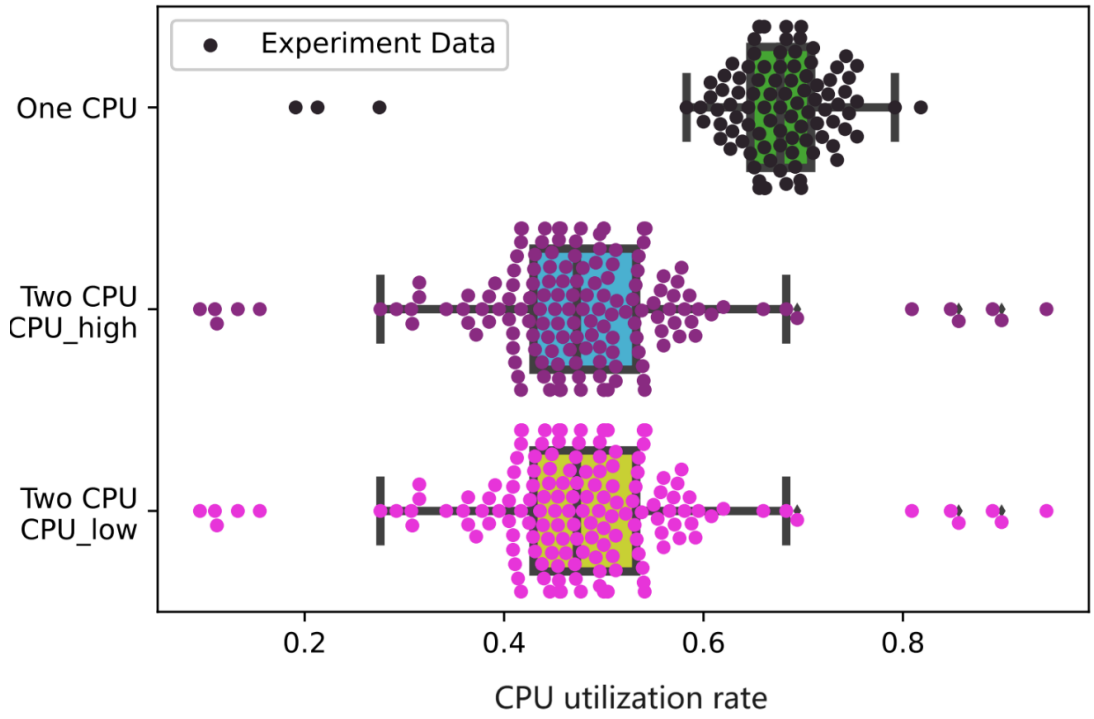


Figure 4.6: CPU Utilization rate for SDN

4.3.4 Shared CPU Utilization vs Separate CPU Utilization

In this chapter, the algorithm we design for power savings is based on the aforementioned experimental results that the SDN processes are multi-thread processes and easier to be parallelized, while SDR processes are not. This gives an opportunity to optimize the total power consumption by packing the SDN and SDR processes in the same cluster of CPU cores. Assuming the function $P(x)$ is the function for the power consumption vs CPU utilization, the objective function is the following Equation (4.1):

$$minimise : P_{total} = \sum_{c=1}^{C} P(U(c)) \tag{4.1}$$

where c is the index of the CPU core in the C number of cores. $U(c)$ is the CPU utilization percentage for the CPU core c and $U(c) \in [0, 100]$. In this equation, we don't differentiate SDN and SDR CPU utilization since they share the same group of CPU cores.

If SDN and SDR utilize the CPU cores separately, we have the total power consumption shown as the following Equation(4.2):

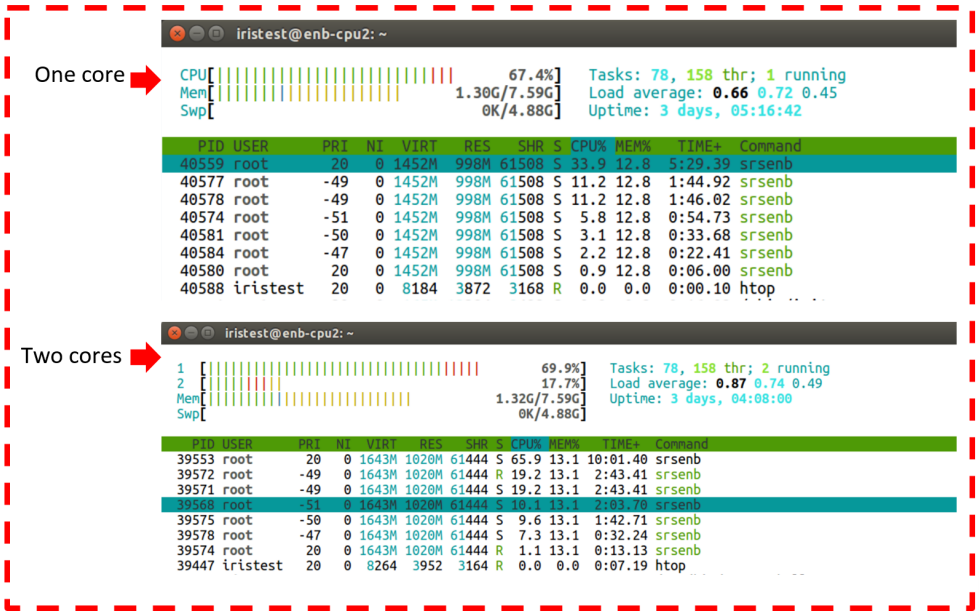


Figure 4.7: CPU Utilization htop measurement for SDR

$$P_{total} = P(U_{SDN}) + P(U_{SDR}) \tag{4.2}$$

In this equation, U_{SDN} is the CPU utilization for SDN, and U_{SDR} is the CPU utilization for SDR. In this case, the CPU utilization is decided by the processes that run separately for SDN and SDR without chances for optimization.

4.3.5 Optimization Algorithm for Power Consumption

We design the power consumption optimization algorithm for the SDN and SDR integrated system as Algorithm 3.

The algorithm is an evolved version of a best-fit algorithm. Since the SDR processes are usually single thread and not capable of being parallelized, the SDR CPU utilization comes in large chunks. On the contrary, the SDN processes come in small chunks and can be parallelized. Therefore, the algorithm tries to fit first the large SDR processes into the CPU cores and then use the remaining space in CPU cores to accommodate the SDN processes.

We have 3 use-case scenarios to test the performance of our power-saving algorithm.

- Use-case 1: a scenario where a large amount of SDR bandwidth is needed but less

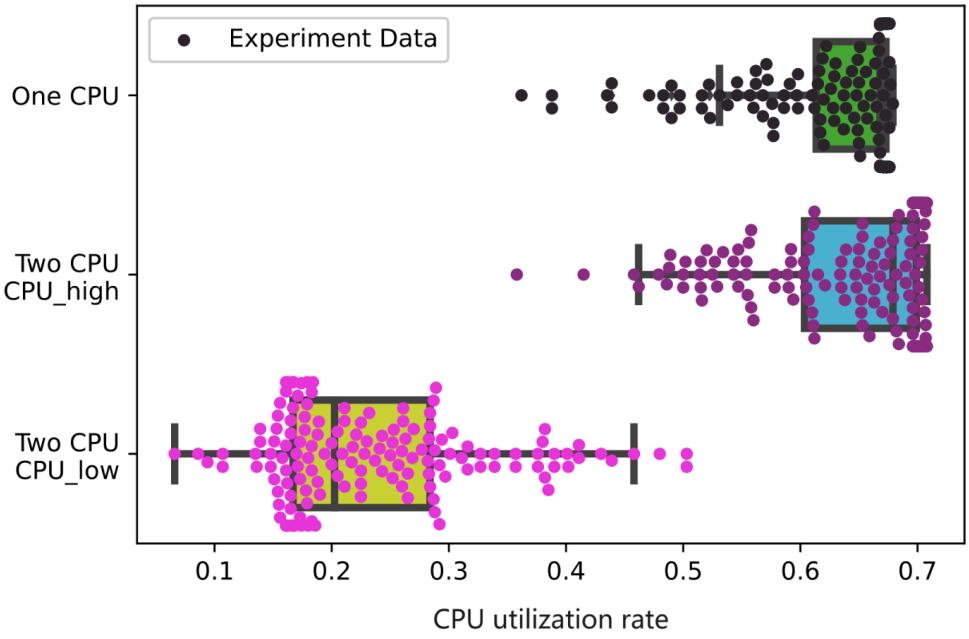


Figure 4.8: CPU Utilization rate for SDR

SDN data plane operations are needed (e.g., a local mobile network for content uploading/downloading to local servers, like mobile hotspot, Virtual Reality, Augmented Reality, etc.).

- Use-case 2: a scenario where little SDR bandwidth is needed, but a large amount of SDN data plane operations are needed, i.e., highly dynamic networking with several routing changes (e.g., mobile edge cloud and migration cases where migrations of computing tasks between edge servers are frequent, but wireless data transmission rate is low).
- Use-case 3: a scenario where both high SDR bandwidth and a large amount of SDN operations are needed for both high bandwidth and high dynamicity (i.e., remote operations, mobile video streaming etc).

Our power-saving algorithm was implemented in Python and tested through extensive simulations. Table 4.3 shows the 3 use case settings for the process computing loads, used as input for the SDN data plane and SDR simulations. Each use case differs with respect to the number of processes that need to be allocated to SDN and SDR and CPU utilization for each process. We assume CPU utilization is randomly distributed within a given range (shown in Table 4.3) with uniform distribution. Figure 4.9 shows the whole procedure of

Algorithm 3 CPU power consumption optimization algorithm

given U^i_{SDR}, U^j_{SDN}: the CPU utilization of each SDR process i and SDN process j
group all SDN processes together to be $U_{SDN} = \sum_{j=1}^{J} U^j_{SDN}$.
$i \in I$ and $j \in J$
for all CPU core $c \in C$ **do**
    Check the remaining CPU capacity of c: $1 - U(c)$
    Fit the largest SDR U^i_{SDR} into c: $U^i_{SDR} \leq 1 - U(c)$
    Get the remaining CPU capacity: $\overline{U(c)} = 1 - U(c) - U^i_{SDR}$
    Fit the $\overline{U(c)}$ with a chunk of U_{SDN} and calculate the remaining U_{SDN} by $U_{SDN} - U(c)$
▷ Comment: because SDN processes can be fully parallelized according to our testbed measurement

our simulation, including 3 main components: the stochastic task generator (for the 3 use cases mentioned above), the power saving optimization algorithm, and the CPU utilization and power consumption analysis (calculating number of CPU cores and utilization, as well as power consumption). Calculation of power consumption is based on values derived from CPU power analysis reports [87] and [88], as described below.

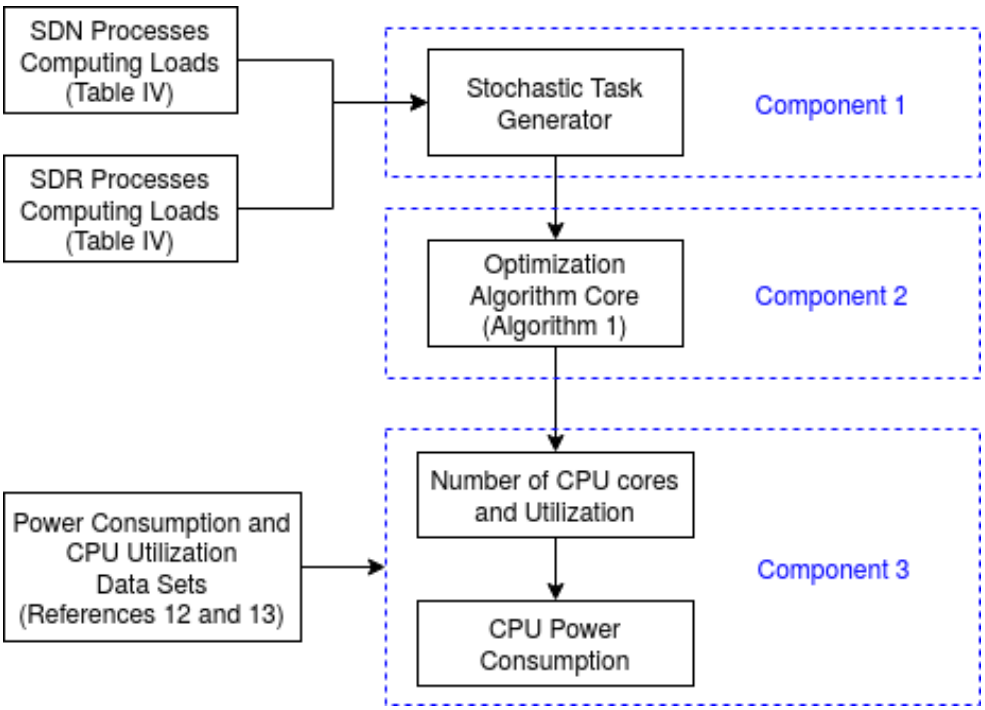


Figure 4.9: Simulation procedure

We compare the results of our algorithm with a baseline where SDR and SDN processes are allocated separately in different CPU cores. Figure 4.10 shows the boxplot of the total number of CPU cores used. Twenty independent tests are randomly generated for each use case with the parameter settings in Table 4.3. From the results, we can see that our scheme

Table 4.3: CPU process information for 3 usecases.

	SDR		SDN	
Usecase	number of processes	CPU utilization range	number of processes	CPU utilization range
Usecase 1	50	[80%, 100%]	30	[10%, 30%]
Usecase 2	30	[60%, 80%]	50	[30%, 50%]
Usecase 3	50	[80%, 100%]	50	[30%, 50%]

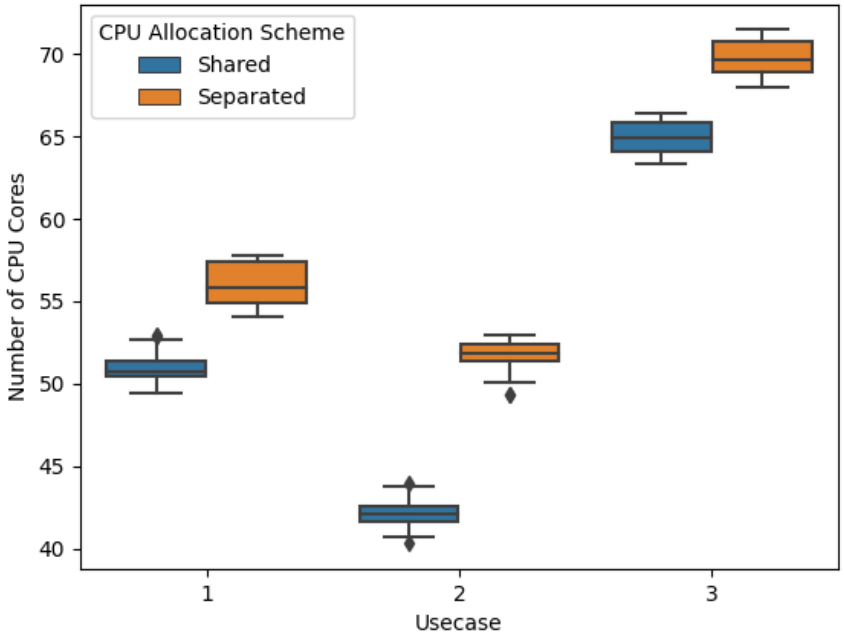


Figure 4.10: Number of CPU cores comparison

of SDN and SDR sharing the same group of CPU cores saves around 10% - 20% number of CPU cores, compared to the case of SDN and SDR utilizing CPU cores separately. The second use case has the most power-saving percentage since, in this use case, the number of SDN processes dominates, and they are easier to be parallelized, providing more room for grouping and consolidating processes in the cloud.

Figure 4.11 shows the total power consumption of these CPU cores. For this, we have used power measurement data for the server model ASUSTeK Computer Inc. RS720-E10-R12, which contains 80 CPU cores, available at [87]. Figure 4.10 and Figure 4.11 show similar trends since the power vs. CPU utilization relationship is close to linear [87]. The power saving is around 10% - 20% for this server.

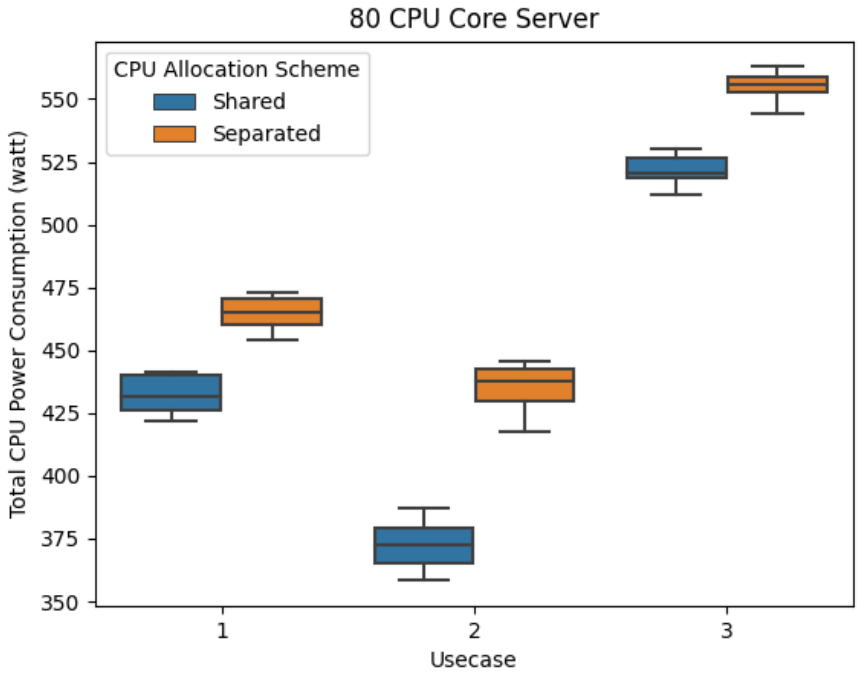


Figure 4.11: Power consumption comparison for 80 CPU core server

We also use a second power measurement dataset for a much larger server Hewlett Packard Enterprise Synergy 480 Gen10 Plus Compute Module, which contains 400 CPU cores, available at [88]. The results are shown in Figure 4.12. The trends are similar to the case of 80 CPU core systems, but the power saving percentage is lower, which is around 5% - 10% percent. This is because the idle power consumption (i.e., the overhead) of this 400-core system is already 700 watts, which is much larger than the 100 watts idle power consumption (i.e., the overhead) of the 80-core system. The results in Figure 4.11 and Figure 4.12 are also boxplots generated from 20 independent tests. There are other types of servers and CPUs that can be investigated to obtain power savings. In summary, our power saving scheme based on integrated SDN and SDR processing has up to 20% power savings compared to the case where SDN and SDR are deployed in separate CPU cores.

4.4 Conclusion

In this chapter, we have used testbed measurements with real cloud-based virtual machines to analyze the power consumption of NFV processes, in particular, SDN and SDR functions.

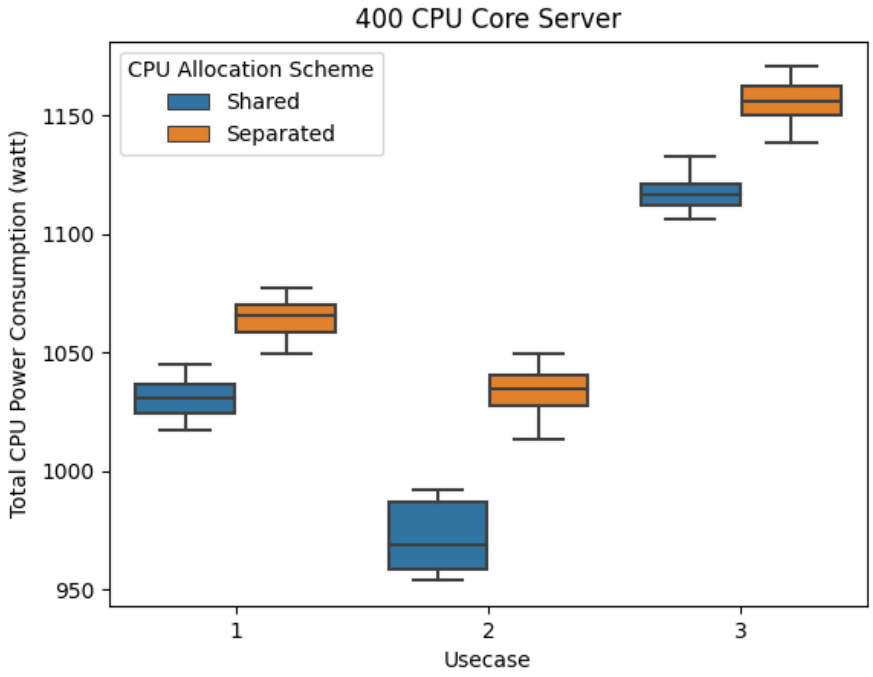


Figure 4.12: Power consumption comparison for 400 CPU core server

We have investigated CPU utilization and parallelization for different configurations and scenarios and developed an energy-saving scheme based on the data collected from the measurements. Our power-saving methods can save up to 20% power consumption compared to the case where SDN and SDR are deployed separately in the system.

5 Simulation with EdgeCloudSim with SDN, SDR and MEC Integration

(This Page Intentionally Left Blank)

Simulation with EdgeCloudSim with SDN, SDR and MEC Integration

In Chapter 3, we proposed resource-sharing algorithms, and in Chapter 4, we investigated SDN and SDR CPU consumption in our testbed. This chapter goes further to investigate a use-case study for the SDN, SDR, and MEC integrating architecture. We employed the algorithm we proposed in Chapter 3 and testbed measurement results in Chapter 4 for mobile edge cloud scenarios, sharing vehicle onboard CPU resources by offloading tasks to the neighbouring vehicles. We simulate task offloading to edge cloud, and optimize the overall CPU resource consumption, and we further focus on energy efficiency of the integrated network. Based on the investigation of related works, the contribution of our work in this chapter can be summarized as follows:

- We propose a four-tier CPU resource sharing architecture, which uses Software-Defined Networking (SDN) for communication control, Software-Defined Radio (SDR) for Vehicle to Infrastructure (V2I) communication, and Mobile Edge Cloud (MEC) for task offloading, and jointly investigate the power consumption of the whole system.
- We propose an energy-efficient cooperation strategy based on self organization, at Vehicle to Vehicle (V2V) layer to reallocate the spare resource of each vehicle, which considers vehicles' cooperation history and willingness.
- We simulate the task offloading schemes (with cutting-edge simulator EdgeCloudSim [89]) and compare our algorithm with other solutions found in the literature.

5.1 Introduction and Background

MEC has become a promising technology to tackle the computational resource shortage problem for smart city and IoT. For instance, the Intelligent Transportation System can utilize MEC as computational resources for processing cloud-based applications. NFV technologies are being proposed to be used broadly in next-generation networks, especially in cloud-based network architecture, e.g., MEC, Internet of Things (IoT), etc. [79, 80]. The softwarization and virtualization of the network functions enable the opportunity for cloud technologies to be embedded with Network Function Virtualization (NFV). Integrating the CPU resources for both the softwarized communication components and task execution components together in the same MEC server becomes feasible. In this chapter, we jointly investigate the CPU resource utilization of the communication and computation parts of this softwarized MEC network. In the previous sections, an integrated network architecture is proposed for the communication part, SDR and SDN play important roles in next-generation wireless and wired networks, as together they can enable unprecedented flexibility in the capacity allocation and service differentiation [31]. However, these are conventionally deployed separately (i.e., in different servers), which can lead to suboptimal use of resources. Chapter 3 proposed a self-organization strategy for user-preference-based resource sharing. Chapter 4 investigated the power consumption for SDN and SDR and CPU utilization by carrying out extensive testbed measurements, collecting and analyzing data of a real cloud-based system for SDN and SDR in the Trinity College Dublin OpenIreland testbed [32]. In this chapter, we further extend and complement this work as follows. We adopt the NFV's parallelization characteristics to allocate the CPU resource algorithm of Edge Servers (ESs) in our integrated architecture by making use of novel testbed measurement results (e.g., the relationship between the CPU utilization and transmission bandwidth in the use case simulation). We then investigate the overall energy savings by integrating the communication and computation CPU resources of SDN, SDR, and MEC altogether.

There are other works in the literature that are related to our work. For instance, authors in [90] wrote a survey on SDN and SDR integration on mobile ad hoc networks, discussing the state-of-the-art and research directions. Authors in [91] developed a proof of concept

of deploying SDR and SDN on edge cloud servers, investigating the wireless transmission performance, e.g., packet loss, delay, etc. Our work goes beyond this by focusing on improving CPU resource utilization and power savings. Authors in [28] presented an experimental study of power consumption of virtualized base stations (vBSs), investigating its relationship with performance. However, they do not investigate SDN or power-saving schemes. Authors in [81] proposed a dynamic resource allocation scheme in Software-Defined-Radio Access Networks based on statistical evaluations. However, they focus on allocating radio resources rather than CPU computational resources. Authors in [82] studied several types of functional splits in the NFV of a dual-site network in Virtualized Radio Access Networks with SDR. The work investigated the combined optimization of the power consumption and the mid-haul bandwidth of the functional splits. Authors in [83] solved an optimization problem to minimize the power consumption of SDN by Integer Linear Programming and routing constraints. However, there was no testbed measurement or experiment involved in either of these works. The novelty of our works is the integration of SDN and SDR in the same group of cloud servers, making use of a real testbed environment to optimize the total power consumption.

With respect to CPU resource utilization, MEC is a technology that can extend many services to the edge cloud to provide more computational capacity at lower latency. In this chapter, we focus on vehicular networks as the use case and investigate the computational tasks generated from the vehicles. Today's vehicles are equipped with On-Board Unitss (OBUs) with multiple sensors, processing units, localization systems, and radio transceivers. These embedded technologies can facilitate the setup of Vehicular Ad Hoc Network (VANET) across vehicles in the near future. However, the processing capacity of vehicles is limited, and it is difficult to execute computationally intense tasks within their own OBUs. Therefore, task offloading to ES or cloud is considered an option to increase the availability of processing power [63]. A vehicular computing system integrating vehicular networks with mobile cloud computing [84] has attracted increasing interest due to its capability to provide real-time services to onboard users.

Authors in [92] proposed a V2V partial computation offloading system to improve the completion ratio of tasks with task offloading between vehicles. However, they didn't con-

sider the willingness of vehicles for resource sharing or the possibility of offloading to the edge cloud with V2I communication. Authors in [93] explored a vehicle edge computing network architecture in which the vehicles can act as mobile edge servers to provide computation services for nearby User Equipments (UEs). Involving vehicles enhanced the scalability and flexibility of the IoT system. They discussed allocating spectrum and backhaul resources in vehicles and edge nodes. Authors in [94] proposed a resource-sharing and cooperation method for cloud-based vehicular networks, but the resource-sharing/negotiation was between Service Providers at the infrastructure layer, not between vehicles. Authors in [68] optimized the computational resources in Road Side Units (RSUs) and vehicles of the vehicular cloud computing system with a semi-Markov decision model. The computational resources in vehicular and RSU were centralized and managed in a resource pool. Therefore, no resource negotiation or cooperation between vehicles was considered. Authors in [95] built a matching-based method to offload subtasks from the user end to appropriate Edge nodes. This thesis adopts the idea of matching-based method as well.

SDN-based technologies can be widely adopted in networking systems in different domains, e.g., access, edge, core, data center networking [96–98], and also useful in V2V systems [49, 99]. The SDN controller inside the MEC server can flexibly construct the network topologies between the vehicles and realize V2V offloading dynamically. Authors in [49] proposed an architecture using SDN and MEC servers, in which the SDN controller can keep calculating and selecting the best V2V routing path between vehicles. In [100], the authors extend the architecture to multi-hop V2V connection and optimize the path base on the SDN controller deployed in MEC for both V2V and V2I task offloading. Authors in [99] proposed a vehicle trajectory prediction model to improve the efficiency of V2V task offloading by utilizing the mobility advantages of vehicles. The SDN controllers played the role of providing data support and building network topologies.

We take advantage of integrating MEC, SDN, and SDR for task offloading and include as parameters the utility for cooperation among vehicles. When we deploy these three components on the same group of CPU resources, the extent to which the integration can be implemented depends on the parallelization of the processes. In Chapter 4, we investigated the parallelization of SDR and SDN, which brings flexibility to CPU resource allocation. In

this chapter, we extend these results by adding parallelization of MEC for utilizing our ESs CPU resource. In addition, the results we obtained from our OpenIreland testbed show that SDN and SDR CPU utilization is directly related to the network bandwidth of SDN and SDR. The data we measured on the relationship of the bandwidth-CPU resource are then used as input for the MEC offloading simulation, which we carry out using EdgeCloudSim [101]. For the computational CPU consumption, we propose an energy-saving algorithm to optimize resource sharing between the users. We then investigate four different application types in our simulation, generating a number of use cases in vehicular networks with different levels of computational task loads. Our simulation results show that with resource sharing, our system can save up to 40% of power consumption (by jointly considering computation and communication aspects) compared with non-sharing strategies.

5.2 System Model

Our proposed architecture is shown in Figure 1.1. It's a four-tier architecture, which includes processing at local vehicle on-board, neighboring vehicles, edge cloud, and remote cloud. We assume each vehicle is equipped with OBU and has a certain computational ability. Each vehicle runs applications that generate tasks. In our scheme, vehicles prefer to execute tasks with the help of neighboring vehicles by V2V communications. If the tasks cannot get enough CPU resources from neighboring vehicles and cannot be executed in time, the vehicles then offload the tasks to ESs by V2I communication. Neighboring ESs are also capable of offloading tasks to each other to keep the ESs load balanced, forming a ES pool. Finally, the last option is to offload the remaining task to the remote cloud in case ESs get congested and don't have enough computational resources, especially when many demands are coming from a large number of vehicles for the ES. A Baseband Unit (BBU) and an LTE core are deployed in each ES as well. Each ES is associated with an Remote Radio Head (RRH). In our architecture, the V2V connection uses IEEE 802.11p standard [101], and the V2I connection uses the LTE standard. The reason why we picked LTE instead of the latest 5G standard is because of the limitation of available open-source 5G Distributed Units (DU) and Centralised Units (CU). Every ES has an SDN switch[102]. All connections establishment and data transmission between the ESs is controlled by the SDN controller

located in the central office.

The architecture is all software-defined, and network functions are virtualized. As the two primary network functions for this architecture, SDN and SDR functions are integrated into the edge servers (i.e., MEC nodes). Inside the MEC nodes, SDN switches, eNodeB BBUs, 4G/5G core function, and computational resources for MEC tasks utilize the same group of servers. The procedure of task offloading is handled by the MEC orchestrator, also located in the central office, adopting architectures such as those defined in [80]. In our architecture, the CPU resources of the ESs respond to execute the offloading tasks from vehicles that cannot get enough computational resources from their neighboring vehicles. Also, SDR and SDN functions occupy some CPU resources of the ESs for the communication of offloading tasks. The network functions of the proposed architecture are shown in Figure 5.1. For modeling the mobility of vehicles, we divide the whole map into several areas by RRH coverage. We define as a dwell time the short period when vehicles drive within an RRH coverage area before moving to a different area. Different locations are assumed to have different levels of dwell time for the vehicles since different areas have different average driving speeds. We have randomly distributed the vehicles across the multiple RRHs and assigned dwell times to simulate the mobility of vehicles in areas covered by different RRHs. Vehicles are assumed to move out of their RRH coverage area after the dwell time has expired, moving into the adjacent RRH coverage area for a new dwell time.

5.3 CPU Utilization in SDN and SDR

CPU power consumption is directly related to CPU utilization. Thus, we focus on CPU utilization in our testbed measurement for different experimental setups. In communication, SDR and SDN functions are the main parts utilizing CPU resources. If their functions can be parallelized and split into small parts, we can distribute those functions on different CPUs. In this section, we investigate the CPU utilization and parallelization features for two Virtual Network Functions (VNFs), SDN and SDR, through testbed measurements and data analysis, on our OpenIreland Testbed.

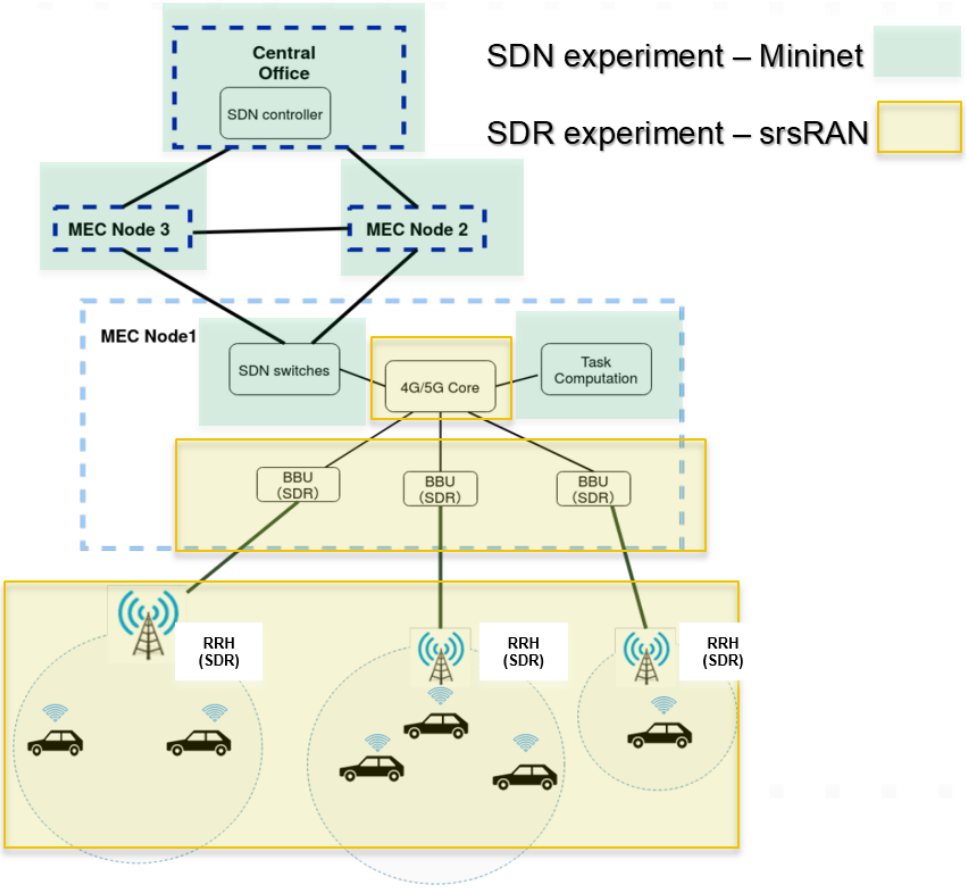


Figure 5.1: Network functions of the proposed architecture

5.4 Computation CPU Resource Sharing in MEC

In this section, we propose a V2V resource-sharing algorithm to reduce the offloading communication data size and computational task length in order to save power on Edge Nodes (ENs). In our 4-layer architecture, when a task is generated from a user/vehicle, it first checks if the local onboard CPU resource in the vehicle is sufficient or not; if not enough, it requests additional resources from neighboring vehicles through V2V communication and distributes tasks among vehicles to start their execution. If still not enough, it offloads the tasks to the MEC edge servers. The tasks distributed among edge servers are load balanced. Finally, if the MEC edge servers do not have sufficient CPU resources, the tasks are offloaded to the remote cloud.

We propose a novel CPU resource sharing strategy for energy saving in MEC task allocation, called Two-Sided(TS) sharing. The method involves two vehicles negotiating

with each other to share their spare CPU resources through V2V communication. For benchmarking, we use three other strategies, named as All-Agree(AA), No-Share(NS), and Only-Edge(OE). In the AA strategy, the neighboring vehicles always accept the resource-sharing requests from the demanding vehicle without any negotiation; in the NS strategy, the vehicles do not share resources with each other, and they only use their own onboard CPU resources; in the OE strategy, the vehicles do not use their own onboard CPU resources and offloading all tasks to the Edge servers. Here we considered two parts of CPU utilization. The CPU resources for communication (SDN and SDR, investigated in previous sections) and the CPU resources for computation (task execution).

5.4.1 Two-Sided(TS) Strategy

Matching Theory

The computational resource allocation problem can be posed as a matching problem between CPU resources and tasks generated by users. Users can be vehicles or smartphone applications. Each user owns a quota that defines the maximum number of cooperators with which it can be matched. The main goal of matching is to optimally match resources and tasks according to their preference list, which is based on their individual, often different, objectives and learned information. The preference represents the individual view of each potential cooperator based on local information. In different scenarios, the preference standard is different. In its basic form, a preference can simply be formulated by an objective utility function. The preference list of each user, in our case, builds a ranking of the CPU resource-sharing candidates (other users) using a preference relation. Matching is an essential method for allocating resources and users. The basic solution is two-sided stable matching. A matching is said to be two-sided stable if and only if there is no blocking pair (BP).

A BP for a stable two-sided stable is defined as a pair of user and resource, presented as (u, r) where u prefers r to its currently matched user j, and r prefers u to its currently matched resource k. Thus,u will leave j to be matched to r, and r would rather be matched to user u than its current matched user k. This definition of stability can extend to all

types of matching problems. The stable concept of matching in CPU resources is further discussed later.

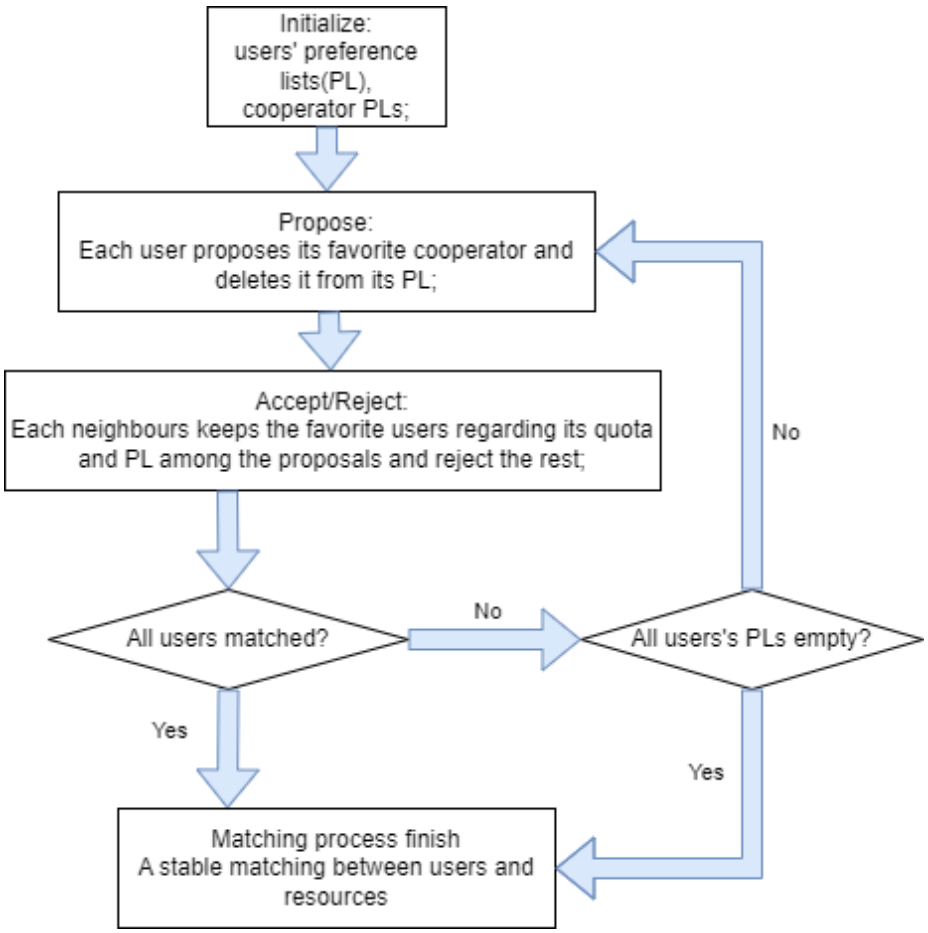


Figure 5.2: Two-Sided strategy

As mentioned above, the vehicle V_i always prioritizes the task $K(L_K, D_K)$ to be executed on V2V layer at first. Here, L_K is the task length denoted by the number of instructions, while $D_K = (D_K^{up}, D_K^{down})$ is the task upload/download data size if the tasks need to be executed somewhere else. We assume every vehicle has the same onboard CPU capacity. The spare CPU resource of the vehicle V_i at time t is represented by $C_i(t)$ (measured by instructions/second).

Step 1 - Cooperative Task Execution with Neighboring vehicles by resource sharing and V2V communication

When a vehicle V_i finds the local estimated delay $d^l(K)$ larger than the task's delay tolerance time $d_{limit,K}$, the vehicle estimates the V2V delay $d^g(K)$ for asking for the cooperation with

its geographical neighboring vehicles. In our scheme, the cooperation has the following steps:

Geographical grouping: Vehicles are grouped by their geographical locations. The SDN controller at the central office collects the information of the vehicle V_i's neighbors' geographical region and spare CPU resource $C_j(t)$ they have and sends the neighbor set $N = \{N_1, N_2..., N_j\}$ to the task owner vehicle V_i. All this information is useful for selecting cooperating neighboring vehicles.

Matching and cooperating: When the task owner is surrounded by more than one neighboring vehicle, it needs to select which ones to ask for a computational resource. The resource allocation between the V2V problem can be posed as a matching problem between the task owner vehicle and neighboring vehicles. Each task owner has a preference list to show the rank of the neighboring vehicles for the matching pair.

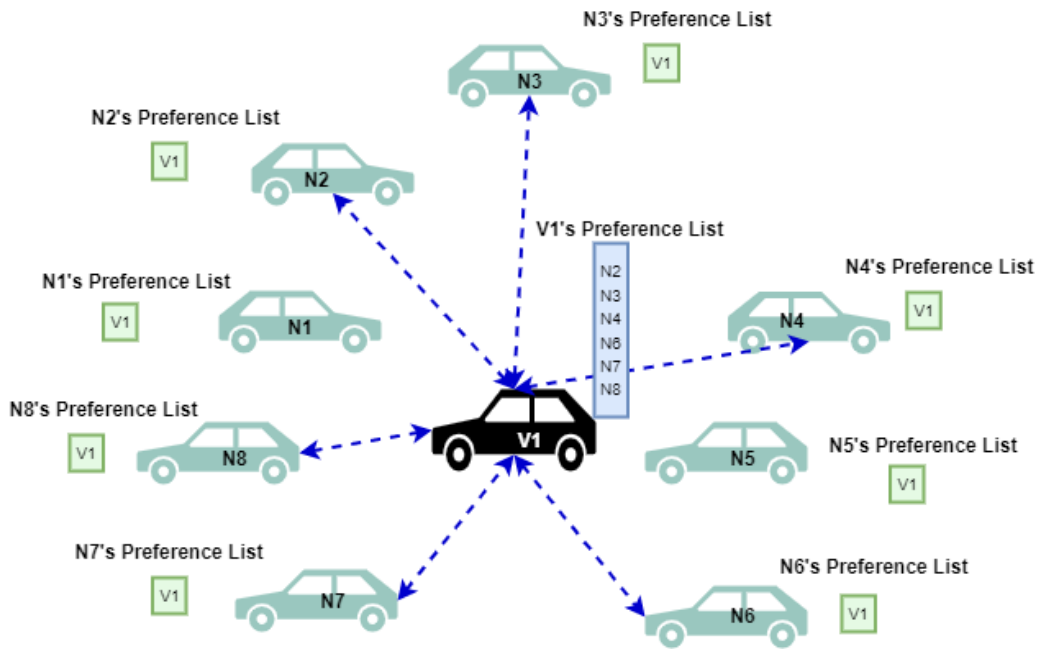


Figure 5.3: One side matching

In matching theory, there are one-to-one matching, one-to-many matching, and many-to-many matching. Our strategy scans the tasks in every time slot δt. If the system only generated one task in a time slot δt, it is a one-to-many matching. The task owner V_i selected a set of the cooperation neighboring vehicles $SetN_j$ by their utility value as Figure 5.3 shows. If more than one task were generated in a time slot δt simultaneously, we adopt

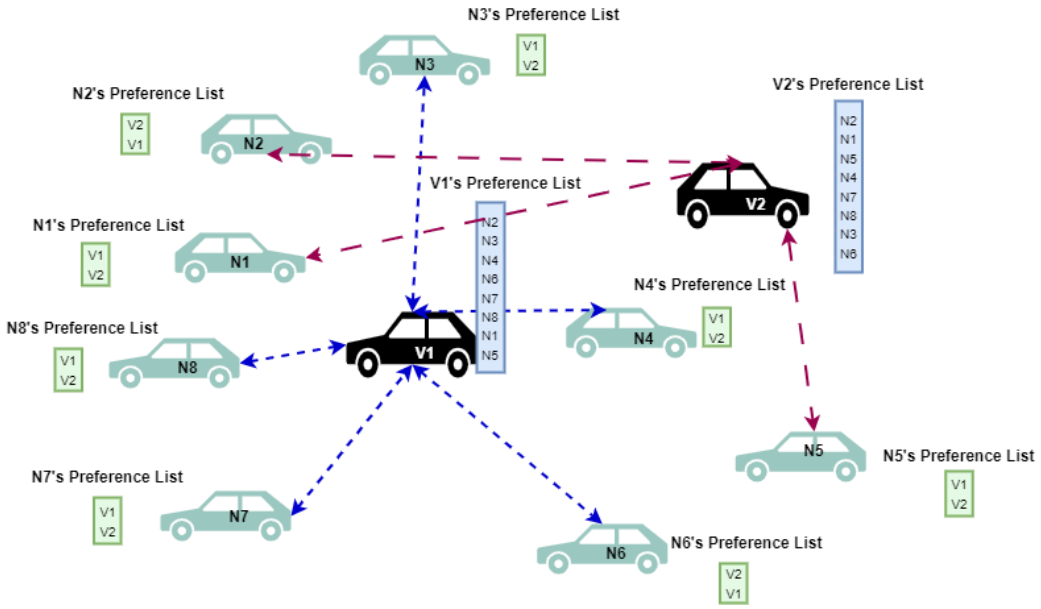


Figure 5.4: Two side matching

a Two-sided stable matching-based algorithm to form the neighboring vehicle matching for each transmission pair $\{(task_n, N_j)\}$, as shown in Figure 5.4. The task can be partitioned into up to n parts since each V_i has n channel to communicate with neighboring vehicles. For example, task K can be represented as a bunch of sub-tasks k in Equation (5.1).

$$\begin{aligned} L_K &= \sum_{k=1}^{n}(L_k); \\ D_K &= \sum_{k=1}^{n}(D_k) \end{aligned} \tag{5.1}$$

where L_k is the sub-task's computation length, and D_k is the sub-task's data length (to be transmitted to other entities for computing). The vehicle selects n neighboring vehicles to form a coalition $\{V_i, SetN_j\}$ according to their preference list.

Utility Equation: In our system, each vehicle V_i evaluates its utility value when it

forms its preference list. Our utility is defined in Equation (5.2)[103].

$$\begin{aligned} U_i(s,a,t) &= E\{J_i(s,a,t)\} = M_\theta \cdot P_{\theta i}^T \cdot J_i(s,a,t) \\ &= \begin{bmatrix} M_{\theta 11}, M_{\theta 12} \\ M_{\theta 21}, M_{\theta 22} \end{bmatrix} \cdot \begin{bmatrix} P_{\theta i}(\theta_0|\theta,t) \\ P_{\theta i}(\theta_1|\theta,t) \end{bmatrix} \cdot J_i(s,a,t) \\ &\qquad (i,j = 1,2....n \quad i \neq j) \end{aligned} \tag{5.2}$$

Where, $P_{\theta i}$ and $J_i(s,a,t)$ are derived by the following Equations[103]:

$$\begin{aligned} P_{\theta i}(\theta_0|\theta,t) = normalize(\frac{B_i(t)}{C_i^r(t)} + (\beta_{i,0}(t) - \beta_{i,1}(t))); \\ P_{\theta i}(\theta_1|\theta,t) = 1 - P_{\theta i}(\theta_0|\theta,t) \end{aligned} \tag{5.3}$$

$$\begin{aligned} J_i(s,a,t) &= P_{ai} \cdot M_a \cdot P_{aj}^T \\ &= [P_{ai}(a_0|s,t), P_{ai}(a_1|s,t)] \cdot \begin{bmatrix} Ma_{11}, Ma_{12} \\ Ma_{21}, Ma_{22} \end{bmatrix} \cdot \begin{bmatrix} P_{aj}(a_0|s,t) \\ P_{aj}(a_1|s,t) \end{bmatrix} \end{aligned} \tag{5.4}$$

It considers the current environment state and the vehicles' willingness to cooperate. The current state of vehicle V_i (i is the ID index of vehicles) at time t is denoted by s, which indicates the onboard CPU resource utilization state of the vehicle. In our case, the vehicle's resource utilization state is supposed to be in only two states: θ_0 means the vehicle has a high risk of lacking onboard CPU resources for executing its next task; θ_1 means the vehicle in low risk of lacking onboard resources. We then define the probability for the vehicle V_i in a high risk state θ_0 at time t as $P_{\theta i}(\theta_0|\theta,t)$, and low risk state as $P_{\theta i}(\theta_1|\theta,t)$ in Equation (5.3) [104]. When the vehicle uses a high percentage of its onboard CPU resources and also prefers to share its spare resource with others, it's in a 'risky' state, which means it's easier in a CPU resource-lacking situation for its own task in the short future. We use Equation (5.3) to define the risk possibility that a user can fall into a CPU resource-lacking situation during the sharing. In order to define the term $P_{\theta i}(\theta_0|\theta,t)$ as a probability for later operations, we normalized it to be within the interval $[0,1]$ [104] in Equation (5.3) $P_{\theta i}(\theta_0|\theta,t)$ is defined as a linear combination of two terms: the first term $\frac{B_i(t)}{C_i^r(t)}$ is the real-time usage $B_i(t)$ spent on real-time resource $C_i^r(t)$. The real-time

resource $C_i^r(t)$ is the total resource vehicle V_i has at time t. The real-time spare CPU resource $C_i(t) = C_i^r(t) - B_i(t)$. The second term $\beta_{i,0}(t) - \beta_{i,1}(t)$ represents the willingness of vehicle V_i to join the current round of cooperation. The cooperation willingness vector is $\beta_i(t) = [\beta_{i,0}(t), \beta_{i,1}(t)]$, $(\beta_{i,0}(t) \geq 0, \beta_{i,1}(t) \geq 0)$ Here $\beta_{i,0}(t)$ is the 'giving' tendency, which denotes the vehicle's willingness to give its resources to other vehicles at a given time, while $\beta_{i,1}(t)$ is the 'getting' tendency, which denotes the vehicle's intention to get resources from other vehicles within that same time. In our system, we assume that $\beta_{i,0}(t) + \beta_{i,1}(t) = 1$, making these two variables mutual-exclusive random variables. This means the users joining the cooperative sharing would either give or get resources from each other but can not both give and get resources at the same time [105]. With this setting, during every sharing step between two vehicles, we compare the $\beta_{i,0}(t)$ and $\beta_{i,1}(t)$ of the two vehicles. The vehicle with higher $\beta_{i,0}(t)$ gives resource to the other vehicle, and the vehicle with higher $\beta_{i,1}(t)$ gets resource from the other vehicle. If they have exactly the same $\beta_{i,0}(t)$ and $\beta_{i,1}(t)$, they don't exchange resources. The cooperation willingness vector $\beta_i(t)$ changes after each cooperation/sharing round and depends on all previous cooperation rounds of V_i. After every sharing step, $\beta_i(t)$ updates its value according to Equation (5.9). The factor $J_i(s, a, t)$ in Equation (5.4) reflects the reward that the vehicle can get from its current action [103]. We define a vehicle V_i action space as $a = [a_0, a_1]$ having two action choices, giving out resources to help others, denoted as a_0, or getting resources from others, denoted as a_1. The probability vector Pa_i represents the probability of the vehicle selecting the give or get action. $P_{ai}(a_0|s, t)$ is the probability that the vehicle chooses the action a_0, agreeing to give out its resource, while $P_{ai}(a_1|s, t)$ is the probability that the vehicle chooses the action a_1 to agree get others' resources. In our case, we assume all vehicles participate in the V2V resource sharing. Based on Game Theory, the payoff matrix M_a and M_θ are defined empirically. The vehicles are encouraged to be rewarded for cooperating with each other to execute the tasks. Therefore, our system encourages the neighboring vehicles to form a coalition with the task owner vehicle without getting into a risky environment, which might lead to a lack of CPU resources to process their own tasks. The central office gives the cooperation candidate list N^c of neighboring vehicles sorted by their utilization values to the task owner vehicle V_i. After the cooperation, the resource value of each vehicle changes, and the system begins the next scan.

After the task owner vehicle V_i gets the cooperation candidate list N^c from central office, it selects the top utility value neighbors to be cooperating candidates to execute the task $K(L_K, D_K)$, which is generated by vehicle V_i at time t. The V_i estimates the delay time $d^g(K)$ of offloading task to those cooperating candidates. This is represented in Equation (5.5). The delay term $d^g(K)$ includes communication delay $d^g_m(K)$ between vehicles and computational delay $d^g_c(K)$. We assume the tasks in the V2V layer can be partitioned. We denoted the communication data rate with b_V. The vehicle V_i selects $min(N^c, N^n)$ neighbors to cooperate. N^n is the maximum number of vehicles that can link. If the estimated execution time $d^g(K)$ is less than the task's maximum tolerable delay limitation $d_{limit,K}$, the task $K(L_K, D_K)$ will be executed on the V2V layer.

$$\begin{cases} d^g(K) = d^g_c(K) + d^g_m(K) \\ d^g_c(K) = \frac{L_K}{C^r_i(t)} \\ C^r_i(t) = C_i(t) + \sum_{j=1}^{min(N^c,N^n)} C_j(t) \\ d^g_m(K) = \frac{D^{up}_K}{b_V} + \frac{D^{down}_K}{b_V} \end{cases} \tag{5.5}$$

In Equation (5.5), L_K denotes the task length and D^{up}_K / D^{down}_K represents task upload/download data size. $C_i(t)$ is the spare CPU resource of V_i, while $\sum_{j=1}^{min(N^c,N^n)} C_j(t)$ is the sum of resources provided by each cooperating neighboring vehicle.

CPU resource Sharing strategy: To decide how many resources the cooperating neighboring vehicles would like to share with each other, here we propose a self-organization based resource sharing scheme, integrating with Asymmetric Nash Bargaining Solution (ANBS) in Game Theory[106], where neighboring vehicle N_j provides part of its spare resources for cooperation. The task owner vehicle V_i cooperates with neighboring vehicles N_j in the cooperation candidate list N^c one by one in descending order of their utility values. In each cooperation round, the cooperation candidate neighboring vehicle N_j, which adopts the TS algorithm, does not provide all its spare resources for cooperation but only part of it to process task owner vehicle V_i's offloading. We consider this cooperation as a bargain problem and assume both vehicles are rational and intend to maximize their extra resource utility in the bargain. Here, we use the strategy mentioned in Chapter 3.

The set of spare resources in the utility equation in this bargain problem can be described

as $\Gamma = \{\gamma_i \,|i = 1, 2\}\}$, where $\gamma_i = \{(C_i^r(t) - B_i(t))|i = 1, 2\}$, which is a nonempty compact convex set with boundary [74, 75]. $C_i^r(t)$ is the real-time total CPU resource for each vehicle, including its own CPU resource and the resource it gets externally, while $B_i(t)$ is the real-time resource usage. The cooperation problem is described in Equation (5.6):

$$\begin{aligned} \Gamma^* = \underset{C_i^r(t)}{argmax} \prod_i (C_i^r(t) - B_i(t))^{\lambda_i(t)} \\ s.t. \qquad \sum_{i=1}^{2} C_i^r(t) = \Phi \\ \sum_{i=1}^{2} \lambda_i(t) = 1 \\ C_i^r(t) \geq B_i(t) \\ C_i^r(t) \geq 0, \quad (i = 1, 2) \end{aligned} \tag{5.6}$$

where Φ is the total real-time resource of the vehicles considered.

$$\lambda_i(t) = \frac{\beta_{i,1}(t)}{\beta_{i,1}(t) + \beta_{j,1}(t)} \quad (i, j = 1, 2, 3...n) \tag{5.7}$$

$$C_i^r(t+1) = B_i(t) + \lambda_i(t) \cdot \sum_{i=1}^{2} (C_i^r(t) - B_i(t)) \tag{5.8}$$

$\lambda_i(t)$ denotes the bargaining power of the vehicles. In our case, the vehicles' bargaining power[75] is decided by their willingness probability vector $\beta_i(t)$, defined by Equation (5.7). At each allocation step, the vehicle gets its available real-time resource as defined in Equation (5.8).

After the cooperation, the algorithm updates the parameters of the cooperating vehicles. Part of the spare resources of neighboring vehicle N_j are provided to execute $K(L_K, D_K)$ offloaded from V_i, thus N_j's risk probability of lacking CPU resource increases. Therefore, the cooperation willingness probability vector $\beta_i(t)$ changes. In addition, the cooperation also changes the participants' bargaining power $\lambda_i(t)$, which will affect their next round of cooperation, derived from [76], defined in Equation (5.9) - Equation (5.10). t is the next

time step, $(t-1)$ is the immediate past step.

$$\beta_{i,m}(t) = \beta_{i,m}(t-1) + \alpha\Delta\beta_{i,m}(t), \quad (m = 0, 1), \quad \alpha \in (0, 1) \tag{5.9}$$

where, α is the learning rate and the $\Delta\beta_{i,m}(t)$ holds as:

$$\Delta\beta_{i,m}(t) = \frac{\Delta J_i(s, a_m, t)}{\Delta J_i(s, a_m, t) + \Delta J_i(s, a_l, t)},$$
$$\Delta J_i(s, a_m, t) = J_i(s, a_m, t) - J_i(s, a_m, t-1), \quad (m, l = 0, 1 \quad m \neq l) \tag{5.10}$$

Step 2 - Offload to Edge Servers and Remote Cloud

If the estimated execution time $d^g(K)$ is more than the task's maximum tolerable delay $d_{limit,K}$, the task $K(L_K, D_K)$ will be offloaded to ES/remote cloud. When vehicles offload tasks to ES/remote cloud, they communicate with their nearest ES. In our settings, although the ESs layer has more CPU resources than vehicles, they still have limited CPU capacity. When the ES is congested, tasks can be offloaded to the remote cloud. The delay $d^e(K)$ of V2I includes communication and computational delay, which is determined by the computational capacity of the offloading ES, shown in Equation (5.11).

$$\begin{cases} d^e(K) = d^e_c(K) + d^e_m(K) \\ d^e_c(K) = \frac{L_K}{C_E} \\ d^e_m(K) = \frac{D^{up}_K}{b_E} + \frac{D^{down}_K}{b_E} \end{cases} \tag{5.11}$$

where, L_K denotes the task length and D^{up}_K/D^{down}_K represents task upload/download data size. C_E is the CPU resource provided by ES/remote cloud.

For benchmarking, we also investigate the following 3 strategies: AllAgree (AA), Noshare(NS), and OnlyEdge(OE) Strategies, described as follows.

5.4.2 Other Strategies

All-Agree(AA) Strategy

As mentioned above, the TS strategy has a negotiation and two-sided matching between neighboring vehicles. For benchmarking, we propose another strategy where the neighboring vehicle N_j provides all of its spare resources to help process V_i's offloading task at current time t, without calculating a utility function value (as in Equations (5.2) - (5.4)) and negotiation (Equations (5.6) - (5.10), after geographical grouping). We call this reallocation algorithm All-Agree (AA) Strategy. The potential drawback of the AA approach is that vehicle N_j might become unable to process its own upcoming tasks, so it has to offload them to other vehicles or even to ES/remote cloud, which would have cost implications.

No-Share(NS) Strategy

For benchmarking, we also propose the No-Share(NS) Strategy, where the vehicles do not share onboard computation resources with each other. If the onboard computation resources are not sufficient for their needs, they offload tasks to edge servers. The potential drawback of this approach is the onboard computation resources on the vehicles are underutilized, and more tasks need to be executed on ESs, leading to more energy consumption of the Edge Servers compared to the TS and AA strategies. Since there is negligible signal propagation delay between the vehicles, the delay $d^l(K)$ only includes the computational delay shown in Equation (5.12). The unfinished tasks are directly offloaded to MEC.

$$d^l(K) = L_K / C_i(t) \tag{5.12}$$

Only-Edge(OE) Strategy

The last strategy for benchmarking is the Only-Edge(OE) Strategy, where all vehicles offload all tasks to edge servers and do not process any tasks onboard.

In summary, the whole procedure of our proposed algorithms (TS, AA, NS, and OE) is shown in Algorithm 4.

Algorithm 4 Proposed algorithm procedures

1: Initialization: task $K(L_K, D_K)$ generated by vehicle V_i
2: **if** strategy is No-Share(NS) **then**
3: estimate delay $d^l(K)$ Eqn.5.12
4: **if** $d^l(K) < d_{limit,K}$ **then**
5: executes the task locally
6: the task successfully executed
7: **else**
8: the task failed
9: **if** V2V cooperate to execute the task **then**
(A) get geographical neighbor set N
10: **if** strategy is Two-Sided(TS) **then**
11: 1) calculate each vehicle's utility value after Eqn.5.2
12: 2) select candidates list N^c from neighbor set N, by sorting by their coalition total utility value $U_{Vi} + U_{Nj}$, Eqn.5.2
13: 3) calculate the total resource V_i can get from neighbors:
14: **for** neighbor N_j in candidate list N^c **do**
15: N_j gives a part of its spare resource Eqn.5.6
16: update willingness probability $\beta_i(t)$, Eqn.5.9-5.10
17: **else if** Strategy is All-Agree(AA) **then**
18: 1) random select candidates list N^c from neighbor set N, and N_j gives all spare resource one by one.
(B) calculate the delay $d^g(K)$, Eqn.5.5
19: **if** $d^g(K) < d_{limit,K}$ **then**
20: V2V cooperates to execute the task
21: **else**
22: Offloading to Edge/remote cloud
23: **if** VM utilization < utilization threshold **then**
24: offload to lowest utilization ES
25: **else**
26: offload to remote cloud
27: calculate the delay $d^e(K)$, Eqn.5.11
28: **if** $d^e(K) < d_{limit,K}$ OR V_i change place **then**
29: the task failed
30: **else**
31: the task successfully executed
32: **if** strategy is OnlyEdge (OE) **then**
33: offload all tasks to Edge Server
34: **if** VM utilization < utilization threshold **then**
35: offload to lowest utilization ES
36: **else**
37: offload to remote cloud

The computational complexity of our proposed algorithms TS and AA is mostly affected by the sorting algorithm in Line 13, in which the algorithm selects candidates list N^c from neighbor set N by sorting candidates' utility value Equation (5.2). In our code, we use Python built-in Timesort algorithm [107], which is a hybrid sorting algorithm derived from merge sort and insertion sort, and the complexity is $O(nlogn)$. The computational complexity of NS and OE is $O(1)$ since it does not have sharing with other vehicles. They

deal with the tasks by executing them on the vehicle itself or offloading tasks to Edge Servers.

5.5 EdgeCloudSim Results

5.5.1 Real Traffic Data Analysis

Table 5.1: Summary of real traffic data analysis

	USA Highway			Korean City	
Location	I405-S	SR90-W	SR29-S	Daejon	Sejong
Average Peak Traffic flows (per minute)	98	62	21	52	15
Average Off-Peak Traffic flows (per minute)	15	12	6	12	4

We have investigated various sources of real traffic data to derive realistic input parameters for our simulations. For example, the authors in [108] analyzed real traffic data from South Korean urban areas, providing detailed data analytics, including vehicle location and speed information. Similarly, the authors in [109] conducted an investigation into California vehicle data, offering valuable insights into traffic patterns in the USA. These studies serve as the foundation for our simulation settings, which are presented in the next section.

Table 5.1 summarizes the average traffic flows for U.S. freeways and urban areas in South Korea. The average traffic flow per minute differs across cities and between peak and off-peak hours [108][109]. The variation in traffic flow can range from 0 to 100 vehicles per minute, depending on factors such as time of day and location. We use these values as input parameters for our simulations in the following sections to ensure that our models reflect real-world traffic conditions as closely as possible.

By incorporating these realistic parameters, our simulations aim to capture the variability of urban and highway traffic, providing more accurate and practical insights into resource requirements and network performance under different conditions. This approach strengthens the validity of our simulation results, making them more applicable to real-world scenarios.

5.5.2 Simulation Settings

We developed a simulator to analyze the resource-sharing algorithm. We implemented our own Python-based simulator for the V2V part, used an open-source Java-based simulator, EdgeCloudSim [89], for the V2I part, and then integrated them together.

In our simulations, the task execution can fail for two reasons. The first reason is the mobility of vehicles. If a vehicle moves out of the wireless network coverage and is not connected to the previous base station and edge server anymore, it cannot get the response to its previously requested task, and the task fails. The second reason is the delay. If a task execution cannot finish within its maximum tolerable delay, it fails.

The MEC application parameters for multiple use cases, as defined in [101], are summarized in Table 5.2. The usage percentage corresponds to the number of vehicles running the given application. The task inter-arrival time, which indicates how frequently the task generates processing load, follows an exponential distribution [101].

The maximum tolerable delay represents the time limit for task completion; if the execution time exceeds this limit, the task fails. Additionally, there are active and idle periods for task generation. During the active period, applications generate tasks at the specified inter-arrival time, while no tasks are generated during the idle period.

The upload/download data size represents the communication data size when a task is offloaded to other vehicles or to the edge/remote cloud. The task length denotes the computational quantity required for the task and is also modeled as an exponentially distributed random variable [101].

The VM utilization reflects the CPU overhead on the virtual machine when running on an ES. In the simulation, we use the Acer Incorporated Altos R380 F2 as our ENs CPU [110].

The simulation parameters are summarized in Table 5.3, including both communication network parameters and vehicle density and speed. We use the built-in nomadic mobility model from EdgeCloudSim for simulating vehicle movement [101]. In this model, different locations have varying average dwell times, representing the amount of time vehicles spend at each location. In our simulation, we categorize locations into three types, each with distinct

Table 5.2: MEC Application Parameters

	Augmented Reality	Health App	Compute Intensive	Infotainment App
Usage percentage (%)	30	20	20	30
Task arrival poison mean (s)	1	1	10	5
Maximum tolerable delay (s)	5	8	8	1
Active/Idle Period (s)	40/5	45/90	60/120	30/45
Upload/Download Data size(KB)	1500/25	1250/20	2500/200	2500/200
Task Length (GI)	9	3	45	15
VM Utilization on Edge (%)	6	2	30	10

average dwell times. Additionally, we use EdgeCloudSim's default values to configure the computational capacity for the ES layer and the remote cloud, as well as the network communication data rates.

Table 5.3: EdgeCloudSim Simulation Parameters

Parameter	Value
Simulation Time	30 minutes
WAN data rate	1 Gbps
V2I communication data rate	100 Mbps [101]
V2V communication data rate	10 Mbps [111]
CPU capacity per Vehicles/Edge/Remote Cloud	2/160/1600 GIPS
Maximun number of V2V connection N^n	6 [112]
Number of locations Type 1/2/3	1/1/2
Average dwell time in Type 1/2/3	2/5/8 minutes
TS algorithm task scan time slot δt	0.01 second
Average speed of vehicles	40 km/h
Vehicle Density (variable)	10-100 vehicles per km^2

5.5.3 Simulation Results

We investigate the performance of our TS strategy and compare it to three baseline algorithms: AA, NS and OE. We focus on the power consumption of the whole MEC system and the power savings enabled by our strategies. We include both the computation power consumption for the CPUs to execute tasks and the communication power consumption for

the CPUs to process SDN and SDR. We assume that the CPUs on the edge servers for task execution and SDN and SDR are dynamically deployed, which can be shut down when they are not in use. On the other hand, the onboard CPU processors in vehicles can not be shut down since these processors are also involved in other essential controlling functions of the vehicles for driving [113]. Therefore, there is very little power-saving potential for the onboard CPUs of the vehicles, thus, the strategies that fully utilise the CPU resources on the vehicles while shutting down more CPU resources on the edge servers save more power.

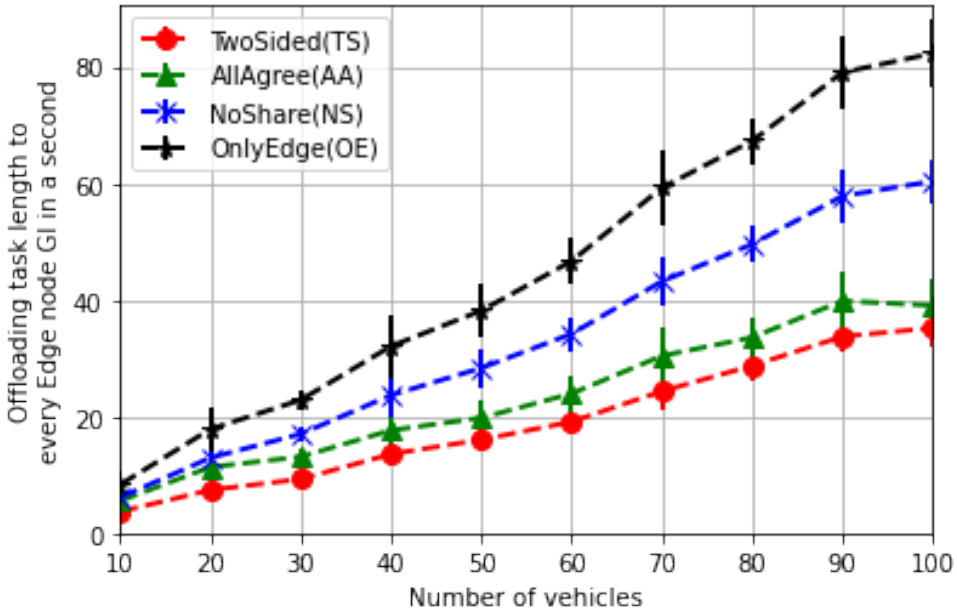


Figure 5.5: Average offloading task length for each edge node (related to computation power consumption).

Figure 5.5 illustrates the average task length offloaded to each edge node, including any potentially failed tasks. Task length is measured in GI (Giga Instructions, as detailed in Table 5.2), representing the number of instructions the CPU must execute for the task. This metric is directly related to the CPU processing time of the edge servers, whose capacities are measured in million instructions per second (MIPS), and thus impacts computational power consumption [114]. The results indicate that the TS strategy has the shortest task lengths offloaded to edge servers, followed by AA, NS, and OE, which has the highest. This suggests that the TS strategy requires the least computational power on the edge servers, leading to the highest energy savings for edge computing.

Figure 5.6 shows the average total data size in offloading to each ES in one minute. We

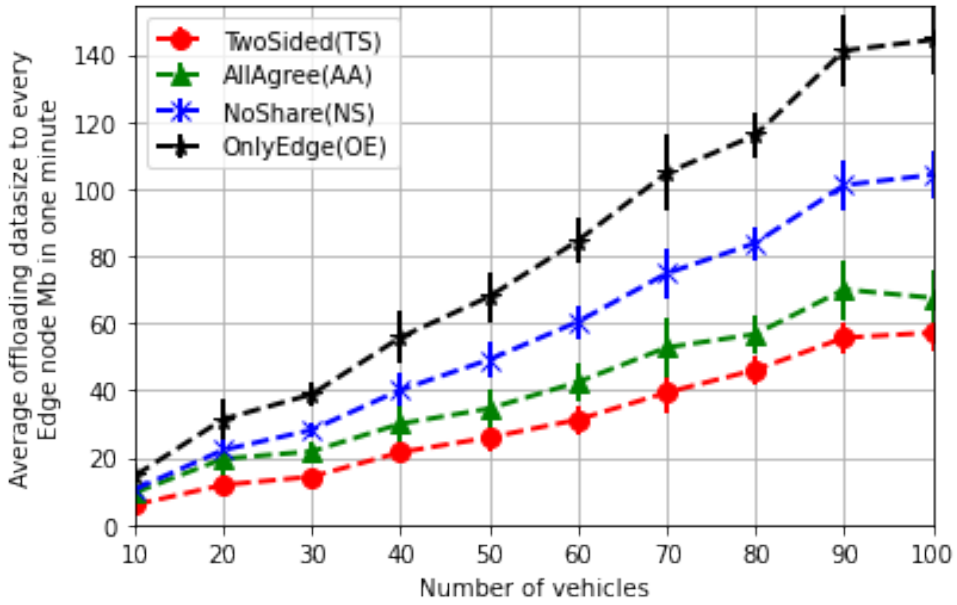


Figure 5.6: Average offloading data size to every edge node in one minute (related to communication power consumption).

can see that TS has the smallest data size among the four strategies. AA is the second least, NS is the third least, and OE is the largest. The data size is the data that needs to be transmitted between vehicles and edge servers. Therefore TS consumes the least power consumption for communication, i.e., saves the most power for communication.

Figure 5.7 presents the normalized total power consumption comparison, including both computation (MEC) and communication (SDN and SDR) power consumption for the four strategies. The TS strategy can save up to 60% of total power consumption on the edge servers compared to OE, with savings increasing as the number of vehicles rises.

We observe that reducing computation on edge servers leads to lower overall power consumption. This is because: 1) The onboard CPUs in vehicles cannot be shut down, meaning there is limited potential for power savings in vehicles. By fully utilizing vehicle CPU resources and shutting down more edge server CPUs, significant power savings can be achieved; 2) When fewer tasks are offloaded to the edge cloud and more processing is done by vehicles, communication power consumption between vehicles and the cloud is also reduced.

It is important to note that the calculations for SDR and SDN CPU utilization and power consumption for communication resources are based on our testbed experiment data

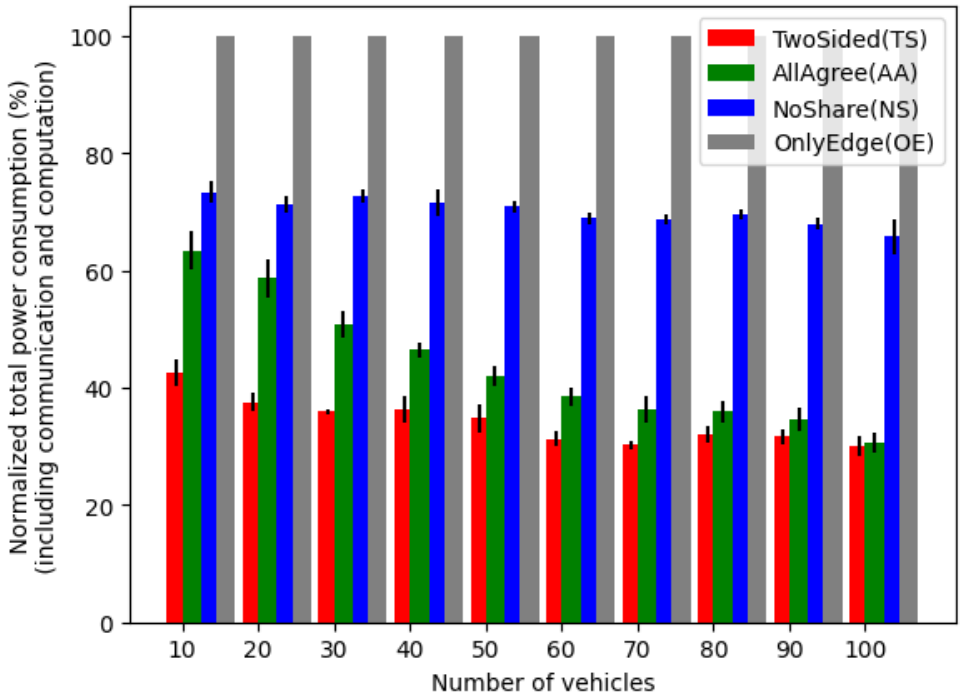


Figure 5.7: Normalized total power consumption (including both computation and communication power consumption)

discussed in previous sections.

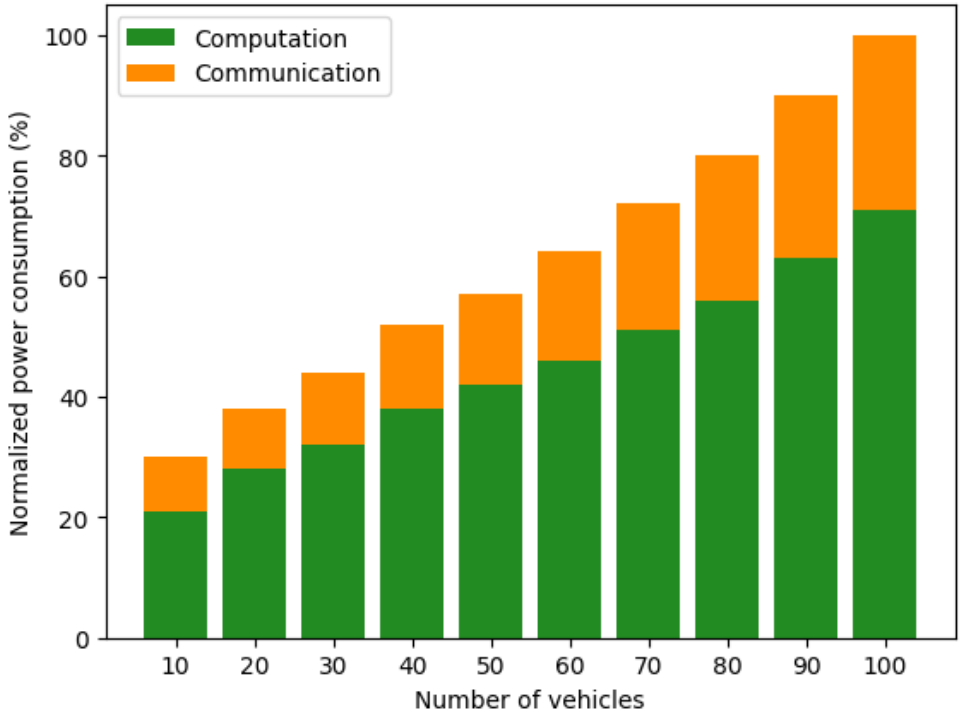


Figure 5.8: Computation vs communication power consumption, normalised

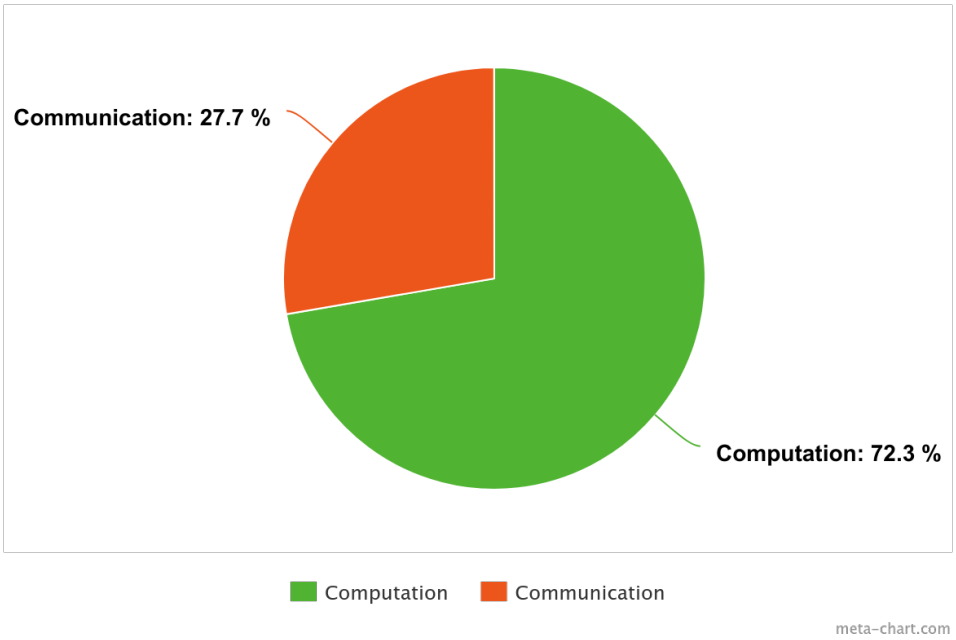


Figure 5.9: Average percentage of computation and communication power consumption

Figure 5.8 compares the normalized computation (MEC) and communication (SDN and SDR) power consumption across different numbers of vehicles in the network, with the 100-vehicle case set as 100%. The increase in power consumption is nearly linear with the number of vehicles, and the ratio between communication and computation power consumption remains almost consistent. Figure 5.9 illustrates the average proportion across all cases, with computation accounting for 72.3% and communication for 27.7% of the total power consumption.

5.6 Integrated CPU Utilization vs. Separated CPU Utilization

The algorithm we design for power savings in SDN and SDR is based on the aforementioned experimental results that the SDN processes are multi-thread and easier to be parallelized, while SDR processes are not. This gives an opportunity to optimize the total power consumption by packing the SDN and SDR processes in the same cluster of CPU cores. Assuming the function $P(x)$ is the function for the power consumption vs. CPU utilization, the objective function is the following Equation (5.13):

$$minimise : P_{total} = \sum_{c=1}^{C} P(U(c)) \tag{5.13}$$

where c is the index of the CPU core in the C number of cores. $U(c)$ is the CPU utilization percentage for the CPU core c and $U(c) \in [0, 100]$. In this equation, we don't differentiate SDN and SDR CPU utilization since they share the same group of CPU cores.

If SDN and SDR utilize the CPU cores separately, we have the total power consumption shown as the following Equation(5.14):

$$P_{total} = P(U_{SDN}) + P(U_{SDR}) + P(U_{MEC}) \tag{5.14}$$

In this equation, U_{SDN} is the CPU utilization for SDN, U_{SDR} is the CPU utilization for SDR, and U_{SDR} is the CPU utilization of MEC. In this case the CPU utilization is decided by the processes that run separately for SDN and SDR without chances for optimization.

5.6.1 Optimization Algorithm for Integrated Power Consumption

Algorithm 5 CPU power consumption optimization algorithm

given $U^i_{SDR}, U^j_{SDN}, U^j_{MEC}$: the CPU utilization of each SDR process i and SDN process j
group all SDN processes together to be $U_{SDN} = \sum_{j=1}^{J} U^j_{SDN}$, and $U_{MEC} = \sum_{k=1}^{K} U^k_{SDN}$.
$i \in I$ and $j \in J$
for all CPU core $c \in C$ **do**
    Check the remaining CPU capacity of c: $1 - U(c)$
    Fit the largest SDR U^i_{SDR} into c: $U^i_{SDR} \leq 1 - U(c)$
    Get the remaining CPU capacity: $\overline{U(c)} = 1 - U(c) - U^i_{SDR}$
    Fit the $\overline{U(c)}$ with a chunk of U_{SDN} and calculate the remaining U_{SDN} by $U_{SDN} - U(c)$
    Fit the $\overline{U(c)}$ with a chunk of U_{MEC} and calculate the remaining U_{MEC} by $U_{MEC} - U(c)$ ▷ Comment: because SDN processes can be fully parallelized according to our testbed measurement

We design the power consumption optimization algorithm for the SDN, SDR, and MEC integrated system as Algorithm 5. The algorithm is an evolved version of the idea of a best-fit algorithm. Since the SDR processes are usually single thread and not capable of being parallelized, the SDR CPU utilization comes in large chunks. On the contrary, the SDN processes come in small chunks and can be parallelized. For the computational part,

the MEC task execution can be parallelized too. Therefore, the algorithm tries to fit first the large SDR processes into the CPU cores and then use the remaining space in CPU cores to accommodate the SDN processes and MEC task execution.

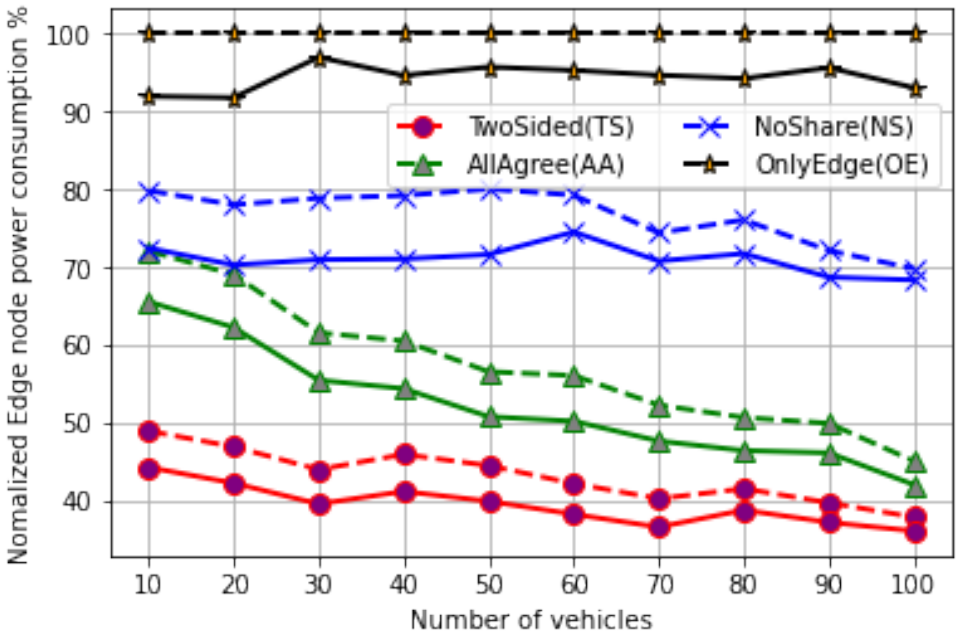


Figure 5.10: Normalized total power consumption on edge servers. Solid lines represent the four V2V algorithms with the integration of MEC and NFV together; Dash lines represent the four V2V algorithms without integration.

Finally, we show the whole system power consumption, including both computation power consumption and communication power consumption, in Figure 5.10. We also show the improvement of integrating SDN, SDR, and MEC in the same group of CPUs compared to the case of deploying them separately. Our results show that TS saves up to 60% of power consumption, compared with the OE strategy. NS saves about 30% and AA can save up to 55%, but is only as effective as TS in the cases of a large number of vehicles involved. Solid lines represent the four algorithms with the integration of SDN, SDR, and NFV together; Dash lines represent the four algorithms without integration. The integration of SDR, SDN, and MEC helps to save power consumption up to an additional 5%.

5.7 Conclusion

In this Chapter, we proposed an architecture for the integration of SDN, SDR and MEC for a V2V and V2I offloading scenario. We have used testbed measurements with real cloud-based virtual machines to analyse power consumption of NFV processes, in particular, SDR and SDN data plane functions. We have investigated the CPU utilization and parallelization for different configurations and scenarios. Using the data collected, we developed an energy saving scheme and performed a case study with vehicles executing and offloading tasks. Our results show that our resource-sharing approach saves up to 60% power consumption (by jointly considering both computation and communication power consumption), compared with non-sharing strategies, without compromising the completion of the tasks generated by the users.

6 Conclusion

This thesis explored CPU resource allocation in the context of MEC leveraging virtualization technologies. First, we proposed a self-organized algorithm for resource sharing in cloud environments. Next, we conducted SDR and SDN experiments on our testbed to investigate their CPU utilization. Finally, we proposed a four-tier architecture integrating SDR, SDN, and MEC technologies and evaluated the energy savings of this architecture.

To the best of our knowledge, this thesis is the first to investigate CPU resource allocation in an integrated SDN, SDR, and MEC environment. It serves as an example for researchers aiming to conduct comprehensive end-to-end studies on this topic using theoretical, testbed, and simulation approaches.

6.1 Summary

In Chapter 1, we outlined the motivation behind this thesis, introducing the research problem and main contributions, with a focus on CPU resource allocation in the proposed integrated network environment.

In Chapter 2, we provided an overview of the research background and related work, summarizing the research gaps and highlighting our contributions beyond the current state of the art.

In Chapter 3, we presented our proposed algorithms for CPU resource sharing. We integrated the concepts of self-organization and personality traits to design a system for resource sharing in a cloud environment. We proposed an algorithm named EMP with Asymmetric Nash Bargaining strategy (EMP-A), along with three additional algorithms: EMP with Fixed step strategy (EMP-F), Single Evolution Multi-robots Personality (SEMP), and Nash

Bargaining Solution Sharing (NBSS), for comparison. Additionally, we included the Cloud Cooperative-Federation Sharing (CCFS) algorithm from the literature as a baseline. All the algorithms facilitate the allocation of spare resources to users in need, enabling resource sharing without requiring additional external resources. This chapter introduces a distributed resource-sharing approach that can serve as a guide for researchers interested in applying user preferences to network resource-sharing problems.

In Chapter 4, we discussed the energy-efficient SDN and SDR joint adaptation of the CPU utilization scheme. We have used testbed measurements with real cloud-based virtual machines to analyze the power consumption of NFV processes, in particular, SDN and SDR functions. We investigated CPU utilization and parallelization for different configurations and scenarios. We also developed an energy-saving scheme based on the data collected from the measurements. Our power-saving methods can save up to 20% power consumption compared to the case where SDN and SDR are deployed separately. This chapter provides real measurement data as a reference for researchers who do not have SDN and SDR experimental conditions. In particular, the investigation of the parallelization features of SDN and SDR processes in our testbed provides an important contribution to this research area.

In Chapter 5, we proposed an architecture for the integration of SDN, SDR, and MEC for a V2V and V2I offloading scenario. We simulate this proposed architecture with EdgeCloudSim, using the algorithm we proposed in chapter 3 to solve the CPU resources allocation at V2V layer. Our simulation also employed the processing SDR and SDN CPU utilization data that was measured in chapter 4. We developed an energy-saving scheme and performed a case study with vehicles executing and offloading tasks. Our results show that our resource-sharing approach saves up to 60% power consumption (by jointly considering both computation and communication power consumption), compared with non-sharing strategies, without compromising the completion of the tasks generated by the users. This chapter provides a solution for the integrated simulation of a vehicle-network-based mobile edge cloud system. Researchers can use this as an example case study to build future experiments.

We addressed the research problems raised in Section 1.2 as follows:

- *What are the benefits of integrating SDN, SDR, and MEC within the same cloud*

infrastructure? - The integration provides: 1) more flexible control over the entire network, covering both data transmission and task execution, and 2) optimized overall CPU resource utilization for both communication and computation components of the network.

- *How to design and optimize a CPU resource-sharing scheme using this integrated network architecture?* - In this thesis, we reviewed state-of-the-art literature on integrating SDN and SDR, as well as MEC and NFV. We also incorporated vehicle-to-vehicle communication schemes. Based on this research, we proposed a four-layer resource-sharing and task-offloading scheme.
- *What is the relationship between network performance (e.g., bandwidth, data rate, etc.) and CPU utilization for SDN and SDR and how does this affect CPU resource sharing?* - We investigated the relationship between network performance and CPU utilization through experimental measurements in our OpenIreland testbed, as detailed in Chapter 4. These results were then used as input for the resource-sharing simulations in Chapter 5.
- *How do user behaviour and preferences influence CPU resource-sharing outcomes?* - In this thesis, we introduced personality traits into our self-organization algorithm to analyze the impact of user behaviour and preferences. We utilized a game-theory-based approach for resource sharing, considering personality traits to ensure user satisfaction throughout the process.
- *How much energy can be saved by through the integration of SDN, SDR, and cloud infrastructure with CPU resource sharing?* - Using EdgeCloudSim, we simulated vehicular network scenarios with MEC, considering communication range and traffic constraints of vehicles using synthetic data. The results indicate that CPU resource sharing can lead to up to 60% energy savings.

6.2 Open Issues and Future Work

As mentioned above, in this thesis, we made contributions to the challenges of CPU resource sharing in the cloud environment. However, there are still many open issues in this research area that need to be solved, providing a decent amount of possibilities to continue research

along this line. In this section, we give a hint at the open issues, as well as possible future work directions in this research area. We have categorized the following two aspects: 1) SDN, SDR and MEC integration and 2) CPU resource allocation.

6.2.1 SDN, SDR and MEC Integration

One of the open issues for the integration of SDN and SDR in this research area is control plane CPU resource sharing. In this thesis, we have investigated the CPU resource sharing between SDN and SDR, mainly on the data plane. The CPU resources for the SDN control plane are considered separately, and the SDR used in this thesis does not have a control plane. In Open RAN, the RAN Intelligent Controller (RIC) is the technology of choice to build an intelligent control plane. We believe an important research are will be the investigation the CPU resource utilization of the RIC and x/rApps running the state-of-the-art algorithms, e.g., machine learning and optimization algorithms. It will be important to analyze the computational complexity of these algorithms and test the CPU utilization on our testbed. Similarly, the investigation of CPU resource utilization for more sophisticated routing algorithms for SDN controllers is a key research target in this area, to investigate its energy saving potential.

Another technical challenge of integrating SDN and SDR is the limitation of the computational capability of the hardware platforms. 5G and future 6G SDR platforms require high-speed signal processing, especially for the physical layer. For building the full-stack SDR in our testbed, in this thesis, we have been restricted to using 4G LTE due to the limited availability of a compatible open source 5G software stack. In future work, in order to set up SDR testbed experiments for 5G and beyond, the processing platforms could be separated into different options of functional splits according to our hardware availability. For example, using Field Programmable Gate Array (FPGA) or other high-speed computational platforms for physical layer signal processing, and CPU for MAC layer and up. If this is the case, only the part of SDR that uses CPU processing can be integrated with SDN's CPU resources in the cloud. Therefore, how to properly select the functional split point for SDR to integrate with SDN will become an interesting research problem for 5G and future 6G networks.

When considering integrating MEC together with SDN and SDR, another open issue is vehicle mobility in real life. From the planning and provisioning perspective, the real traffic data of vehicle mobilities are essential for deploying the SDN, SDR and MEC infrastructures at the roadside. Telematics data such as speed, location and vehicle densities would affect our decisions for resource sharing. For example, the selection of communicating vehicles or base stations are affected. In this thesis, we have used synthetic data for simulation with EdgeCloudSim for vehicle computational task offloading, in future work, we will try to collect real telematics data from vehicles or obtain the data from a third party to make our simulation results more realistic.

6.2.2 CPU Resource Allocation

Cloud and virtualization technologies enable dynamic CPU resource allocation. Therefore a container and/or VM based architecture needs to be in place to support the infrastructure sharing the CPU resources. In our OpenIreland testbed, we use OpenStack-based technologies to manage container and VM resource assignments. The number of CPUs are dynamically allocated according to the requirement of our experiment. In this thesis, the whole allocation procedure in the OpenStack environment is pre-defined, and not automatically triggered by our algorithms. Since we have designed and implemented self-organization methods in our simulator and obtained promising results, the next step forward to make more contributions to the research community is to design and implement a prototype to automatically execute our algorithms in OpenStack-based cloud testbed environment. The auto-scaling of CPU resources in the testbed in real cases would potentially bring interruptions for task executions, which is interesting to be looked at in academia and even the industry cloud as well.

Furthermore, in this thesis, we assume that each user in the system participates fairly in personality-trait-based resource sharing, which is not always reflective of real-world scenarios. The security and robustness of the system in the presence of malicious users have become an emerging research challenge. Malicious users could potentially dominate or disrupt the entire resource-sharing system if it is not securely designed. Ensuring secure CPU allocation and utilization is crucial for maintaining the health and stability of the cloud

system. In our future work, we will investigate malicious user behaviours, analyze their potential impact on CPU resource sharing, and develop strategies to mitigate damage and prevent losses to the cloud system.

Acronyms

ANBS Asymmetric Nash Bargaining Solution

BBU Baseband Unit

BP blocking pair

C-RAN Cloud Radio Access Network

CCFS Cloud Cooperative-Federation Sharing

EMP Evolutionary Multi-robots Personality

EMP-A EMP with Asymmetric Nash Bargaining strategy

EMP-F EMP with Fixed step strategy

EN Edge Node

ES Edge Server

FPGA Field Programmable Gate Array

IaaS Infrastructure as a Service

IoT Internet of Things

MEC Mobile Edge Cloud

NBS Nash Bargaining Solution

NBSS Nash Bargaining Solution Sharing

NFV Network Function Virtualization

OBU On-Board Units

PRB Physical Resource Block

PT Personality Traits

QoE Quality of Experience

RIC RAN Intelligent Controller

RRH Remote Radio Head

RSU Road Side Unit

SDN Software-Defined Networking

SDR Software-Defined Radio

SEMP Single Evolution Multi-robots Personality

V2I Vehicle to Infrastructure

V2V Vehicle to Vehicle

VANET Vehicular Ad Hoc Network

VNF Virtual Network Function

UE User Equipment

Bibliography

[1] Y. Zhang, F. Barusso, D. Collins, M. Ruffini, and L. A. DaSilva, "Dynamic allocation of processing resources in cloud-ran for a virtualised 5g mobile network," in *2018 26th European Signal Processing Conference (EUSIPCO)*, 2018, pp. 782–786.

[2] H. Hawilo, A. Shami, M. Mirahmadi, and R. Asal, "Nfv: state of the art, challenges, and implementation in next generation mobile networks (vepc)," *IEEE Network*, vol. 28, no. 6, pp. 18–26, 2014.

[3] T. Kaitz and G. Guri, "Cpu-mpu partitioning for c-ran applications," in *7th International Conference on Communications and Networking in China*, 2012, pp. 767–771.

[4] OpenStack. An OpenInfra Foundation Project. Accessed: Aug. 2023. [Online]. Available: https://www.openstack.org/

[5] OpenAirInterface. 5G software alliance for democratising wireless innovation. Accessed: Aug. 2022. [Online]. Available: https://openairinterface.org/

[6] srsRAN. Software Radio System. Accessed: Aug. 2022. [Online]. Available: https://www.srslte.com/

[7] T. Salman, "Cloud ran : Basics , advances and challenges a survey of cran basics , virtualization , resource allocation , and challenges," in *semanticscholar*, 2016.

[8] ETSI, "Mobile Edge Computing (MEC); Deployment of Mobile Edge Computing in an NFV environment," Feb 2018, Accessed: Aug. 2021. [Online]. Available: http://www.etsi.org/deliver/etsi_gr/MEC/001_099/017/01.01.01_60/gr_MEC017v010101p.pdf

[9] ——, "Cloud RAN and MEC:A Perfect Pairing ," Feb 2018, Accessed: Aug. 2021. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp23_MEC_and_CRAN_ed1_FINAL.pdf

[10] C. Liang, F. R. Yu, and X. Zhang, "Information-centric network function virtualization over 5g mobile wireless networks," *IEEE Network*, vol. 29, no. 3, pp. 68–74, 2015.

[11] T. Ahmed, A. Alleg, and N. Marie-Magdelaine, "An architecture framework for virtualization of iot network," in *2019 IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 183–187.

[12] Ryu, "Ryu sdn framework," Accessed: Feb. 2021. [Online]. Available: https://ryu-sdn.org/

[13] The Linux Foundation. OpenDaylight Project. Accessed: Aug. 2023. [Online]. Available: https://www.opendaylight.org/

[14] S. Sun, M. Kadoch, L. Gong, and B. Rong, "Integrating network function virtualization with sdr and sdn for 4g/5g networks," *IEEE Network*, vol. 29, no. 3, pp. 54–59, 2015.

[15] M. Klymash, H. Beshley, A. Masiuk, and I. Strykhalyuk, "Concept for ensuring effective functioning of mobile communication system in heterogenous 5g infrastructure," in *2017 14th International Conference The Experience of Designing and Application of CAD Systems in Microelectronics (CADSM)*, 2017, pp. 272–274.

[16] M. Erel-Ozcevik and F. Tekce, "Sdn/nfv based secure scma design in sdr," in *2021 17th International Conference on Network and Service Management (CNSM)*, 2021, pp. 319–325.

[17] G. C. Valastro, D. Panno, and S. Riolo, "A sdn/nfv based c-ran architecture for 5g mobile networks," in *2018 International Conference on Selected Topics in Mobile and Wireless Networking (MoWNeT)*, 2018, pp. 1–8.

[18] H. Yu, C. Ye, Y. Yuan, Y. Sun, B. Guo, and X. Zhang, "Implementation of c-ran architecture with cu-cp and cu-up separation based on sdr/nfv," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, 2019, pp. 1–5.

[19] E. R. Machado, M. Feldman, and I. MÃ¼ller, "A container-based architecture to provide services from sdr devices," in *2023 IEEE 21st International Conference on Industrial Informatics (INDIN)*, 2023, pp. 1–6.

[20] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: https://doi-org.elib.tcd.ie/10.1145/1355734.1355746

[21] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "Estinet openflow network simulator and emulator," *IEEE Communications Magazine*, vol. 51, no. 9, pp. 110–117, 2013.

[22] M. Soursouri and M. Ahmadi, "Adaptive resource allocation for software defined networking controllers," *Journal of High Speed Networks*, vol. 23, 06 2017.

[23] N. Deric, A. Varasteh, A. Van Bemten, A. Blenk, and W. Kellerer, "Enabling sdn hypervisor provisioning through accurate cpu utilization prediction," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1360–1374, 2021.

[24] B. Velevski, V. Rakovic, and L. Gavrilovska, "Implications of network resources and topologies over sdn system performances," in *Future Access Enablers for Ubiquitous and Intelligent Infrastructures*, O. Fratu, N. Militaru, and S. Halunga, Eds. Cham: Springer International Publishing, 2018, pp. 25–31.

[25] P. Isaia and L. Guan, "Performance benchmarking of sdn experimental platforms," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 116–120.

[26] H. Holtkamp, G. Auer, V. Giannini, and H. Haas, "A parameterized base station power model," *IEEE Communications Letters*, vol. 17, no. 11, pp. 2033–2035, 2013.

[27] B. H. Jung, H. Leem, and D. K. Sung, "Modeling of power consumption for macro-, micro-, and rrh-based base station architectures," in *2014 IEEE 79th Vehicular Technology Conference (VTC Spring)*, 2014, pp. 1–5.

[28] J. A. Ayala-Romero, A. Garcia-Saavedra, X. Costa-Perez, and G. Iosifidis, "Bayesian online learning for energy-aware resource orchestration in virtualized rans," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[29] ——, "Bayesian online learning for energy-aware resource orchestration in virtualized rans," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[30] I. Gomez-Miguelez, A. Garcia-Saavedra, P. D. Sutton, P. Serrano, C. Cano, and D. J. Leith, "Srslte: An open-source platform for lte evolution and experimentation," in *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, ser. WiNTECH '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 25–32. [Online]. Available: https://doi-org.elib.tcd.ie/10.1145/2980159.2980163

[31] S. Sun, M. Kadoch, L. Gong, and B. Rong, "Integrating network function virtualization with sdr and sdn for 4g/5g networks," *IEEE Network*, vol. 29, no. 3, pp. 54–59, 2015.

[32] CONNECT Centre, Trinity College Dublin. OpenIreland Testbed. Accessed: Sep. 2022. [Online]. Available: http://www.openireland.eu/

[33] Y. Kim, H.-W. Lee, and S. Chong, "Mobile computation offloading for application throughput fairness and energy efficiency," *IEEE Transactions on Wireless Communications*, vol. 18, no. 1, pp. 3–19, 2019.

[34] X. Huang, R. Yu, J. Kang, Y. He, and Y. Zhang, "Exploring mobile edge computing for 5g-enabled software defined vehicular networks," *IEEE Wireless Communications*, vol. 24, no. 6, pp. 55–63, 2017.

[35] I. Jang, S. Choo, M. Kim, S. Pack, and G. Dan, "The software-defined vehicular cloud: A new level of sharing the road," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 78–88, 2017.

[36] C. Liang, F. R. Yu, and X. Zhang, "Information-centric network function virtualization over 5g mobile wireless networks," *IEEE Network*, vol. 29, no. 3, pp. 68–74, 2015.

[37] W. Zhuang, Q. Ye, F. Lyu, N. Cheng, and J. Ren, "Sdn/nfv-empowered future iov with enhanced communication, computing, and caching," *Proceedings of the IEEE*, vol. 108, no. 2, pp. 274–291, 2020.

[38] J. Liu, J. Wan, B. Zeng, Q. Wang, H. Song, and M. Qiu, "A scalable and quick-response software defined vehicular network assisted by mobile edge computing," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 94–100, 2017.

[39] R. Bruschi, F. Davoli, P. Lago, and J. F. Pajo, "A multi-clustering approach to scale distributed tenant networks for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 499–514, 2019.

[40] A. Huang, N. Nikaein, T. Stenbock, A. Ksentini, and C. Bonnet, "Low latency mec framework for sdn-based lte/lte-a networks," in *2017 IEEE International Conference on Communications (ICC)*, 2017, pp. 1–6.

[41] srsLTE, "srsLTE," accessed: Jul. 2021. [Online]. Available: https://github.com/srsran/srsran

[42] W.-K. Chiang and Y.-H. Shang, "Es-5g: A novel edge-based sdn-enabled 5g architecture for lower latency," in *Proceedings of the 2023 6th International Conference on Information Science and Systems*, ser. ICISS '23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 143–153. [Online]. Available: https://doi.org/10.1145/3625156.3625178

[43] S. D. A. Shah, M. A. Gregory, S. Li, and R. D. R. Fontes, "Sdn enhanced multi-access edge computing (mec) for e2e mobility and qos management," *IEEE Access*, vol. 8, pp. 77 459–77 469, 2020.

[44] A. Liatifis, T. Lagkas, G. P. Katsikas, A. Bujari, V. Argyriou, A. Triantafyllou, A. Lytos, A.-A. A. Boulogeorgos, and P. Sarigiannidis, "Evaluating sdn applicability in the edge," in *2023 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2023, pp. 1571–1575.

[45] J. A. Habibi, F. Djohar, and R. Hakimi, "Analyzing sdn-based vehicular network framework in 5g services: Fog and mobile edge computing," in *2018 4th International Conference on Wireless and Telematics (ICWT)*, 2018, pp. 1–7.

[46] X. Xu, Q. Huang, H. Zhu, S. Sharma, X. Zhang, L. Qi, and M. Z. A. Bhuiyan, "Secure service offloading for internet of vehicles in sdn-enabled mobile edge computing," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 6, pp. 3720–3729, 2021.

[47] X. Ge, Z. Li, and S. Li, "5g software defined vehicular networks," *IEEE Communications Magazine*, vol. 55, no. 7, pp. 87–93, 2017.

[48] S. S. Hoseini, T. Beyer, A. Ghaderi, and Z. Movahedi, "Latency-aware sdn-based mobile edge computation offloading in industrial iot," in *2023 28th International Computer Conference, Computer Society of Iran (CSICC)*, 2023, pp. 1–5.

[49] C. Huang, M. Chiang, D. Dao, W. Su, S. Xu, and H. Zhou, "V2V Data Offloading for Cellular Network Based on the Software Defined Network (SDN) Inside Mobile Edge Computing (MEC) Architecture," *IEEE Access*, vol. 6, pp. 17 741–17 755, 2018.

[50] S. Goudarzi, M. H. Anisi, H. Ahmadi, and L. Musavian, "Dynamic resource allocation model for distribution operations using sdn," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 976–988, 2021.

[51] Y. Ding, Y. He, and J.-p. Jiang, "Self-organizing multi-robot system based on personality evolution." *IEEE International Conference on Systems, Man and Cybernetics, Systems, Man and Cybernetics, 2002 IEEE International Conference on, Systems, man and cybernetics*, vol. 5, 2002.

[52] G. S. N. and S. H. M., "A game theoretic approach to swarm robotics." *Applied Bionics and Biomechanics*, vol. 3, no. 3, pp. 131 – 142, 2006.

[53] M. Dorigo, V. Trianni, E. Sahin, R. Gross, T. Labella, G. Baldassarre, S. Nolfi, J. Deneubourg, F. Mondada, D. Floreano, and L. Gambardella, "Evolving self-organizing behaviors for a swarm-bot," *AUTONOMOUS ROBOTS*, vol. 17, no. 2-3, pp. 223 – 245, 2004.

[54] G. Willcox, D. Askay, L. Rosenberg, L. Metcalf, B. Kwong, and R. Liu, "Measuring group personality with swarm ai," in *2019 First International Conference on Transdisciplinary AI (TransAI)*, 2019, pp. 10–17.

[55] Y. Wang, M. Chen, Z. Li, and Y. Hu, "Joint allocations of radio and computational resource for user energy consumption minimization under latency constraints in multi-cell mec systems," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 3, pp. 3304–3320, 2023.

[56] Y. Chen, Z. Li, B. Yang, K. Nai, and K. Li, "A stackelberg game approach to multiple resources allocation and pricing in mobile edge computing," *Future Generation Computer Systems*, vol. 108, pp. 273–287, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X19311653

[57] K. Zhang, J. Yang, and Z. Lin, "Computation offloading and resource allocation based on game theory in symmetric mec-enabled vehicular networks," *Symmetry*, vol. 15, no. 6, 2023. [Online]. Available: https://www.mdpi.com/2073-8994/15/6/1241

[58] N. Li, J. Yan, Z. Zhang, J. F. Martinez, and X. Yuan, "Game theory based joint task offloading and resource allocation algorithm for mobile edge computing," in *2020 16th International Conference on Mobility, Sensing and Networking (MSN)*, 2020, pp. 791–796.

[59] Z. Wen, K. Yang, X. Liu, S. Li, and J. Zou, "Joint offloading and computing design in wireless powered mobile-edge computing systems with full-duplex relaying," *IEEE Access*, vol. 6, pp. 72 786–72 795, 2018.

[60] W. Ejaz, M. Basharat, S. Saadat, A. M. Khattak, M. Naeem, and A. Anpalagan, "Learning paradigms for communication and computing technologies in iot systems," *Computer Communications*, vol. 153, pp. 11–25, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419312411

[61] D. SanjitKumar, D. Siddhant, M. ibitesh, and M. Sasmita, "Opportunistic mobile data offloading using machine learning approach," *Wireless Personal Communications*, vol. 110, no. 1, pp. 125–139, 2020.

[62] Y. He, C. Liang, F. R. Yu, and Z. Han, "Trust-based social networks with computing, caching and communications: A deep reinforcement learning approach," *IEEE Transactions on Network Science and Engineering*, vol. 7, no. 1, pp. 66–79, 2020.

[63] A. Mukhopadhyay, G. Iosifidis, and M. Ruffini, "Migration-Aware Network Services With Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 19, no. 2, pp. 1458–1471, 2022.

[64] M. Afrin, J. Jin, A. Rahman, Y.-C. Tian, and A. Kulkarni, "Multi-objective resource allocation for edge cloud based robotic workflow in smart factory," *Future Generation Computer Systems*, vol. 97, pp. 119–130, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18326785

[65] L. Cui, C. Xu, S. Yang, J. Z. Huang, J. Li, X. Wang, Z. Ming, and N. Lu, "Joint optimization of energy consumption and latency in mobile edge computing for internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4791–4803, 2019.

[66] X. Xu, S. Fu, Y. Yuan, Y. Luo, L. Qi, W. Lin, and W. Dou, "Multiobjective computation offloading for workflow management in cloudlet-based mobile cloud using nsga-ii," *Computational Intelligence*, vol. 35, no. 3, pp. 476–495, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1111/coin.12197

[67] W. Wu, F. Zhou, R. Q. Hu, and B. Wang, "Energy-efficient resource allocation for secure noma-enabled mobile edge computing networks," *IEEE Transactions on Communications*, vol. 68, no. 1, pp. 493–505, 2020.

[68] C. Lin, D. Deng, and C. Yao, "Resource Allocation in Vehicular Cloud Computing Systems With Heterogeneous Vehicles and Roadside Units," *IEEE Internet of Things Journal*, vol. 5, no. 5, pp. 3692–3700, 2018.

[69] J. Xu and B. Palanisamy, "Cost-aware resource management for federated clouds using resource sharing contracts," in *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, 2017, pp. 238–245.

[70] C. Li and C. Yang, "A novice group sharing method for public cloud," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018, pp. 966–969.

[71] S. N. Givigi Jr and H. M. Schwartz, "Swarm robot systems based on the evolution of personality traits." *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 15, no. 2, pp. 257 – 282, 2007.

[72] T. Yin, D. Hong-hui, J. Li-min, and L. Si-yu, "A bandwidth allocation strategy for train-to-ground communication networks." *2014 IEEE 25th Annual International Symposium on Personal, Indoor and Mobile Radio Communication (PIMRC)*, p. 1432, 2014.

[73] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey." *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, vol. 4, pp. 237 – 285, 1996.

[74] H. Yaiche, R. Mazumdar, and C. Rosenberg, "A game theoretic framework for bandwidth allocation and pricing in broadband networks." *IEEE/ACM Transactions on Networking, Networking, IEEE/ACM Transactions on, IEEE/ACM Trans. Networking*, vol. 8, no. 5, pp. 667 – 678, 2000.

[75] X.-P. Ma, H.-H. Dong, P. Li, L.-M. Jia, and X. Liu, "A multi service train-to-ground bandwidth allocation strategy based on game theory and particle swarm optimization." *IEEE INTELLIGENT TRANSPORTATION SYSTEMS MAGAZINE*, vol. 10, no. 3, pp. 68 – 79, 2018.

[76] e. Schwartz, Howard M., *Multi-agent machine learning: a reinforcement approach.* John Wiley and Sons, Inc., Hoboken, New Jersey, 2014.

[77] S. Wu, Z. Wu, X. Wu, J. Tao, and Y. Gu, "Queuing-based federation and optimization for cloud resource sharing," *Information*, vol. 13, no. 8, 2022. [Online]. Available: https://www.mdpi.com/2078-2489/13/8/361

[78] R. M. Rolly and S. Poornima, "Cell zooming for energy efficient cellular networks," in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 2014, pp. 1075–1078.

[79] M. Ruffini, A. Ahmad, S. Zeb, N. Afraz, and F. Slyne, "Virtual dba: virtualizing passive optical networks to enable multi-service operation in true multi-tenant environments," *Journal of Optical Communications and Networking*, vol. 12, no. 4, pp. B63–B73, 2020.

[80] S. Das, F. Slyne, A. Kaszubowska, and M. Ruffini, "Virtualized east-west pon architecture supporting low-latency communication for mobile functional split based on multiaccess edge computing," *Journal of Optical Communications and Networking*, vol. 12, no. 10, pp. D109–D119, 2020.

[81] M. Einhaus, I. Kim, M. B. Charaf, and P. Arnold, "Processing time aware resource allocation in software defined rans," in *2019 IEEE Conference on Standards for Communications and Networking (CSCN)*, 2019, pp. 1–6.

[82] T. Ismail and H. H. M. Mahmoud, "Optimum functional splits for optimizing energy consumption in v-ran," *IEEE Access*, vol. 8, pp. 194 333–194 341, 2020.

[83] A. Fernandez-Fernandez, C. Cervello-Pastor, and L. Ochoa-Aday, "Achieving energy efficiency: An energy-aware approach in sdn," in *2016 IEEE Global Communications Conference (GLOBECOM)*, 2016, pp. 1–7.

[84] M. Ruffini and H.-J. Reumerman, "Power-rate adaptation in high-mobility distributed ad-hoc wireless networks," in *2005 IEEE 61st Vehicular Technology Conference*, vol. 4, 2005, pp. 2299–2303.

[85] Mininet. An Instant Virtual Network on your Laptop (or other PC). Accessed: Aug. 2022. [Online]. Available: http://mininet.org/

[86] Linux htop, "htop interactive process viewer," Accessed: Oct. 2022. [Online]. Available: https://linux.die.net/man/1/htop

[87] SPECpower . (2022) ASUSTeK Computer Inc. RS720-E10-R12. Accessed: Oct. 2022. [Online]. Available: https://www.spec.org/power_ssj2008/results/res2022q1/power_ssj2008-20211214-01145.html

[88] SPECpower. (2022) Hewlett Packard Enterprise Synergy 480 Gen10 Plus Compute Module. Accessed: Oct. 2022. [Online]. Available: https://www.spec.org/power_ssj2008/results/res2022q1/power_ssj2008-20211207-01138.html

[89] C. Sonmez, A. Ozgovde, and C. Ersoy, "EdgeCloudSim: An Environment for Performance Evaluation of Edge Computing Systems," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, 2017, pp. 39–44.

[90] D. Kafetzis, S. Vassilaras, G. Vardoulias, and I. Koutsopoulos, "Software-Defined Networking Meets Software-Defined Radio in Mobile ad hoc Networks: State of the Art and Future Directions," *IEEE Access*, vol. 10, pp. 9989–10 014, 2022.

[91] M. Bouet, V. Conan, H. Khalife, K. Phemius, and J. Seddar, "MUREN: Delivering edge services in joint SDN-SDR multi-radio nodes," *CoRR*, vol. abs/1606.00994, pp. 1–7, 2016.

[92] J. Shi, J. Du, J. Wang, and J. Yuan, "Deep Reinforcement Learning-Based V2V Partial Computation Offloading in Vehicular Fog Computing," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*, 2021, pp. 1–6.

[93] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep Reinforcement Learning for Offloading and Resource Allocation in Vehicle Edge Computing and Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11 158–11 168, 2019.

[94] R. Yu, X. Huang, J. Kang, J. Ding, S. Maharjan, S. Gjessing, and Y. Zhang, "Cooperative Resource Management in Cloud-Enabled Vehicular Networks," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 12, pp. 7938–7951, 2015.

[95] M. Feng, M. Krunz, and W. Zhang, "Joint task partitioning and user association for latency minimization in mobile edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 8108–8121, 2021.

[96] N. Ahmed, A. Roy, A. Mondal, and S. Misra, "SDN-Based Link Recovery Scheme for Large-Scale Internet of Things," in *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*, 2021, pp. 1–6.

[97] P. Alvarez, F. Slyne, C. Bluemm, J. M. Marquez-Barja, L. A. DaSilva, and M. Ruffini, "Experimental demonstration of sdn-controlled variable-rate fronthaul for converged lte-over-pon," in *2018 Optical Fiber Communications Conference and Exposition (OFC)*, 2018, pp. 1–3.

[98] S. Bera, S. Misra, and A. V. Vasilakos, "Software-Defined Networking for Internet of Things: A Survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1994–2008, 2017.

[99] H. Guo, L. Rui, and Z. Gao, "V2V Task Offloading Algorithm with LSTM-based Spatiotemporal Trajectory Prediction Model in SVCNs," *IEEE Transactions on Vehicular Technology*, vol. 71, no. 10, pp. 11 017–11 032, 2022.

[100] C. Huang and J. Lin, "The k-hop V2V Sata Offloading Using the Predicted Utility-centric Path Switching (PUPS) Method Based on the SDN-controller Inside the Multi-access Edge Computing (MEC) Architecture," *Vehicular Communications*, vol. 36, p. 100496, 2022.

[101] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy Workload Orchestration for Edge Computing," *IEEE Transactions on Network and Service Management*, vol. 16, no. 2, pp. 769–782, 2019.

[102] W. Huang, L. Ding, D. Meng, J. Hwang, Y. Xu, and W. Zhang, "QoE-Based Resource Allocation for Heterogeneous Multi-Radio Communication in Software-Defined Vehicle Networks," *IEEE Access*, vol. 6, pp. 3387–3399, 2018.

[103] Beiran Chen, Yi Zhang, George Iosifidis, "Resource Sharing in Public Cloud System with Evolutionary Multi-agent Artificial Swarm Intelligence," Feb 2021, accessed: Aug. 2021. [Online]. Available: https://www.springerprofessional.de/en/resource-sharing-in-public-cloud-system-with-evolutionary-multi-/19210238

[104] S. N. Givigi Jr and H. M. Schwartz, "Swarm robot systems based on the evolution of personality traits." *Turkish Journal of Electrical Engineering & Computer Sciences*, vol. 15, no. 2, pp. 257–282, 2007. [Online]. Available: https://elib.tcd.ie/login?url=https://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=26916446

[105] S. Givigi and H. Schwartz, "Evolutionary Swarm Intelligence Applied to Robotics," in *IEEE International Conference Mechatronics and Automation, 2005*, vol. 2, 2005, pp. 1005–1010 Vol. 2.

[106] E. Kalai, "Nonsymmetric Nash Solutions and Replications of 2-person Bargaining," *International Journal of Game Theory*, vol. 6, no. 3, pp. 129–133, 1977.

[107] Python, "TimeComplexity," Accessed: Feb. 2023. [Online]. Available: https://wiki.python.org/moin/TimeComplexity

[108] H. Song and M. Chung, "Recurrent traffic demand generation using urban traffic simulation with cell-based behavior model and real traffic data," in *2023 IEEE International Conference on Big Data (BigData)*, 2023, pp. 6286–6288.

[109] Y. Lv, Y. Duan, W. Kang, Z. Li, and F.-Y. Wang, "Traffic flow prediction with big data: A deep learning approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 865–873, 2015.

[110] SPECpower . (2023) Acer Incorporated Altos R380 F2. Accessed: May. 2023. [Online]. Available: https://www.spec.org/power_ssj2008/results/res2013q2/power_ssj2008-20130409-00602.html

[111] S. Hara and Y. L. Morgan, "Managing DSRC and WAVE Standards Operations in a V2V Scenario," *International Journal of Vehicular Technology*, pp. 1–19, 2010.

[112] Rohde&Schwarz, "Intelligent Transportation Systems Using IEEE 802.11p," Accessed: Nov. 2022. [Online]. Available: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_application/application_notes/1ma152/1MA152_5e_ITS_using_802_11p.pdf

[113] S. Lu and W. Shi, *Vehicle Computing*. Springer Cham, 2024.

[114] A. Chattopadhyay, *Handbook of Computer Architecture*. Springer Singapore, 2024.