Exploring Intelligent Service Migration in a Highly Mobile Network

Onyekachukwu Augustine Ezenwigbo
M00465482

School of Science and Technology
Middlesex University London

A thesis submitted to Middlesex University in partial fulfilment of the requirements for the degree of Doctor of Philosophy

April 2022

I would like to dedicate this thesis to my father, my wife, my brothers and in memory of my lovely mother!

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Onyekachukwu Augustine Ezenwigbo
M00465482
April 2022

Acknowledgements

First and foremost, praises and thanks to God, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.

I am truly grateful to Dr. Glenford Mapp, my director of studies and supervisor, for giving me the opportunity to work under him. He has guided me and encouraged me to carry on through these years and has contributed to this thesis with a remarkable impact. I am forever indebted for his guidance, enthusiasm, and persistent support throughout this process. I benefited a lot from his advice at different stages during this research project, especially when exploring innovative ideas. I could not have imagined having a better advisor and mentor for my Ph.D. study.

I would like to express my deep and sincere gratitude to my research supervisor, Dr. Ramona Trestian for her valuable insights and suggestions through this research project.

I would like to thank the MDX VANET Research Group as follows: Dr. Arindham Ghosh for his work on seamless handover in vehicular networks, Dr. Vishnu Vardhan Paranthaman for his work on resource allocation in vehicular networks, Gayathri Karthick for her work on RASP protocol, Jose Ramirez for his work on migration experiments, and Vatsal Mehta and Amareshwari Anupama Saraswathi Bollampalli for their work on Markov models.

I would also like to thank Terri Demetriou and the Research Degrees Administration team for providing me administrative support and taking the time in addressing all my queries, and finally the administrative staff at the School of Science and Technology for providing me with all the required facilities and arrangements for doing my research and travelling to conferences.

Middlesex University has provided me with a very inspiring environment in terms of the extraordinary quality of its academic staff, and that experience will leave a mark beyond this thesis.

I extend my thanks and gratitude to my dear friend, Raphael Ejime who has always been a significant source of support and encouragement when things would get a bit disappointing. A special word of thanks goes to all my other friends and colleagues who have provided me with continuous support and encouragement throughout my Ph.D. journey.Thank you, guys, for always being there for me.

I would like to dedicate this thesis to my father Onyeka Innocent Ezenwigbo and my late mother, Ngozi Martina Ezenwigbo. This accomplishment would not have been possible without them. Thank you for always supporting me and believing in me. Thank you for teaching me respect, faith, and good manners. I will never truly be able to express my sincere appreciation to both of you.

I would like to express my heartfelt thanks and gratitude to my brothers Uchechukwu Emmanuel Ezenwigbo and Oluchukwu JohnPaul Ezenwigbo for being part of my life's journey. To my Parent-in-law, Meinrad Waldmann and Helga Scharinger, thank you for your love, support and encouragement.

Finally, to Marilena Scharinger Ezenwigbo, love of my life, there are no words to express how much you have helped me throughout these years. Without you, this would have been impossible. Thanks for taking care of us, making us smile no matter how bad the days were. I am blessed to have you.

Abstract

Mobile services allow services to be migrated or replicated closer to users as they move around. This is now regarded as a viable mechanism to provide good Quality of Service to users in highly mobile environments such as vehicular networks. The vehicular environment is rapidly becoming a significant part of the internet and this presents various challenges that must be addressed; this is due to continuous handovers as mobile devices change their point of attachment to these networks resulting in a loss of service. Therefore, this explains the need to build a framework for intelligent service migration. This thesis addresses these issues.

It starts by discussing the requirements for intelligent service migration. Then it investigates a low latency Quality of Service Aware Framework as well as an experimental transport protocol that would be favoured by vehicular networks.

Furthermore, two analytical models are developed using the Zero-Server Markov Chain technique which is a way of analysing scenarios when the server is not continuously available to serve. Using the Zero-Server Markov Chain, the first analytical model looks at lost service due to continuous handovers and the communication dynamics of vehicular networks, while the second model analyses how service migration affects service delivery in these networks. Formulas are developed to yield the average number of packets in the system, the response time, the probability of blocking and a new parameter called the probability of lost service. These formulas are then applied to the Middlesex VANET Testbed to look at reactive and proactive service migration. These techniques are then incorporated into a new Service Management Framework to provide sustainable Quality of Service and Quality of Experience to mobile users in vehicular networks. This thesis also shows that this new approach is better than current approaches as it addresses key issues in intelligent service migration in such environments, and hence can play a significant part in the development of Intelligent Transport Systems for Smart Cities.

Table of contents

List of figures

List of tables

Abbreviations

$K,0$	State K of Markov Chain 0
$K,1$	State K of Markov Chain 1
L_{EH}	average number of people in the system
N	Resource Hold Time
NDD	Network Dwell Distance
$NP1$	not possible to migrate to the first network
$NP2$	not possible to migrate to the second network
$NP3$	service migration could not be performed because $X_{min} \geq X_{max}$
P_B	blocking probability
P_{LS}	Probability of lost service
$P_{n,0}$	State probability in Chain 0
$P_{n,1}$	State probability in Chain 1
R_E	Exit Radius
R_H	Handover Radius
SHT	Service Hold Time
SMT	Service Migration Time
T_v	vacation time
T_{EH}	Handover Time

T_{resp}	Response Time
X	starting handover
$\aleph$	Network Dwell Time
$\hbar$	handover execution time
λ	Arrival rate of requests
$\mathbb{N}$	Velocity
μ	Service Rate
μ_v	exit velocity
μ_v	rate
ρ	utilisation
h	handover time
t_{adp}	Adaptation Time
t_{con}	Configuration Time
t_{det}	Detection Time
t_{reg}	Registration Time
v	Velocity
v_1	Transition rate from Chain 1 to Chain 0
v_2	Transition rate from Chain 0 to Chain 1
K	capacity

Acronyms / Abbreviations

5G	Fifth-generation
AL	Application Layer
AP	Access Point
AR	Access Router

BPSO	Binary Particle Swarm Optimizers
br0	virtual network
BS	Base Station
BSM	Basic Safety Message
C-ITS	Cooperative-ITS
CAM	Cooperative Awareness Message
CAP	computing access point
CAV	Connected and Autonomous Vehicles
CLI	command-line interface
CPU	Central Processing Unit
CRIU	Checkpoint/Restore in user-space
CRRM	Cooperative Radio Resource Manager
CXTP	Context Transfer Protocol
DfT	Department for Transport
DPDK	Data Processing Data Kit
EECO	energy-efficient computation offloading
fcl	forward connection label
FEC	Forward Error Correction
Fi-Wi	fibre-wireless
FUSE	Filesystem in Userspace
H-CRAN	Heterogeneous Cloud Radio Access Network
ICA	Intersection Collision Alerts
ID	Identification
IIS	Intelligent Interface Selection

IP	Internet Protocol
IPsec	Internet Protocol Security
ITS	Intelligent Transport System
KVM	Kernel-based Virtual Machine
LoS	Line-of-Sight
LTE	Long Term Evolution
LXC	Linux Containers
LXD	Linux container hypervisor
MAC	Media Access Control
MANET	Mobile Ad hoc Network
MDX	Middlesex University
MEC	Multi-access Edge Computing
MIMO	multi-cell multipleinput and multiple-output
MMC	Mobile Cloud Computing
MMC	Mobile Memory Cache
MN	Mobile Node
MPTCP	Multi-Path Transmission Control Protocol
MT	Mobile Terminal
NDT	Network Dwell Time
NFV	Network Functional Virtualization
NIIS	Network Intelligent Interface Selection
NLA	Network Level Agreements
NMCP	Network Management Control Protocol
NMS	Network Memory Server

OBU	Onboard Unit
OS	Operating System
PoA	Point of Attachment
PSS	Persistent Storage Server
PVD	Probe Vehicle Data
QoS	Quality of Service
RAM	Random Access Memory
RAN	Radio Access Network
RASP	Resource Allocation Security Protocol
RHT	Resource Hold Time
RSA	Road Side Alert
RSU	Road Side Unit
RTT	Round Trip Time
Rx	Receiver
SBMC	Service-Based Markov Chain
SCL	Service Connection Layer
SDN	Software Defined Networking
SLA	Service Level Agreements
SLTP	Simple Light-weight Transport Protocol
SM	Service Migration
SManL	Service Management Layer
SMF	Service Management Framework
SML	Service Migration Layer
TBVH	Time Before Vertical Handover

TCP	Transmission Control Protocol
TfL	Transport for London
Tx	Transceiver
V2P	Vehicle-to-Pedestrian
VANET	Vehicular Ad hoc Network
VEC	Vehicular Edge Computing
VLR	Visitor Location Register
VM	Virtual machine
VPN	Virtual Private Network
WLAN	Wireless Local Area Network
WSMP	Wave Short Message Protocol
Xmax	Maximum value of x
Xmin	minimum value of X
ZSMC	Zero-Server Markov Chain

Chapter 1

Introduction

1.1 Motivation

In recent times, vehicles can become aware of the dynamic behaviour of other vehicles without relying on the driver's faculties. This technology is achieved when all vehicles are interconnected in a Vehicular Ad hoc Network (VANET). Furthermore, people have wondered about how to better understand and implement service migration in such a network due to an exponential increase in vehicular population. Few attempts have been made by researchers to identify this problem, but none of them got very far because it is a complex problem and hence it is difficult to solve in a single attempt. As a result, I examine an alternative path in this thesis by breaking down the topic into key subsystems including low latency and QoS frameworks, analytical models for handover as well as service migration and finally the development of a Service Management Framework.

1.2 Background

There is an increasing amount of vehicle and traffic congestion globally, which can be addressed by the development of an Intelligent Transport System (ITS) [87]. VANET technology is considered as one of the principal elements of ITS. It consists of a high-speed network and high mobility vehicles equipped with sensors and a communication capability device. It makes each of the neighbouring vehicles into a wide range wireless network, by enabling vehicular connectivity and content sharing. Unlike other mobile networks, VANETs can be categorized in terms of highly variable network topologies, specific speed patterns, communication conditions etc. Emerging technologies such as 802.11p and Fifth-generation mobile communication standards (5G) will give rise to the ubiquitous deployment of vehicular networks. VANET is an example of such a system. These networks work by using Road Side Units (RSUs), Access Points (APs) or Base Stations (BSs) on the roadside infrastructure and Onboard Units (OBUs) in the vehicles or on cyclists and pedestrians. Meanwhile, VANET has a limited capacity to disseminate critical road safety information over such a large area, and thus some major integration issues occur when implemented in real life. These major challenges are (1) network disconnection problems (2) network coverage (3) high bandwidth (4) mobility management (5) broadcast storm problems (6) data dissemination technique, and (7) Service Migration. For this research we mainly focus on Intelligent Service Migration in the VANET system because the need to provide a sustainable Quality of Service (QoS) is vital in this environment.

The existing centralized structure of the cloud-based architecture [48] has made a generally large geographical separation between the mobile users and the cloud infrastructure. In this scenario, end-to-end communication between the mobile user and cloud infrastructure can involve a lot of network hops, thereby introducing high network latency. Additionally, the network bandwidth of the cloud may also depreciate because the cloud infrastructure is accessed on a many-to-one basis. A new type of approach to resolve the above problems would be to install computing infrastructures towards the edge of the network, which can be used to enable mobile services. These services can therefore be migrated to different Edge Clouds as users move around, hence maintaining a high QoS for mobile users. In this context, Multi-access Edge Computing (MEC) [66, 85] is an emerging and significant technology that is being used to address some of the challenges in Mobile Cloud Computing (MCC). It provides services at the edge of the core internet that can reduce latency, improve efficiency, and ensure better service delivery.

Furthermore, MEC is an existing and capable approach to access large data locally and evade extensive latency, especially in vehicular networks as the Mobile Node (MN) moves at a high speed and hence, requires low latency. Once vehicular users request services through a core network from edge networks far from the cloud, it may cause extensive latency. MEC was established to overcome these disadvantages of traditional cloud computing [78, 65]. A lot of research has motivated Vehicular Edge Computing (VEC). Most of the recent research focused on the VEC architecture design but failed to investigate mobility and how it can affect service migration in a highly mobile environment. Moreover, the migration of the cloud services close to the MNs helps to address the problem of high latency by moving the service closer to the MNs. However, in the conventional MCC [86], the cloud services are called up via the internet connection whereas, in the case of the MEC, the computing resources are located in the proximity of the MNs. On the other hand, the MEC provides only limited computational and storage resources with respect to centralised cloud computing.

Therefore, the MEC can offer significantly lower latency and jitter when compared to the traditional MCC, and also provides an environment for service migration to take place.

Service migration [93] in a mobile environment refers to the migration of services from the original location to the destination for a better load balancing in the system. In a dynamic large-scale mobile environment such as a vehicular system, service migration is a more satisfactory service deployment method. By implementing an appropriate service migration scheme, the system can maintain better QoS. In this research, two categories of service migration, reactive and proactive service migration will be investigated. Reactive migration migrate services at a fixed time while proactive migration is variably based on a given situation. This gives rise to the consideration of intelligent service migration, thereby enabling efficient service delivery with the help of better analytical models and an architecture that dynamically supports it. The intelligent migration of services to edge systems such as RSUs, APs and BSs, will result in the ability to maintain a high QoS to users as they move around.

Authors in [75] proposed a new system for efficient live migration for applications using an intelligent system capable of load balancing and self-automation by executing a secure live migration. The system went through performance analysis to prove its efficiency and was compared with the previous standard systems.

Wang et al. [90] used a two-layer cellular network that had mobility-aware mobile equipment's that would improve the efficiency of task processing by using a MEC environment. A forward joint task offloading, and computation allocation schemes based on mobility-aware migration was developed to lower the migration probability and maximize the total revenue. This resulted in a mixed-integer non-linear programming problem; therefore, a Reinforcement Learning Based algorithm was derived to solve this problem. They obtained the maximum total revenue of mobile equipment's availably.

It is well known that issues with the likes of data leakage happen with virtual machine migration on the cloud computing platforms. The paper in [83] showed a simulating result that secured virtual machine migration method to assure the integrity and improve security performance at the point of virtual machine migration. It also demonstrates how it decreased total migration time and downtime and increases the online migration efficiency of their virtual machines.

Authors in [84] proposed a statistical method for estimating total migration time. This enabled them to determine the optimal size of the server monitoring window.

The change of VM's from one environment to another makes them prone to attacks. Also, there is a higher risk of a network channel been prone to a high-level security breach when initiating a live service migration. Authors in [61] showed how these channels can be protected using Internet Protocol Security (IPsec) tunnelling. It was enhanced using IPsec and onion routing algorithm.

Putra et al. [63] conducted a live migration based on cloud computing to increase load balancing. Based on their analysis it was concluded that in a normal migration the larger data on disk is migrated, the more time it takes for migration, live migration requires less time than normal migration and live migration can easily balance server loads without obstructing running guest in VM.

Authors in [3] proposed an energy-efficient VM migration was based on a time-constrained file transfer technique. The research showed how it increases the sleeping time of mobile nodes by delaying crossing requests.

The survey analysis of live migration overhead in a cloud computing environment was done by authors in [19]. They discussed the energy cost that comes with live migration. Furthermore, it showed how different research methods for estimating this cost were based on different parameters. The authors concluded that the network bandwidth is the main factor causing the energy cost of migration to be high.

There are lots of research efforts that have looked into migrating services from the core cloud to the edge of the network, but the focus of this research is to explore intelligent service migration at the edge of the network between Edge Clouds in order to provide better QoS in highly mobile environments such as vehicular networks. In this context, it investigates not just the service migration in a system but also the communication dynamics. A real experimental vehicular network testbed is introduced. Results collected from the real experimental testbed are used to better understand the impact of service migration within a VANET environment.

Therefore, to attain an intelligent service migration in a system, we will need to develop a low latency environment, have the ability to measure the QoS at various locations in the network, develop analytical modelling suitable for MEC environments, investigate service migration mechanisms, and define a service orchestration framework. This research will help us to understand the parameters needed to migrate services from the core to edge and from edge to edge in a highly mobile environment. In addition, service migration protocols can be developed based on relevant parameters.

1.3 Key Research Questions

Targeting the challenges of providing an intelligent service migration discussed above, this research attempts to answer the following question: Can we develop a system to help us decide the best place to run a service at any place in time, given the mobility of a user in a highly mobile environment? This can be broken down into the following questions:

- When is the best time to migrate the service?
- Where should the service be migrated?
- How should the service be migrated?
- How do we develop an optimised model for service migration?
- How do we apply this to a VANET system?

In the above questions, we identify the two key factors that need to be addressed. Firstly, we define user mobility as the movement of the mobile user, and how their location, velocity and acceleration change over time while accessing a service. Secondly, the distance between the user and the mobile node is considered as the key parameter by which we can improve QoS and reduce network traffic. This is due to mobile users moving randomly and hence makes it impossible to guarantee QoS in any wireless networks used by mobile users. Therefore, the best approach is to identify and reduce this to its bare minimum by considering the effects of the key elements, such as latency and bandwidth, on the QoS.

1.4 Original Contributions

The key contributions of this thesis are as follows:

- The proposal of a new Application Framework which would facilitate intelligent service migration.
- It explains the Zero-Server Markov Chain concept and shows how this can be used to develop an analytical model for vehicular networks.
- Generalised equations for the Markov steady state probabilities are obtained using two key parameters: The Resource Hold Time and the handover time. The results were obtained in terms of the average number of requests in the system, response time, blocking probability and probability of lost service.
- It looks at the Service Hold Time and the Service Migration Time and then devises equations for intelligent service migration based on these parameters. It first considers reactive service migration and then goes on to consider proactive service migration.

- This developed model and the equations are applied to the Middlesex VANET Testbed which is a real connected vehicle testbed located in West London.
- These results are integrated into a newly developed Service Management Framework to provide sustainable QoS for mobile users in vehicular networks.

1.5 Thesis outline

The following is an outline of the final thesis:

- Chapter 2: The literature review presents a critical analysis of the existing solutions and approaches produced by researchers and scientists.
- Chapter 3: Details the methods and approaches used for directing this research.
- Chapter 4: Investigates a low latency QoS aware environment by analysing frameworks and detailing the use of an experimental low latency transport protocol. The transport protocol underwent performance testing and yielded interesting results.
- Chapter 5: This details the Handover process in a vehicular network by developing an analytical model, using Markov Chain analysis. This chapter generated results based on its analysis from the VANET testbed.
- Chapter 6: The development of analytical model for intelligent service migration coupled with extensive results are displayed.
- Chapter 7: It investigates the design of an Intelligent Service Management Framework for vehicular network and its implementation in the C programming language.
- Chapter 8: This concludes the thesis with a summary of the thesis and directions for future work, in order to ensure continual improvement in the current and related field of study.

1.6 List of Papers

1. "Exploring the Provision of Reliable Network Storage in Highly Mobile Environments", *13th International Conference on Communications, COMM 2020, Bucharest, Romania, June 18-20, 2020, Proceedings*

2. "A New Service Management Framework for Vehicular Networks", *23rd Conference on Innovation in Clouds, Internet and Networks and Workshops, ICIN 2020, Paris, France, February 24-27, 2020, Proceedings*

3. "Exploring Intelligent Service Migration in Vehicular Networks", *13th EAI International Conference, TridentCom 2018, Shanghai, China, December 1-3, 2018, Proceedings*

4. "Exploring a New Transport Protocol for Vehicular Networks", *5th International Conference on Internet of Things: Systems, Management and Security, IoTSMS 2018, Valencia, Spain, October 15-18, 2018, Proceedings*

5. "Exploring a New Framework to Build Mobile QoS-Aware Applications and Services for Future Internet", *Journal of Communications Vol. 13, No. 10, October 2018*

Chapter 2

Technical Background and Related Work

2.1 Brief Introduction

This chapter is broken down into two parts. The first part is focused on the technical background which introduces us to a number of technologies that are designed to achieve seamless communication. Secondly, it details the related work of researchers who investigated MEC, computational offloading decisions, and Service Migration. Then it explores the requirements and solution approach for intelligent service migration.

2.2 Technical Background

2.2.1 Y-Comm Reference Framework

Y-Comm [57] is a framework designed for the future, while keeping in mind the diversity of mobile wireless infrastructure which works in an ever-increasing heterogeneous environment. The architecture of the internet will receive a significant change when heterogeneous networks are deployed, such as faster internet speeds using optical cables that reach gigabyte per second data speed and the deployment of new wireless technologies. The framework consists of two parts, the Core Framework and the Peripheral Framework both of which join at the two bottom layers forming a Y-shape hence the name. Y-Comm's investigation of proactive handover and mobility prediction are essential parts of the investigation presented in this research. The Y-Comm framework is presented in Figure 2.1 and its constituent layers are explained below.

- The Core Framework displays the functionality essential in the core network to support the Peripheral Framework.

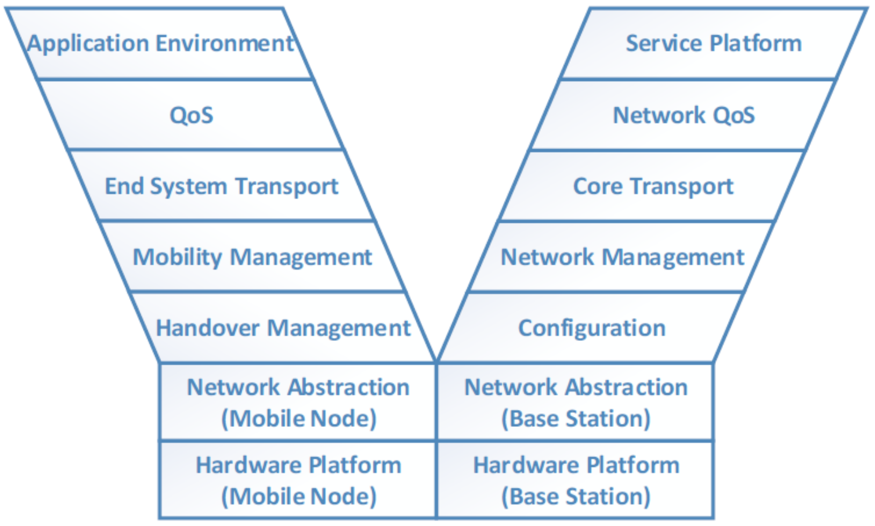


Fig. 2.1 Y-Comm Reference Framework

- The Peripheral Framework handles functions and operations on the MN and on peripheral wireless networks.

Peripheral Framework

The first two layers for both frameworks are comparable in functionality. The Hardware Platform Layer is used to classify all relevant wireless technologies such as modulation techniques and Media Access Control (MAC) while the Network Abstraction Layer offers a common interface to control and manage different wireless technologies. The Handover Management Layer acquires the resources from the Configuration Layer then executes vertical handover. The Mobility Management Layer chooses when and why handover would occur. The End Transport Layer provides the functionalities of Transport layers and Network Layers of the Transmission Control Protocol/Internet Protocol (TCP/IP) module in the IP Suite of protocols. The QoS Layer supports two mechanisms for handling QoS, the Downward QoS, where an application specifies its required QoS to the system, and Upward QoS, where the application tries to adapt to the changing QoS. Finally, the Application Environments Layer specifies a set of functions, routines and objects to build applications that make use of the framework.

The Core Framework

The Configuration Layer is in control for dealing with key infrastructure such as switches, routers and mobile network infrastructure, while the Network Management Layer controls the networking operations in the core and gathers information on peripheral networks such that it can inform the Mobility Management Layer on mobile nodes about wireless networks at their various locations. The Core Transport System helps to move data through the core network whereas the Network QoS Layer is concerned with QoS issues within the core network especially the interface between the peripheral networks and core network. Finally, Service Platform Layer permits services to be installed on various networks simultaneously.

2.2.2 Multi-access Edge Computing

Multi-access Edge Computing [81] is a network architectural concept that enables cloud computing capabilities and an IT service environment at the edge of the core network. This allows us to place peripheral devices such as routers, switches, computers etc., at the edge of the network. This environment is designed to reduce the latency and helps to provide mechanisms to supply services efficiently. This addresses the research question of where a service can be migrated because it introduces the environment for service migration to take place.

2.2.3 Vehicular Ad hoc Networks

VANETs [87] are created by applying the principles of Mobile Ad hoc Networks (MANETs) [30] to the domain of vehicles. VANETs were first mentioned and introduced in 2001 under "car-to-car ad hoc mobile communication and networking" applications, where networks can be formed and information can be relayed among cars. It was shown that vehicle-to-vehicle and vehicle-to-roadside communications architectures will co-exist in VANETs to provide road safety, navigation, and other roadside services. VANETs are a key part of the ITS framework. Sometimes, VANETs are referred to as Intelligent Transportation Networks. Figure 2.2 shows a graphical representation of a VANET network scenario with mobile cars.

The main reason why we are looking into VANET is that it is an example of a highly mobile environment and hence it will be investigated for this research. This helps us to address the research question of where to migrate services. Some examples of VANET applications are Electronic Brake Lights, Platooning, Traffic Information Systems, Road Transportation Emergency Services and On-The-Road Services. VANETs can use any wireless networking technology; the most prominent are short range radio technologies like Wireless Local Area Network (WLAN). In addition, cellular technologies such as Long Term Evolution (LTE) and 5G can be used for VANETs.

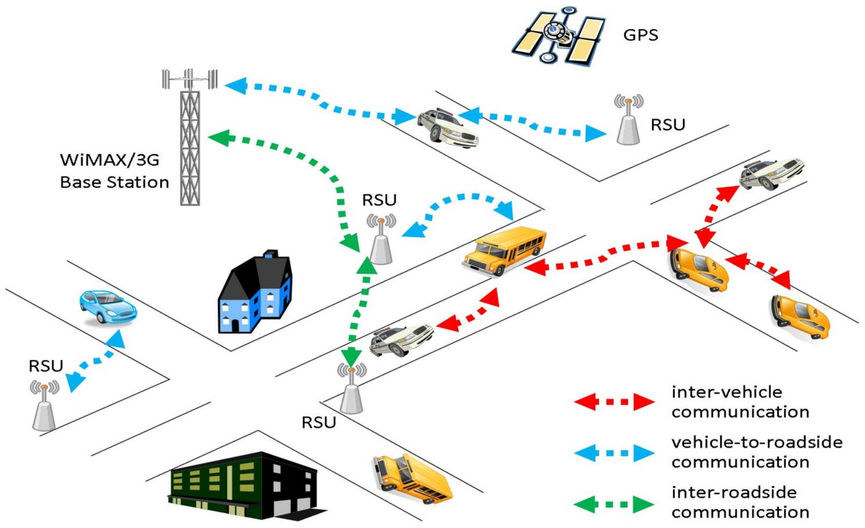

GPS
RSU
WiMAX/3G Base Station
RSU
RSU
RSU
inter-vehicle communication
vehicle-to-roadside communication
inter-roadside communication

Fig. 2.2 VANET network scenario

2.2.4 Container Technology

In this section, we talk about the service migration technology that exists, starting from the older to the latest systems. The container technology as discussed below helps to address the research question on how we will be able to migrate service across the network and they are:

Virtual Machine

This is an emulation of a computer system. Virtual machines [24] are based on computer architectures and provide the functionality of a physical computer. They involve specialized hardware and software.

There are two types of virtual machines:

- System virtual machines: These provide substitutes for a real machine. They provide the functionality needed to execute entire operating systems. A hypervisor uses native execution to share and manage hardware, allowing for multiple environments which are isolated from one another, but still exist on the same physical machine.
- Process virtual machines: They are designed to execute computer programs in a platform-independent environment. It is designed to run applications in the same way regardless of the platform.

Docker

This is a computer program that performs operating-system-level virtualization, also known as "containerization" [60]. It was first released in 2013 and developed by Docker, Inc [18]. Docker is used to run software packages called "containers". In a typical example, one container runs a web server and web application, while a second container runs a database server that is used by the web application. Docker containers are isolated from each other and bundle their own tools, libraries and configuration files; they can communicate with each other through well-defined channels. All containers are run by a single operating system kernel and are thus more lightweight than virtual machines. Containers are created from "images" that specify their precise contents. Images are often created by combining and modifying standard images downloaded from repositories.

Kernel-based Virtual Machine (KVM)

In cloud computing virtualization techniques, which are typical representatives of the cloud that have developed rapidly, mobile users can use the virtualized product with no detailed specification know how. Presently, operating systems can coexist on the same machine with the help of virtual machines. Software manufactories like Redhat, Cisco, VMware, Microsoft, are developing their own hypervisors such as KVM [97]. KVM is a full virtualization solution that is easily customized by cloud providers. It has been incorporated within Linux Operating system such as CentOS, Fedora, and Ubuntu as a component. This makes KVM often use Linux Operating System (OS)-based resources for virtualization tasks in the cloud. Such characteristics make KVM a popular hypervisor.

Linux Container Hypervisor (LXD)

LXD [64] is an open source container management extension for Linux Containers (LXC). LXD both improves upon existing LXC features and also provides new features and functionality to build and manage Linux containers. It employs a REST API that communicates with LXC through the liblxc library. LXD also supplies a system daemon that applications can use to access LXC and has a template distribution system to allow faster container creation and operation. Some important features of LXD are a powerful command-line interface (CLI), high scalability, improved security, improved control over computing resources, network and storage management, and live migration of running containers between hosts.

Unikernels

They [41] are developed by executing high-level languages directly into specialised machine images which run directly on a bare metal hypervisor. The minimal set of libraries required to run an application on specific hardware is selected from a modular stack. It eliminates the overhead introduced by traditional operating systems. This provides many benefits compared to a traditional Operating System, including improved security, smaller footprints, more optimisation and faster boot times.

2.3 Related Work

2.3.1 Literature Review on Multi-access Edge computing

With regard to delay-constrained offloading for Multi-access Edge Computing (MEC) in cloud-enabled vehicular networks, the authors in [95] proposed a vehicular offloading framework in a cloud-based MEC environment. They were able to investigate the computation offloading mechanism. The latency and the resource limitations of MEC servers were taken into consideration which enabled the proposal of a computation, resource allocation and a contract-based offloading scheme. The scheme was intended to exploit the utility of the MEC service provider to satisfy the offloading requirements of the task. Given the significance of increased research in combining networking with MEC to support the development of 5G, the authors in [67] investigated the conceivable outcomes of engaging coordinated fibre-wireless (Fi-Wi) to get networks to offer MEC abilities. More predominantly, imagined plan situations of MEC over Fi-Wi networks for typical Radio Access Network (RAN) advancements were explored, representing both network architecture and enhanced resource management.

Moreover, authors of [94] showed the architectural description of the MEC platform along with the key functionalities. They agreed that the RAN is enhanced by the computation and storage capacity provided using MEC. The primary benefit of MEC is to allow significant latency reduction to applications and services as well as reduced bandwidth consumption. The enhancement of RAN technology with the MEC's capability facilitates the use of its edge server cloud resources to provide context-aware services to nearby mobile users in addition to conducting the user traffic forwarding. Kikuchi et al. [37] proposed a MEC-based VM migration scheme whereby a VM migration is conducted to reduce congestion at the edge of the network. They solved two QoS problems which were the congestion in a wireless access network and congestion in computing resources on an edge based on the network TCP throughput by considering different network scenarios that increase it.

A recent survey on architecture and computation offloading in MEC [47], explained that the current research being carried out regarding the MEC is basically around how to guarantee service continuity in highly dynamic scenarios. They clearly state that this part is lacking in terms of previous research and is one of the key hindrances to the use of the MEC concept in real world scenarios. Furthermore, they argued that recent validated research will not be accepted due to their simplistic scenarios, simulations or analytical evaluations but instead real tests and trials are further required in realistic scenarios. Recently, the ability to use low cost devices that have agreed virtualised services appear to be a better alternative to support computational requirements at the edge of a network. Lertsinsrubtavee et al. [44] introduced PiCasso, which is a lightweight service orchestration at the edge of a network. They further analysed and discussed their benchmarking results which enable them to identify important parameters that PiCasso would need to play a key role in future network architectures.

2.3.2 Computation Offloading Decision

MEC emerges as a promising model to improve the quality of computational experience for mobile devices. Liu et al [45] adopted a Markov decision approach to handling the execution of a task scheduling policy used for MEC systems by integrating different periods in the task execution process and the channel fading process. Based on their analysis of the average delay and the average power consumption at the mobile device, they developed an efficient one-dimensional search algorithm to find the optimal task scheduling policy.

Furthermore, Mao et al [49] investigated MEC systems with Energy Harvesting mobile devices. They developed a dynamic computation offloading policy, called the Lyapunov optimization-based dynamic computation offloading (LODCO) algorithm. It is a low-complex online algorithm and requires little prior knowhow. Their study provides a sustainable approach to design future MEC systems with renewable energy-powered devices.

Researchers in [34] considered an upgrade of 4G architecture, by placing cloud computing resources at the edge of the network. They investigated Joint radio resource allocation and offloading strategies to leverage this new feature, by implementing policies designed to minimize the average power consumed by the user equipment while satisfying predefined delay constraints.

Labidi et al [42] proposed to jointly optimize the radio resource scheduling and offloading to minimize the average energy consumed by the mobile terminal by processing its application under average delay constraints. Both deterministic and randomized offline policies based on dynamic programming were studied.

Authors in [43] addressed computation offloading difficulty from mobile users to their serving small cell base stations by jointly optimizing the radio resource scheduling and computation offloading in order to minimize the average energy consumed by all the users' terminals to process their mobile applications under average delay constraints allowed by these applications. They investigated the problem of offline and online dynamic programming approaches and devised a deterministic solution to find the optimal scheduling offloading.

The exploitation of cloud computing capabilities can be enhanced by allocating radio and computational resources jointly. Barbarossa et al [6] proposed a method to jointly optimize the transmission power, the number of bits per symbol and the Central Processing Unit (CPU) cycles assigned to each application in order to minimize the power consumption on the mobile side. They considered the case of a set of mobile handsets served by a single cloud and showed that the optimization leads to a one-to-one relationship between the transmit power and the percentage of CPU cycles assigned to each user. Finally, they proposed a computation scheduling technique, verified the stability of the computational queues, and showed how these queues are affected by the degrees of freedom of the channels between mobile handsets and server.

Authors in [70] explored the computation offloading problem in a multi-cell mobile edge-computing scenario, where they used a dense deployment of radio access points to facilitate proximity of high bandwidth access to computational resources and also increased the intercell interference. They formulated the resource optimization problem as the joint optimization of radio and computational resources, which was aimed at minimizing mobile users' energy consumption, under latency and power budget constraints.

Sardellitti at al [69] considered the use of a single cloud and proposed an optimization approach to merge the computational and communication resources in a multi-cell multiple-input and multiple-output (MIMO) Femto-cloud network in order to reduce the overall transmitting energy of all the mobile users asking for computation offloading to a common cloud server, by using advanced optimization strategies.

Authors in [96] investigated the MEC offloading mechanisms in 5G heterogeneous networks. In order to improve the energy efficiency of the offloading system, they minimized the energy consumption of the computation task implementation as well as its communication process. The authors proposed an energy-efficient computation offloading (EECO) scheme, which jointly optimizes the computation offloading decisions and the radio resource allocation strategies to minimize the system energy cost under the delay constraints. Furthermore, they conducted a simulation study that clearly shows the energy efficiency enhancement.

Authors in [13] proposed a game theory approach for the computation offloading decision making among multiple mobile device users for mobile-edge cloud computing scenario. They were able to formulate the problem as a multi-user computation offloading game and showed that the game always reaches a Nash equilibrium. They also designed a distributed computation offloading algorithm that can achieve a Nash equilibrium.

Chen et al [11] recommended an efficient offloading decision algorithm by semi-definite relaxation and a novel randomization mapping method. Their simulation results showed that the proposed algorithm gave a close optimal performance with only a small number of randomized iterations, and adding computing access points to the traditional separation of mobile devices and remote cloud servers can drastically improve mobile cloud computing performance.

Authors in [10] considered a mobile cloud computing system that consists of multiple users, a remote cloud server, and a single computing access point (CAP). The CAP takes into account the receiving of tasks from the mobile user and offloading them to the cloud. In order to optimise offloading decisions, they proposed an efficient solution by using a semi-definite relaxation and a novel randomization mapping method.

Furthermore, authors in [9] considered the offloading of mobile computing based on a heterogeneous network. They proposed an energy-optimal offloading algorithm of mobile computing to achieve the maximum saving energy of the mobile terminal under the requirement of the given application execution time.

As an increasing number of mobile users now run complex applications, more energy and computing power are required. Authors in [17] presented a fine-granularity offloading policy, focused on reducing the energy depletion while satisfying a strict delay constraint. They achieved this by using a practical application consisting of a set of tasks and modelled it as a generic graph topology. Then, they formulated the energy-efficient task offloading problem as a mathematically constrained programming. Furthermore, to solve low computing resource issues, the use of Binary Particle Swarm Optimizers (BPSO) [32] algorithms was adopted.

In [98] authors targeted the reduction of terminal energy consumption, and studied the joint optimization of radio and computational resources for multiple users in mobile cloud computing and recommended a heuristic strategy, based on the latency constraint and the application type of each mobile terminal, for resource allocation of low computational complexity.

2.3.3 Service Migration

Xu et al. [91] presented a highly efficient live migration system called the Sledge, which ensures component integrity by combining both management context and images during the runtime migration of Docker containers. The authors ran some experiments and proved that, when compared with the state-of-the-art, their efficient live migration the system reduces the overall system time by 70% downtime, 57% of total migration time, and 55% of image migration time.

Authors in [27] presented a locality live migration model by considering available bandwidth, the distance, and the cost between containers. They conducted experiments on a cluster and showed extensive simulations that revealed its validity in improving the utilisation of resources.

Authors in this paper [14] developed an intelligent service migration algorithm model surrounding machine learning algorithms. The algorithm takes into consideration the dynamics of battery level and bandwidth of the mobile device. They performed a simulation which showed that the performance of the proposed service migration algorithm was better when compared with the greedy algorithm and the dynamic programming algorithm.

Authors in [92] studied the network congestion that is caused by MN movement in a mobile edge environment. A VM migration system is anticipated by migrating the VM to the edge with less congestion. If latency caused by accessing more edge networks is smaller than when migration has not occurred, the TCP throughput is increased to improve the QoS through migration.

Due to an increasing demand of data, authors in [99] proposed a solution to this problem by distributing data traffic to multiple device nodes for distributed processing by using the approach of data migration. They took into consideration of the bandwidth constraints, load balance, migration costs, and other factors. They designed a two-level model of multi-intelligent data migration strategy based on SDN to solve the problem of data migration.

In [79], the authors proposed a Compressed Sensing Routing control-method with Intelligent Migration-mechanism based on Sensing Cloud-computing. It starts by determining the speed and position of the target node through compressed sensing theory. Reduction of network load is achieved by a routing tree. A centre of fog nodes is established to obtain data in route effectively and optimize the data aggregation routing process. Therefore, making the energy cost of the whole network balanced. Finally, their simulation experiments show that method and other algorithms have improved the average data aggregation rate by 8.19%, and the average network coverage has increased by 12.65%. this signified that the proposed algorithm works.

Tan et al. [82] presented a method on the live migration of VM in edge clouds which was based on hybrid memory. They optimized the pre-copy algorithm for a hybrid memory structure by setting up a transmission queue, then migrated the cold data to the Persistent Memory area of the destination iteratively and finally transfer the hot data to a Dynamic Random Access Memory area. By doing so it greatly reduced the amount of data transmitted in the migration process and optimized the data placement in the hybrid memory of the destination server. They tested several workloads that simulated real edge computing scenarios, their result showed an improvement in the migration performance.

To reduce the impact of user equipment mobility on computational results feedback of the MEC server, authors in [22] used a resource allocation strategy based on VM migration to maximize the system energy efficiency. User equipment can harvest radio frequency energy from multiple frequency bands as they move. Considering the mobility of the user equipment they used VM migration method to transfer computational tasks. The sub-optimal solution was achieved by introducing the genetic algorithm method.

2.4 Research Gap

The dynamics of the wireless environment makes the provisioning of guaranteed QoS a significant challenge especially within highly mobile environments like VANET. The major difference with my research as compared to other works is that they have not been able to take into account the network resources and mobility of a mobile user for performing a service migration in a VANET network scenario and that is the key area this research aims to address.

2.5 Intelligent Service Migration

Intelligent service migration can be defined as a system that takes into account low latency, QoS feedback mechanisms, analytical models, service migration, and orchestration frameworks to decide the best where, when, and how should a service be migrated such that an optimum QoS can be maintained in a MN throughout the system. It takes into account the communication dynamics. The next section describes the requires for intelligent service migration in details.

2.6 Requirements for Intelligent Service Migration

2.6.1 Low Latency

The author in [71] proposed a new service delivery framework, centred on the convergence of Mobile Cloud Computing and future networks for the purpose of improving service delivery in a mobile environment, but his framework lacked analytical modelling and hence intelligence. The author's work strongly advocated the need for low latency to support mobile environments.

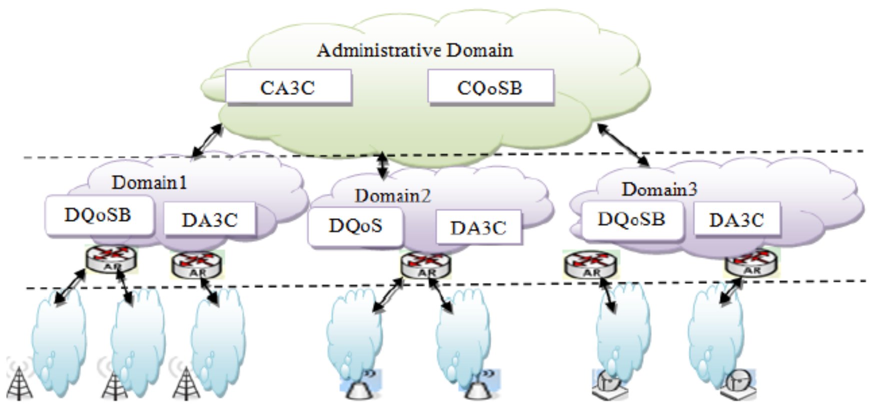


Fig. 2.3 Structure of a Heterogeneous Edge Network Environment

2.6.2 Quality of Service

This section presents the work of Aiash et al. [2] and his work describes the need for QoS model in the mobile environment. It is an example of a heterogenous edge environment which comprises different domains where each domain is dominated by a different network technology. This is important because the edge environment is supported in this thesis. Figure 2.3 shows a more detailed view of the network topology. The Core End-Point represents an Administrative domain (Ad-domain), connected to one or more domains. Although each domain is technology-dependent, cooperation between domains is possible and is managed by Core-End Point.

Similar to [57], for scalable support of Security, QoS and handover in heterogeneous networks, different operating entities exist in the network such as Domain QoS Broker (DQoSB), Core QoS Broker (CQoSB) and A3C servers. These entities collaborate and function on both network and service management levels:

- **A. Core A3C (CA3C):** The top level A3C server resides in the administrative domain and is responsible for service level management. It holds users' Service Level Agreements (SLAs) that contain the subscribed services along with the associated QoS and Network Level Agreements (NLA) which contain the networks and Operators that the user can access with the corresponding QoS. The NLA is passed to the CQoSB for network level management.

- **B. Core QoS Broker (CQoSB):** It plays a major role in managing inter-Administrative domains functions as well as negotiating QoS parameters with other CQoSBs in the case of cross administrative domain connection. CQoSB initially extracts users' NLA from the CA3C.

- **C. Domain A3C (DA3C):** The DA3C is responsible for handling users' service aspects. Initially, it extracts users' profile information from the CA3C and uses this information for authorizing the users' requests to access services.
- **D. Domain QoS Broker (DQoSB):** It gets user profile information from the CQoSB and manages the resources of the attached peripheral networks with respect to users' preferences and network availability, it also makes per-flow admission control decisions. In order to support handover, DQoSB uses a Network Intelligent Interface Selection (NIIS) module for load balancing and handover initiation between peripheral networks. There is an obvious resemblance between the QoSB and the Visitor Location Register (VLR) of the mobile cellular systems.
- **E. Access Router (AR):** This is the link between the domain and the peripheral networks; it enforces the DQoSB's admission control decision.
- **F. Mobile Terminal (MT):** The MT user's device is used to access the network and request a service. To comply with the heterogeneity of 4G systems, the MT should be able to get the subscribed service using the best available access network. Therefore, for the integration of Handover and QoS, the MT contains a mobility decision module called Intelligent Interface Selection (IIS) and a QoS module called QoS Client (QoSC).

  Optionally, some service providers, not shown in Figure 2.3 such as Video On- Demand providers, might reside in the Core-End Point or the Administrative domain; these providers have agreements with the network providers to guarantee the required QoS.

In conclusion, the network topology viewed in this section helps us to have a better understanding of the strategic key point in implementing a more diverse network scenario that would accommodate mobile users in a heavily populated area.

2.6.3 Analytical Modelling

Yonal Kirsal et al. [39] proposed a broad and flexible service delivery framework for Cloud based mobile media environments. As such, an analytical model was established as part of the framework to provide Quality of Experience for mobile users and analyse the overall performance of each network. In this setup, mobile users are able to choose between networks and request a handover to a closer network or request service to be migrated closer to the mobile user. The two scenarios analysed depicted the effect of increasing the network slice provided to the user. Generally, the results were good and showed that the approach is advantageous for the provision of sustained QoS for mobile users by real networks.

With the inefficiency of traditional models of service delivery and lack of scalability to cover mobile users' future needs, the proposed Cloud-based service delivery framework is a stepping stone towards ensuring better solutions to the efficient management of network resources and provision of high QoS for mobile users. The proposed model has the benefit of supporting various kinds of services and applications. It can also help to decrease network congestion on a global scale for frequently accessed websites or large multimedia content. This ensures increased bandwidth especially in streaming scenarios with high-definition media trend. More so, where there is an increased demand for service in the area, migrating the entire service closer to a geographical region will be an excellent import as the type of service usually has interactive content that cannot be cached regionally.

Referring to analytical models for service migration, the authors' paper does not consider the handover process in detail. It does analyse the use of handover in an instantaneous way but in reality, handover does not work that way. In order to perform an intelligent service migration you still need to analyse the communication dynamics.

2.6.4 Service Migration Mechanisms

This section briefly discuses the service migration mechanisms at present in a mobile environment.

Kubernetes Model

This an open-source container orchestration system [59] for automating deployment, scaling and management of containerized applications. It was originally designed by Google and is now maintained by the Cloud Native Computing Foundation. It aims to provide a "platform for automating deployment, scaling, and operations of application containers across clusters of hosts [59]. It works with a range of container tools such as Docker. Though Kubernetes is useful and is widely used in industry, it suffers from lack of QoS support, the use of simple heuristics and no detailed analytical modelling.

Service Management Framework for Mobile Services

It is pivotal to advance a novel service architecture that permits services to be organised, derived or moved to support mobile users. In order to achieve this, the system allows for algorithms that integrate the organisation of traffic and the QoS requests of the flow. As

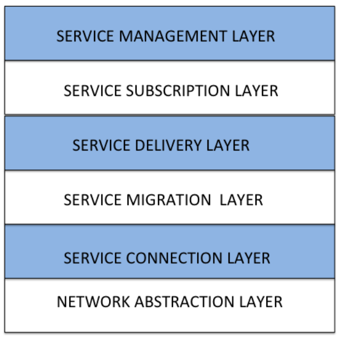


Fig. 2.4 Service-Oriented Framework for Mobile Services

illustrated in Figure 2.4, this novel framework which has six layers was proposed by Sardis et al. [71], and they include:

- **The Service Management Layer:** The function of this layer is to identify the tasks of the service, catalogue the service in a service registry and obtain an exclusive service Identification (ID). In essence, it controls the provided service as it determines the minimum assets required by cloud and networking infrastructures in order to run the service, including network QoS and storage needs as well as computing resources.

- **The Service Subscription Layer:** This layer allows clients to subscribe to services as it takes care of the actions needed by universal clients to access the service. Furthermore, it allocates for a new subscriber an exclusive client ID, a given SLA as well as determines accounting and payment tools.

- **The Service Delivery Layer:** The layer grants a given client access to the service. It does this by mapping the SLA to a given QoS and then certifies that the designated server and attendant networks can match the needed QoS. The service also accepts notifications and prompts regarding handovers and either duplicates or moves the service closer to the user based on received notifications.

- **The Service Migration Layer:** Migration or movement is usually undertaken at the command of the Service Delivery Layer. Here, this layer is in charge of duplicating or moving services to various cloud platforms to encourage good QoE for the mobile user.

- **The Service Connection Layer:** This layer handles the ongoing connection between a client and the service and feeds back alterations in network and transport parameters, the likelihood of changes in the QoS or interruption or suspension to the Service Delivery Layer.

- **The Network Abstraction Layer:** Subject to the network architecture and addressing, this layer oversees the function of getting a service to interface with varied kinds of networks as it maps to IP networking via TCP/IP. The ability to do this is split between the QoS and Transport Layers in Core and Peripheral Frameworks in more progressive systems like Y-Comm.

Service Migration Survey

Authors in [12] compared the characteristics of Unikernel and Container technologies to show which edge computing scenarios are suitable for migration, then summarized lightweight virtualization technologies in edge computing. Based on their survey results, they explained the lightweight virtualization technologies that fit with specific application scenarios. Based on all their findings and results gotten from the most recent work till date, there has not been enough research effort to develop the best intelligent service migration mechanisms.

2.6.5 Solution for Intelligent Service Migration

The above shows the conventional ways that authors have modelled their environment and we know these do not fully meet the Intelligent Service Migration standard for communication due to lack of two or more from the lists below:

- **Low Latency:** This describes a computer network that is enhanced to process a very high volume of data communications with low delay. The lack of low latency in terms of communication simply means that by the time you migrate the service, the MN may have already left due to mobility. Chapter 4 describes an experimental transport protocol called the Simple Light-weight Transport protocol (SLTP) [20] that was used in the research to achieve low latency in communication networks.

- **QoS Feedback Mechanisms:** It helps in the evaluation transmission of the QoS technology that manages data traffic to reduce packet loss, latency, bandwidth and jitter on a network. It basically answers the question of how you know when to move the model in the first instance. Also, SLTP provides good QoS feedback mechanisms, for which results in chapter 4 showed its capabilities.

- **Analytical Models:** These are mathematical models used to define changes in a system that can be stated as mathematical analytical equations relative to previous states. To

Table 2.1 Communication Models Analysis

Model	Low Latency	QoS Feedback Mechanisms	Analytical Model	Service Migration	Orchestration Framework
Aiash	NO	YES	NO	NO	NO
Sardis	YES	NO	YES	NO	YES
Kubernetes	NO	NO	NO	YES	YES
Yonal	NO	NO	YES	NO	YES
Our Research	YES	YES	YES	YES	YES

make an intelligent service migration, we must understand how the communication dynamics affects services delivery especially in the case of handover and loss of service. In Chapter 5, we develop an analytical model that deals with service delivery, handover and the loss of service in vehicular networks. This kind of model is a new idea, with which we were able to get some interesting results.

- **Service Migration Mechanisms:** Service Migration is the ability for a system to move services from one system to another. Chapter 6 buttresses the point that there are certain circumstances that warrants a service to be migrated before a particular standard time. We develop a service migration model and include some results for our claim.

- **Orchestration Framework:** An Orchestration Framework helps to automate, manage coordination of computer systems, applications, and services. A Service Management Framework is developed in Chapter 7 to support and manage complex tasks thereby making service delivery easier.

In conclusion, the list above needs to be satisfied in order for systems to provide an intelligent service migration. Table 2.1 shows what the recent communication models have accomplished and what they lack to enable intelligent service migration in highly mobile environments such as vehicular networks.

2.7 Chapter Summary

This Chapter revealed the technical background which comprises the Y-Comm Reference Framework that supports heterogeneous networking, MEC which describes a mobile edge environment, VANET which houses vehicular networks and discussed the use of different container technologies. Furthermore, it looked at what other research concerning MEC computing, computational offloading decisions for intelligent service migration and then displayed the research gap in this field. Lastly, it reviewed and compared the current communication models and gave a solution approach to achieve an intelligent service migration in vehicular networks.

Chapter 3

Research Methodology

3.1 Brief introduction

This research mainly concentrates on three specific methods to address the issues highlighted in this thesis. Firstly, we will look at analytical modelling whereby an analytical model is developed to capture the key parameters and to create estimated solutions. The second method is to use a testbed that will enable us to perform experiments which will be used to increase our understanding of the problem. And lastly, to explore the use of a Network Memory Server (NMS) in a FUSE file system as a service to show how service migration can be adopted.

3.2 Analytical Modelling

Analytical modelling is a mathematical procedure used for explaining, exploring, and making forecasts about complex processes. In order to construct such a model, an extensive process is undertaken where several parameters affecting the system must be considered accurately in order to achieve a reliable model. In this study, an analytical model has been proposed for vehicular networks using the Zero-Server Markov Chain technique to study the system performance as seen by the mobile user. It introduces a new analytical parameter called probability of lost service.

Furthermore, the queueing analysis was conducted to showcase the benefits of using a proactive service migration and the application of the analytical model developed to the queueing system is explored. The operations involved in such proactive service migration queue management are described in detail and how the relevant parameters are calculated is discussed.

3.2.1 Exponential Distribution

The exponential distribution [4] is a commonly used distribution to describe continuous random variables. It is frequently used to model the lifetimes of different products and times between random events known as interarrival times, such as arrivals of clients in a queueing system or arrivals of goods orders. This type of distribution has one rate parameter.

The major use of the exponential distribution is to model the times between events that occur randomly. It also describes the probability for the times between the events combined with the probabilities for the number of events occurring in each period, known as the Poisson distribution. A Poisson process [4] is used to mathematically describe these events. In this research, the number of requests arriving into the system is modelled using a Poisson process.

3.2.2 Queuing Theory

Queueing theory [33] helps to ease the stress of the most unfriendly experiences of life, which is waiting. At the beginning of the 20th century, Erlang was the first to examine congestion problems. His work inspired engineers and mathematicians to deal with queueing complications with the help of probabilistic methods. Queueing theory grew into a field of applied probability; its results have been used in telecommunications, operations research, traffic engineering, computer science, and reliability theory, etc.

The structure of service and service discipline reveal the number of servers and capacity, i.e., the maximum number of jobs or customers in the system including the ones that are presently being served. The service discipline regulates how the next customer is selected for service. The most generally used service disciplines are Last In First Out (LIFO) in which who comes later leaves earlier, Random Service (RS) where the customer is selected randomly, and First In First Out (FIFO) where who comes prior leaves earlier, etc. The interarrival and service times of jobs are usually independent random variables. The objective of analytical models which is based on queueing theory is to acquire the performance measures of the system which are probabilistic. The properties such as distribution function, density function, variance and mean [33] of the following random variables: number of customers in the system, waiting time of a customer, idle time of the server, the response time of a customer, utilization of the servers, busy time of a server, number of waiting customers, the average number of people in the system and the blocking probability are explored using the queueing theory. The result severely depends on the assumptions concerning the distribution of service discipline, service times, number of servers, capacity and interarrival times.

Kendall‘s Notation

A notation founded by Kendall [8] to describe a queueing system is displayed below:

(3.1)

where

- **A:**distribution function of the inter-arrival times
- **B:**distribution function of the service times
- **m:**number of servers
- **K:**capacity of the system, the maximum number of customers in the system including the one being serviced,
- **n:**population size, number of sources of customers,
- **D:**service discipline.

Queuing theory can be explored by measuring, modelling, and analysing the service times arrival times, and wait times of queuing systems.

Markov Chain Model

A Markov chain [33] is a mathematical model that was named after A. A. Markov, which is a mathematical system that hops from one state to another. A system process is called continuous or discrete state dependent on the values its state can receive. When the number of likely values are countable or finite (e.g. discrete values such as 0,1,2,. . . ,N), the system is called discrete state. The system process waiting time can take any value on a queue, therefore this is defined as the continuous state process.

If the future states of a system process are independent of the past and changes only on the present, the process is called a Markov process. Its property makes a process easier to analyse since there is no need to remember the complete past trajectory. Therefore, a discrete-state Markov process is called a Markov chain.

Markov Chain analysis will be used to develop the analytical models in this thesis.

3.3 Experimental Testbed

The distribution of Connected and Autonomous Vehicles (CAVs) will change the environment we live in. In general, Connected Vehicles permit us to build an ITS by enabling strong connectivity among vehicles and the transport infrastructure. This is referred to as Cooperative-ITS (C- ITS). The deployment of C-ITS will result in better traffic and road management, fewer accidents, shorter journey times, better collision avoidance mechanisms and increased efficiency to manage major disasters.

For better understanding nowadays, the building of new technology is a necessity; the use of testbeds and applications will give us a better understanding of this new era. Middlesex University and the Department for Transport (DfT) have built a Connected Vehicle Testbed that uses ITS-G5 (VANET) technology.

This section provides the details of the real experimental VANET testbed. The Connected Vehicle Testbed was built by Middlesex University and the DfT using ITS-G5 technology. The testbed was built on the Hendon Campus in London and alongside the surrounding roads and then extends to the A41 (Watford Way) behind the campus. Four RSUs which were deployed in the MDX buildings were backhauled directly to the university's gigabit ethernet network and the three RSUs deployed along the A41 were backhauled using LTE with a secure Virtual Private Network (VPN) tunnel service provided by Mobius Network as shown in Figure 3.1. They are now fully operational and trials have been held to fully understand the technology and concerns around its wide-scale deployment as well as communication dynamics to attain seamless communication for this environment.

3.3.1 Designing and Implementation of VANET Testbed

In this section, the process of the testbed deployment at Middlesex University (MDX) is shown. In compliance with the IEEE 802.11p (WAVE) standard specifications and the required maximum output power, 200mW or +23dBm was used [74]. The testbed consisted of seven RSUs, with three testbeds by the A41 road behind the university and four on the MDX buildings. The channel employed to send transmissions was CH172, while the operating frequency of the RSUs was 5.9 GHz. Both the RSUs and OBUs were produced by Lear Corp. The below Figure 3.2 illustrates the respective GPS coordinates and location names.

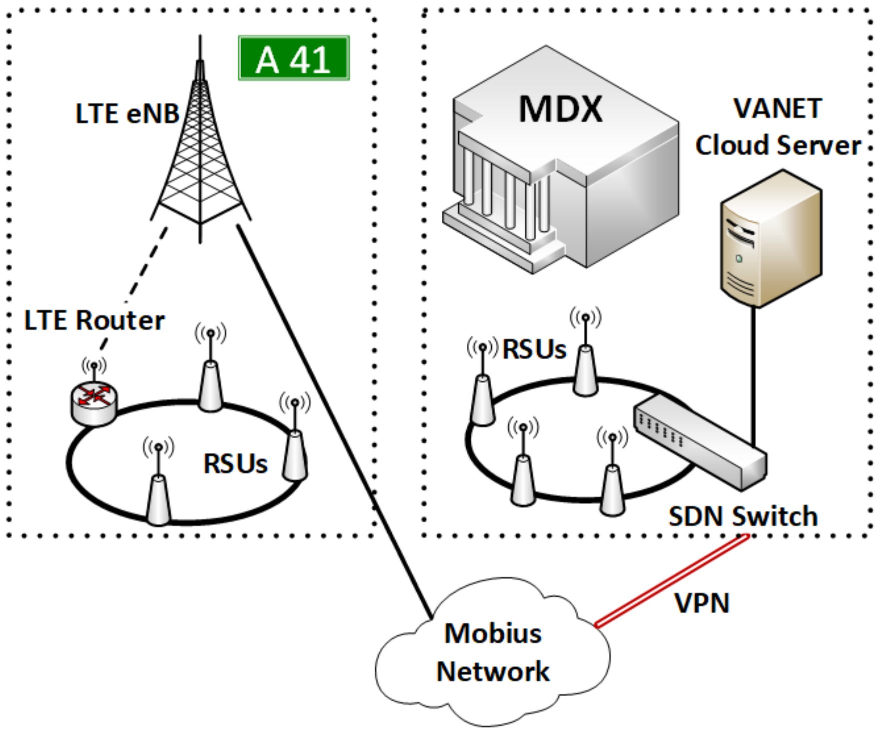


Fig. 3.1 MDX VANET Testbed Network Diagram

RSU	Location	Latitude	Longitude
1	Grove Building	51.588837	-0.230769
2	Sheppard Building	51.590808	-0.229672
3	Williams Building	51.590494	-0.228594
4	Hatchcroft Building	51.589073	-0.228374
5	Lamp Post (No.6)	51.586145	-0.231452
6	Lamp Post (No.42WW)	51.591779	-0.234701
7	Lamp Post (No.87WW)	51.599255	-0.233245

Fig. 3.2 RSU Location Information

3.3.2 MDX Deployment

It was expedient to reduce the distance between the RSU and the router components in the university network, hence, the best location for the RSU at Middlesex University had to be discovered. As illustrated in Figure 3.3, this enables the direct backhauling of data to the central MDX VANET Server positioned in the basement of the Sheppard library from the RSU employing the university network. To study the mobility of vehicles and pedestrians using Vehicle-to-Pedestrian (V2P) applications, four RSUs were installed at different locations at MDX - Williams building, Hatchcroft building, Grove building and Sheppard building - to accommodate areas across the MDX campus.

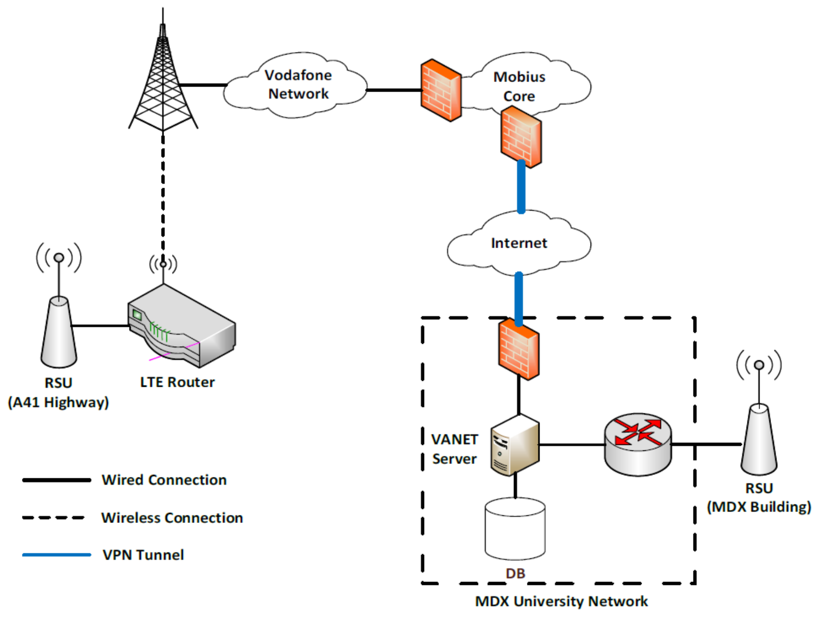


Fig. 3.3 Network Diagram

3.3.3 A41 Deployment

The deployment had to be achieved in collaboration with Transport for London (TfL) with three RSUs mounted on lamp-posts along the A41 (Watford Way) A41. These were positioned to considerably expand the coverage of the Middlesex testbed. An LTE router was used as the data accepted by the RSU was backhauled as a result of inadequate communication network systems on the lamp-post. Furthermore, Figure 3.3 also shows that a VPN tunnel service made available by Mobius Networks through Vodafone was used owing to security constraints on the MDX network.

3.3.4 Application Description

There are various forms of communications used to convey different information within the OBUs and RSUs such as Basic Safety Messages (BSMs), Cooperative Awareness Messages (CAMs), Intersection Collision Alerts (ICAs), Road Side Alerts (RSAs), and Probe Vehicle Data (PVD).

The BSM message, which pertains to safety applications, was the first communication to be analysed. In this thesis, BSMs are broadcast at 10 Hz to the RSU from the OBU and have a minimum length of 41 bytes. BSMs comprise data relating to movement, location, and the overall state of the vehicle which is illustrated in Figure 3.4 [68]. However, only the Message ID and 3D Position – Longitude, Latitude and Elevation constraints of the BSM message were used as other fields such as acceleration set and steering wheel angle had to be collated from the vehicle using sensors and other mechanical devices before being joined to the BSM packet and broadcast. However, this complete setup was not within the scope of this thesis.

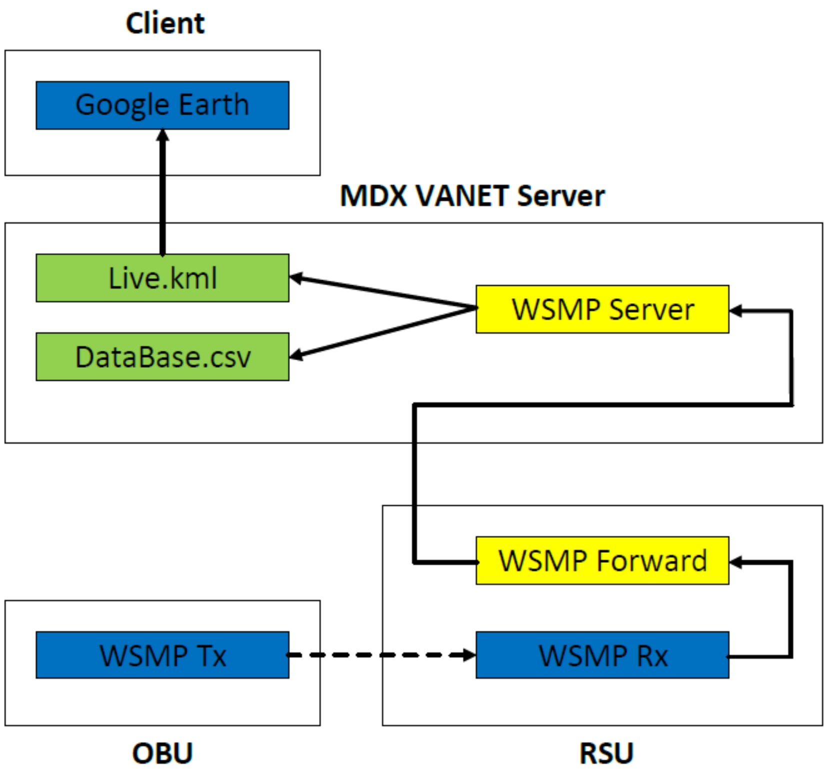


Fig. 3.4 Packet forwarding from RSU to Server

To get readings from the OBU to the VANET Server, different applications were used. The RSU obtained packets using Wave Short Message Protocol (WSMP) Receiver (Rx) application while the WSMP Transceiver (Tx) was used by the OBU to transmit the BSM packets. As illustrated in Figure 3.4, using the WSMP Forward application via an IPv6 address of the server, the obtained packets by the RSU were trasmitted to the server. The MDX VANET Server received the packets while saving the data with added information such as timestamp and the RSU's IP address. This is achieved using a WSMP Server application.

Two different files were used to save the received data: Live.kml and Database.csv. The live.kml comprised the current or immediate locations of each OBU through the packets received from the OBU and stored in the Apache Web Server space for direct access. Also, the live tracking of the OBUs was realised using Google Earth and the live.kml file network link. A MySQL database was used to back up the Database.csv file daily for analysis. The other file, Database.csv comprises a great anount of available information in the packets like the GPS coordinates within the time stamp of the packet, OBUs MAC address, the received signal strength indicator (RSSI) range of the received packet and IP address of the RSU by which the packet has been transmitted.

3.3.5 MDX and A41 Coverage Map Result

Figure 3.5 illustrates how the unique GPS coordinates in the packets are sent by the OBUs and received by the MDX VANET Server. On 17th May 2017, the coverage was mapped from the trial data for a moving car within the MDX campus as shown on Figure 3.5. The individual coverage ranges attained by the RSUs located along the A41 road and RSUs located on each MDX building are shown in the coverage map using various coloured dots. The Hendon Campus and surrounding roads were covered by the first four (1-4) RSUs - the area covered was about 0.7 miles/1.1 km. The other (5-7) RSUs covered the A41 as the coverage was from the between the entrance of the Great Northern Way (above the brown line) to Hendon Central Tube Station (underneath the blue line) – with a distance of 2 miles/ 3.2 kms. As such, the total coverage of the testbed was approximately 2.7 miles/ 4.31 kms. The MDX testbed was fascinating, allowing for an inclusive investigation and examination of signal propagation for RSUs installed by the road, on buildings – MDX campus - and lamppost as it includes motorway and urban roads. It was observed that there is a significant positive increase in coverage for the RSUs with elevated installations and clear Line-of-Sight (LoS) for the projected roads [26]. Due to the elevation of the RSU deployment, the coverage was more enhanced than projected.

3.3.6 The Relationship Between Analytical Model and Testbed

In this work we are not trying to validate the analytical model, instead we will be using the analytical model to study the communication dynamics and the service migration in highly mobile environments such as vehicular networks.

3.4 The File system

File systems [28] play a crucial part in every operating system. It is a place where users keep their files. The organization of the file system also plays an important role in helping the user find files. There are many file systems that have been developed by different programmers.

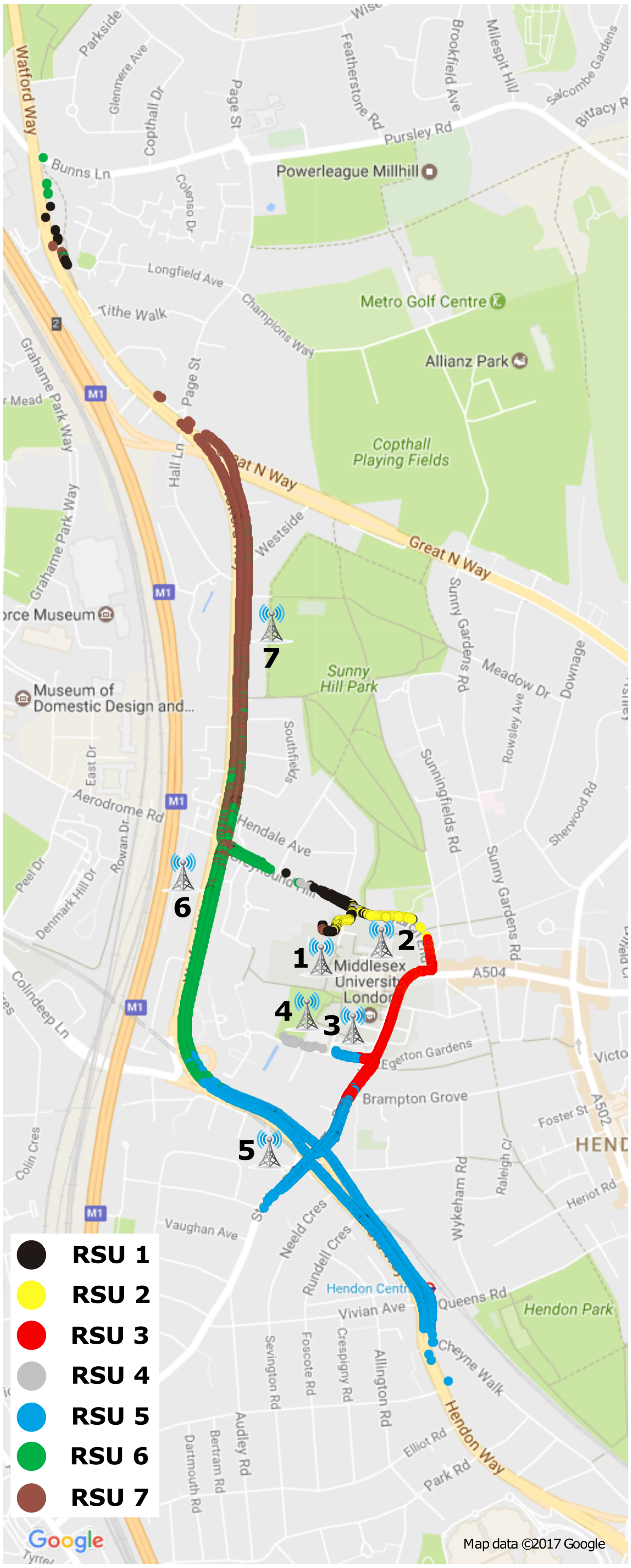


Fig. 3.5 Full Coverage and Overlapping Map for A41, Watford Way, Hendon, London

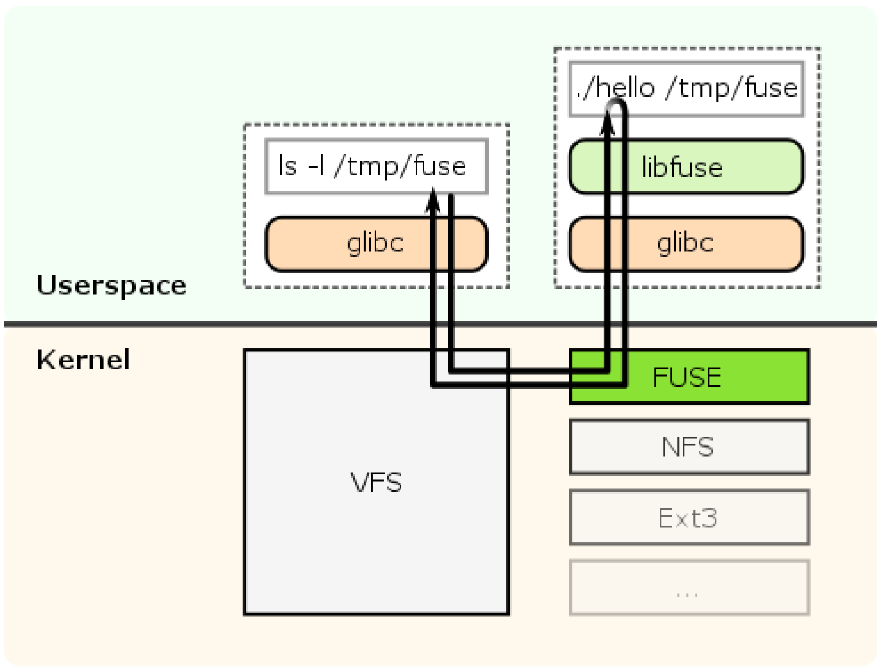


Fig. 3.6 How FUSE works

Every different file system has its own advantages and disadvantages. It is vital for a user to select the most suitable file system. Having a suitable and appropriate file system enables the computer system to operate at a higher efficiency. Additionally, a file system allows the user to attach special attributes to the file such as the owner of the file, the permissions over the file. The most common file system that is used by end users is NTFS which is short for New Technology File System. Choosing the correct file system is important in order to maximise the computer systems' performance. Some file systems have special features, some offer better reliability and robustness and some provide quicker read and write speeds. Some file systems are part of the operating system and therefore execute in kernel space. User-space file systems are different because they execute in user space along with the application. This provides better flexibility as the kernel is complicated.

3.4.1 Filesystem in Userspace (FUSE)

The FUSE [28] is a software interface that provides a bridge between user space and kernel space. This allows the file system to be placed in the user space and hence outside of the kernel.

According to Layton et al, the illustration in Figure 3.6 describes a file system named **hello** is compiled and being executed. When **hello** is executed, the FUSE mounts the "test" file system in the directory "/tmp/fuse". Here, the user can store his/her data using the **hello** file system. All the data will be stored in the user space directory “/tmp/fuse”. Then the user executes an “ls –l”. This commands go through the glibc to the VFS in the kernel. The VFS then goes to the FUSE module. The FUSE module will contact the **hello** file system through the glibc and libfuse (which is the FUSE library in user space) and asks for the result of the command. The result will then be returned back to the FUSE module and passed through VFS and finally to the “ls –l” command.

3.4.2 Network Memory Server

The Network Memory Server (NMS) [29] is an example of a simple, stateless service; it stores blocks of data from clients in its memory (RAM). Clients can create, read, write and delete blocks of data. The NMS was primarily designed as a storage platform for mobile users. In order to provide support for mobility, the NMS is divided into two parts: The Mobile Memory Cache (MMC) and the Persistent Storage Server (PSS). The MMC initially runs on the same network as the mobile client. If a client moves to another network then the MMC is migrated in order to achieve better performance. The PSS offers permanent data backup for the MMC and there is a level of redundancy implemented so that an MMC can be backed up in multiple instances of the PSS. This is achieved by a multicast call to all the associated PSS. The Network Memory Service is an example of a mobile service used in this research.

3.4.3 Mobility of the NMS

Mobility is an important part of the NMS, the clients will need to be able to access its content from anywhere and at any time regardless of the physical location. Mapp et al. [57] clarify the problem and provide some solutions in providing mobility for a storage server and its clients. Researchers have found that the Context Transfer Protocol (CXTP) will enhance the mobility of the NMS [23], for which the illustration in Figure 3.6 describes NMS architecture.

In this research, we will use the NMS to provide network backing storage for the FUSE file system on the MN. We will use SLTP as the protocol to facilitate the movement of the NMS between RSUs. Hence, we will build a platform in order to migrate the NMS using container technologies. The NMS is a service that has already been researched and implemented.

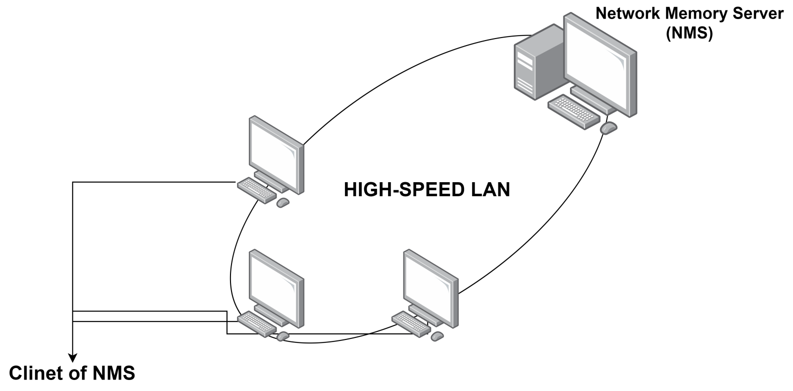


Fig. 3.7 Network memory architecture for memory server

3.5 Chapter Summary

This chapter has successfully shown the three major methodologies used in this research i.e. analytical modelling, a constructed testbed used to conduct our experiments, and the use of NMS in a file system as a service to show how we could intelligently migrate services across the network in the VANET testbed. This is important to prove the application and validity of the proposed intelligence service migration in a real-time scenario for vehicular networks.

Chapter 4

Investigating a Low Latency QoS Aware Environment

4.1 Brief Introduction

This chapter will explore the implementation framework of the Y-Comm Architecture, then it discuss a new application framework based to the implementation framework which consists of five layers. Finally, it will detail a new experimental transport protocol which provides low latency and QoS and thus is suitable for vehicular networks.

4.2 Y-Comm Implementation Framework

Mapp et al. [53] designed a set of collaborative mechanisms as an Implementation Framework for Y-Comm shown in Figure 4.1.

The Peripheral Implementation Model consists of:

- **IEEE 802.21 mechanism:** This layer makes use of the IEEE 802.21 mechanisms to control the various wireless interfaces.
- **Proactive Handover:** This does proactive handovers using Network Dwell Time (NDT) and Time Before Vertical Handover (TBVH).
- **Mobility Management:** It is based on GPS readings and parameters as well as triggers from network interfaces.
- **SLTP:** This is a new transport protocol between the Mobile Node and the Base Stations.

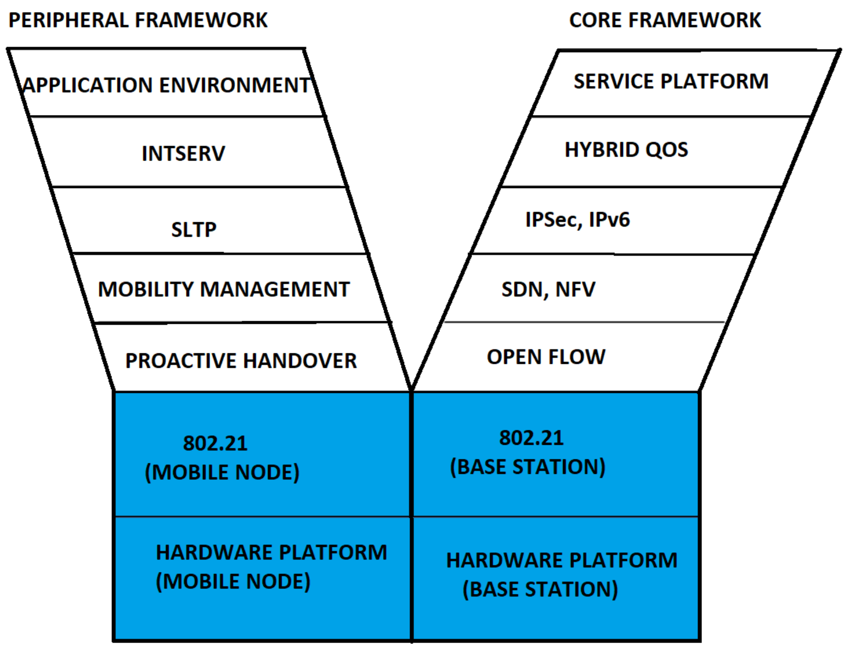


Fig. 4.1 The New Implementation Model for Y-Comm

- **IntServ:** It helps the mobile node to specify its QoS requirements on a per-flow basis.
- **The Application environment:** It uses an enhanced IntServ specification to specify its QoS requirements on different connections.

While the Core Implementation Model consists of:

- **Base Station Controllers running 802.21:** This permits the Access Routers and Base Station to be managed using an enhanced IEEE 802.21 protocol.
- **OpenFlow:** It is used for reconfiguring the access network routers based on the mobility of the user.
- **Software Defined Networking (SDN) and Network Functional Virtualization (NFV):** This is used to implement the mechanisms and services for the Network Management Layer.
- **IPSec, IPv6:** The TCP/IP Suite is maintained but IPSec [15] is used to ensure that secure tunnels can be set up between Core-End Points.
- **Hybrid QoS:** The integration IntServ and DiffServ mechanisms is done at this layer.
- **Service Platform Layer:** It executes a Mobile Services platform where servers can be moved closer to mobile users.

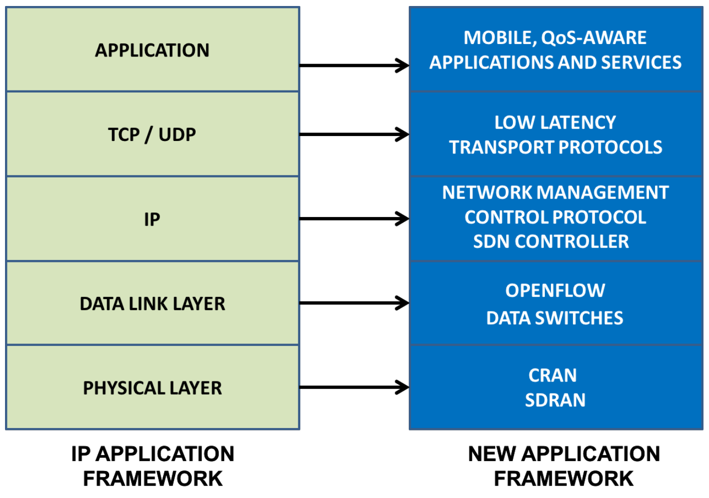

APPLICATION
TCP / UDP
IP
DATA LINK LAYER
PHYSICAL LAYER
IP APPLICATION FRAMEWORK
MOBILE, QoS-AWARE APPLICATIONS AND SERVICES
LOW LATENCY TRANSPORT PROTOCOLS
NETWORK MANAGEMENT CONTROL PROTOCOL SDN CONTROLLER
OPENFLOW DATA SWITCHES
CRAN SDRAN
NEW APPLICATION FRAMEWORK

Fig. 4.2 New Framework for Building Future Applications

4.3 The New Application Framework

Figure 4.2 shows the new Application Framework, it draws its inspiration from the Y-Comm implementation framework. This consists of five layers which are detailed below:

4.3.1 CRAN SDRAN

The proposed system will evolve with the development of new mobile technologies as shown in Figure 4.3. This evolution will allow the smooth management of local heterogeneous networks by the Heterogeneous Cloud Radio Access Network (H-CRAN) and the Cooperative Radio Resource Manager (CRRM). H-CRAN will be used to access and control individual networks while CRRM will be used to optimize the overall radio access environment. CRRM will also support OpenFlow and hence the upper layers of the architecture can remain unchanged. The use of NFV and SDN at the Core-End Point will also facilitate the softwarisation of radio technologies as proposed in 5G with the deployment of Cloud-RAN [89] at the Core-End Point.

4.3.2 OpenFlow Data Switches

As shown in Figure 4.3 the SDN controller controls access to both the H-CRAN and Open-Flow Ethernet data switches. The controller interfaces to the upper layers using NETCONF interface using the YANG data model [7].

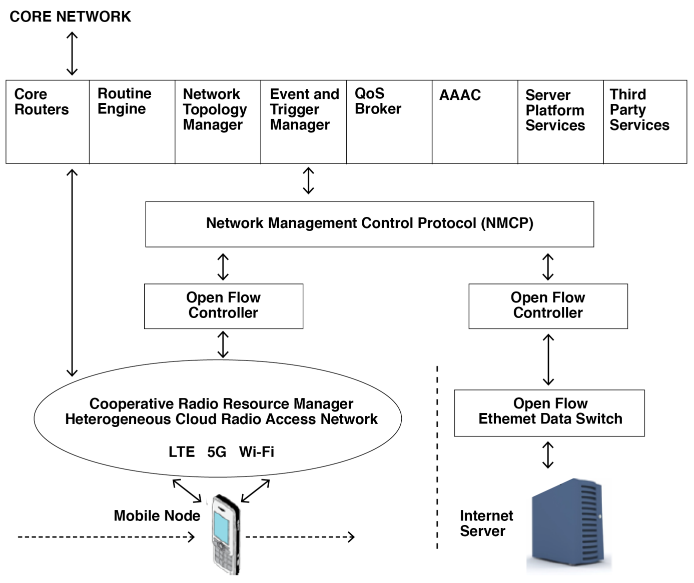


Fig. 4.3 The Operational Structure of a Core-End Point

4.3.3 Network Management Control Protocol SDN Controller

The Network Management Control Protocol (NMCP) is used to allow the high-level network management functions and services discussed above to control and manage networking infrastructure. NMCP can be implemented by directly translating it into OpenFlow commands or by using a number of emerging Northbound APIs. NMCP also supports various communication entities such as endpoints (IPv addresses), links, a path (one or more links), and data flow.

Connections in NMCP work by specifying links between endpoints and core network elements such as Core Endpoints as shown in Figure 4.4. Each link involved in the connection is specified using a TUPLE which specifies endpoints on the link as well as a forward connection label (fcl). An fcl is needed to forward any packet along a link and is treated as a capability and hence cannot be tampered with. The fcl specifies which addresses should be used to communicate over a link. Once the links between the end points are specified, it is possible to create a path using the TUPLEs specified. The data flow between endpoints represents the data being exchanged and is specified as a flow along a specified path. Once this is done the connection can be activated and the two end points can send data with each other.

The improvement of NMCP over IP is that it is not bound by a specific address format. The system decides which network technology can be used on each link, and this arrangement is able to readily adjust to changes in the network topology. In addition, by making data-flows first class objects we also need not associate them with any network technology; that allows

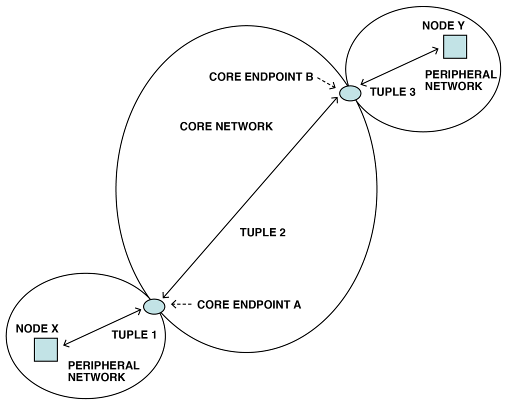


Fig. 4.4 Connection setup using NMCP

us to implement things such as vertical handover because we can easily specify that a flow can be changed to go on a different path. Finally, the fcl can be used to specify a given quality of service required by applications using a given connection. This means that it is easy to find out if the QoS on the network is being broken.

4.3.4 Low latency protocol

New networks such as VANET networks require low latency and high bandwidth. In addition, new features can be used to fine tune transport protocols to application requirements. These includes:

- **Running Efficiently in User Space:** Since the transport protocol should be under the direct control of the application, it must run efficiently in user-space. Though running protocols in the kernel has advantages such a lower latency and guaranteed CPU cycles as kernel code normally executes at a higher priority than user space, running transport protocols in the kernel results in a huge amount of cross talk for all applications. So this means that a reliable fast video stream could be affected by activities from other applications. Secondly, because of the development and proliferation of multiprocessor architectures, which are now common even on PCs, it now very hard to argue that there is not enough CPU cycles in user-space to run transport protocols efficiently. Running in user-space will eliminate transport crosstalk and allow applications to be able to directly tune protocol parameters without the need for obscure socket system calls. Support for user space protocol processing is being actively pursued by several companies including TCP Offload and the Data Processing Data Kit (DPDK) [77] initiatives.

- **Selective retransmission by default:** Protocols such as TCP use the go-back-n mechanism which can result in many packets being retransmitted even though they have already been received at the other endpoint. So it is necessary that future protocols implement Selective Retransmission by default.
- **Variable Reliability:** Applications should be able to apply different reliability characteristics to different connections using the same protocol. So a transport protocol should be able to provide the entire spectrum: from totally reliable to totally unreliable connections. Therefore, support for forward error correction is allowed.
- **Support for Forward Error Correction (FEC) functionality:** Most transport protocols provide checksumming and retransmission of packets to assure reliability. However, for applications that require low latency, retransmissions are seldom beneficial. In this situation, FEC techniques are used to ensure reliable reception. So streaming network audio could use FEC rather than just dropping corrupted packets.
- **The ability to tune specific aspects of the protocol:** This becomes very relevant for certain operations. So one parameter that should be changeable is the window size of a given transport protocol. This may be due to buffering issues but it could be used to support other communication events such as handover [77]. So when a handover to another network begins, the protocol closes its window preventing other packets from being sent until the handover takes place where its window size can be re-opened. Other parameters can be indicated are the maximum message size, etc.
- **Support of priority for different end-to-end data flows:** This has become a key issue as different types of data flows are being transported and so there might be times when you want to send packets on certain connections with different priorities.
- **Up calls from the transport protocol into the application:** Most transport systems use PUSH-PULL mechanisms developed by the traditional socket layer libraries where senders transmit or PUSH data towards the client while receivers retrieve or PULL the data from the underlying socket for the connection. However, in many cases, a server may wish to simply provide an upcall on the receipt of a service request message from the client.
- **Providing alternative for flow control:** In current transport protocols, applications have no say how flow control is done. TCP uses a sliding window based on congestion and receive window parameters as well as slow start and congestion avoidance algorithms. These mechanisms have proven to be effective but at times have been too conservative.

4.4 Simple Lightweight Transport Protocol (SLTP)

SLTP is an example of a low latency lightweight protocol that has been designed to support the new QoS-Aware framework. This motivation for designing SLTP came form the need to support research into services using Cloud based environments [56] as well as to provide low latency and tuneable support for Vehicular and Haptic Networks.

The SLTP Header

Figure 4.5 shows the Diagram of the SLTP while Table 4.1 shows the length of the individual fields.

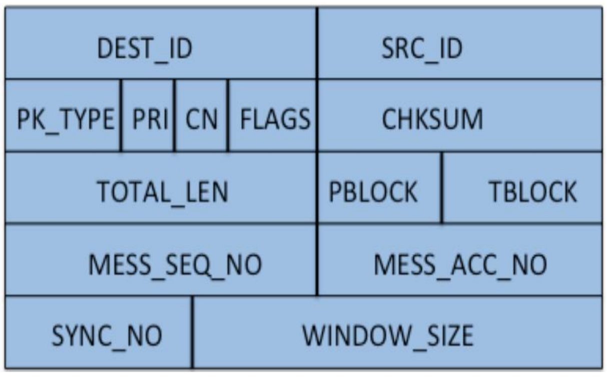


Fig. 4.5 The Structure of SLTP Header (Total Size is 20 bytes)

SLTP Packet Types

SLTP supports a number of packet types as shown in Table 4.2.

SLTP Flags

SLTP FLAGS comprises a field containing 8 bits. Their functions are detailed in Table 4.3.

In SLTP, when a connection is started, each side measures the bandwidth and burstiness of the connection through a modified packet-pair approach in which two packets of a given size are sent back-to-back and the round-trip times of each packet is measured as well as

Table 4.1 The fields of SLTP and their functions

FIELD	BITS	FUNCTION
DEST_ID	16	Connection_Id at the remote end
SRC_ID	16	Connection_Id at the local end
PK_TYPE	4	Type of packet
PRI	2	Priority of the packet
CN	2	Congestion Notification Indication
FLAGS	8	Indicates actions needed to process the packet
CHKSUM	16	Uses the TCP Checksum
TOTAL_LEN	16	Total length of the packet
PBLOCK	8	Current block or fragment
TBLOCK	8	Total number of blocks in the message
MESS_SEQ_NO	16	Sequence number of the last message sent
MESS_ACC_NO	16	Sequence number of the last message received
SYNC_NO	12	Random number to prevent replay attacks
WINDOW_SIZE	20	The Receive Window Size

Table 4.2 Packet types and their functions

PACKET_TYPE	FUNCTION
START	First packet transmitted on a connection
REJECT	Signals that the connection request has been rejected
DATA	Data packet
ACK	Acknowledgement (ACK) packet
NACK	Used for selective retrnsmission
END	Used to close a connection
FIN	Final packet sent
ECHO	Used to measure RTT
ECHO_1	First back-to-back packet
ECHO_2	Second back-to-back packet
STATUS	Used to maintain flow control
IDLE	Sent when there is no data to send
CWIN	Used to change the window size

the time-difference, d, between the packet replies as shown in Figure 4.6. In SLTP, ECHO 1 and ECHO 2 packet types are used to perform this test. The diagram shows how bandwidth is measured in SLTP, here two packets are sent from source to destination. Here, t1 and t2 are the times when the packet has started being sent from the source and t3 and t3 are the times when the ECHO 1 and ECHO 2 are received back at the source after being echoed by the destination. The Round Trip Time (RTT) is defined as the time in which a packet sent,

Table 4.3 Flags and their functions

BIT	NAME	FUNCTION
0	W_VAL	Window-Size is valid
1	ST_CKS	Checksum this packet
2	ST_RTR	Retransmission is permitted
3	ST_RET	Indicates a retransmitted packet
4	REMOTE_RESET	Connection reset by the other side
5	REPLY_REQ	A reply is requested
6	REPLY	Reply to a previous request
7	EOM	Last message was correctly received

receive, and acknowledged. RTT is given as (t3- t1) or (t4-t2) and therefore the bandwidth will be given as S/(RTT/2) where S is defined as the size of the packet. From Figure 4.6, d is the time difference between the two replies received by the source i.e., t3 and t4 for packet A and B respectively. SLTP uses packets to measure the burst on a connection, where the burst is the maximum amount of data you can send in one transmission to keep the pipe or channel at maximum bandwidth capacity to keep the connection fully engaged at a full rate. At that value, the maximum bandwidth for the connection is achieved, i.e., the pipe is full. Hence sending data beyond the burst value will cause packets to be dropped. Furthermore, we work out the maximum burst for a connection by saying that if the packets get separated by d, then the maximum number of packets you can burst is (RTT/d). Therefore with this formula, we calculate our burst size to be (S*RTT/d), where, S is the size of the packet. And so we set the maximum unacknowledged packets to the calculated burst size. When this value is reached, the sender should stop sending and wait for an acknowledgement because sending more data will likely result in packet loss.

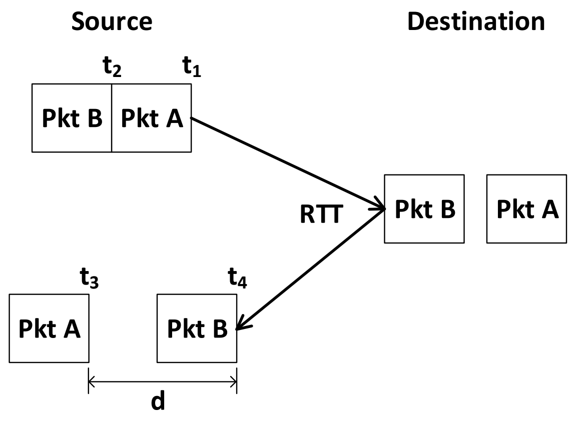


Fig. 4.6 SLTP Bandwidth and Burst Size Calculation

Preliminary Result on SLTP

In order to fully analyse the effect of SLTP, we obtained a set of results that looked at the cost of communication between a client and a server. This is directly dependent on the transport protocol being used. Results were obtained when the client and server are connected via an Ethernet Switch and when they are connected via a router.

Since SLTP runs over User Datagram Protocol (UDP), the size of a single SLTP packet can be up to (64KBs - 8 bytes (the size of the UDP)). However, for testing we wanted to ensure that SLTP packets could fit into a whole number of Ethernet packets which can carry a payload of 1500 bytes. For larger UDP packets we took into account IP fragmentation over Ethernet. So we varied this parameter as follows:

- **SP 0** represents (1500 - (IP header size (20) + UDP header size (8) + SLTP Header Size (20))) = 1452 bytes
- **SP 4** represents (SP 0 + (4 * (1500 - 20)) = 7372 bytes or 7.2 KBs
- **SP 8** represents (SP 0 + (8 * (1500 - 20)) = 13292 bytes or 12.98 KBs

Finally, for these tests we used a window size of 144 KBs;

Results for Traditional Networking Environment

We performed our benchmarks by using two PCs equipped with the following hardware:

- Processor: Intel(R) Core(TM) i5-3770 CPU (4 cores).
- RAM: Both PC with 16GB DDR3
- Storage: Both PC with 320GB HDD
- Network: 1 Gigabit Ethernet Cable
- OS Type: Fedora 25 64-bit
- Router: CISCO 1941 Series
- Switch: NETGEAR Gigabit desktop switch GS108

Figure 4.7 shows the results when the connection is going through an Ethernet Switch and Figure 4.8 shows the results when the connection is going through a router with the client and server on two different networks of the router. These results show that SLTP generally performed better than standard TCP. These results at least indicate that modern systems now have enough resources in terms of CPU, memory and networking to allow user-space protocols to run efficiently.

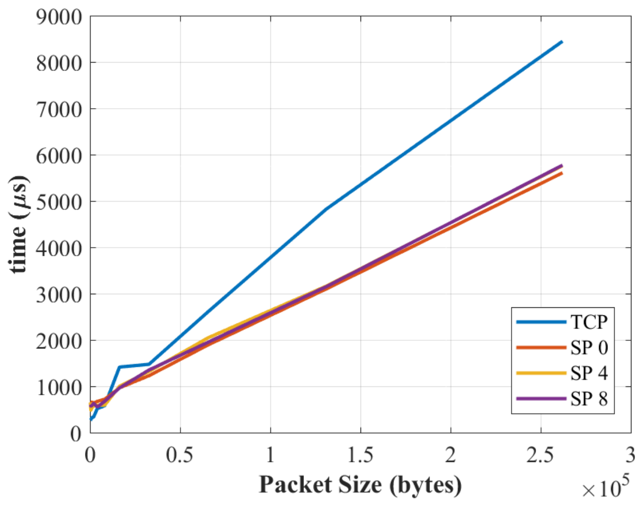


Fig. 4.7 PC to PC (Switch) - Network Time: TCP vs SLTP with SLTP packet sizes

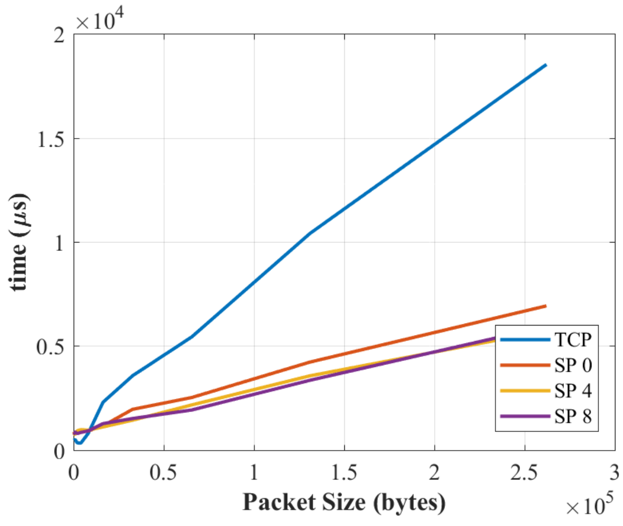


Fig. 4.8 PC to PC (Router) - Network Time: TCP vs SLTP with SLTP packet sizes

VANET Server and RSU

In this section, we present the results for RSU to VANET communications. We performed our benchmarks by using the following hardware specifications as shown in Table 4.4. It can be clearly seen that the resources on the RSU are quite constricted compared with the VANET server or a modern PC.

The result in Figure 4.9 show that in this organisation, less CPU cycles as well as memory are available on the RSU and hence, the performance of SLTP is not much greater than the standard TCP.

Table 4.4 Hardware Specifications

Specifications	**VANET Server**	**RSU**
Processor	Intel (R) Xeon (R) CPU E5-2683 v4	MIPS 24Kc V7.4 (1 core)
RAM	32GB	64 MB SDRAM (512 Mbits)
Storage	500GB	16 MB Flash
Network	1 Gigabit Ethernet	1 Gigabit Ethernet
OS Type	Debian 3.16.43(64 Bit)	Debian 2.6.32.27 (32 Bit)

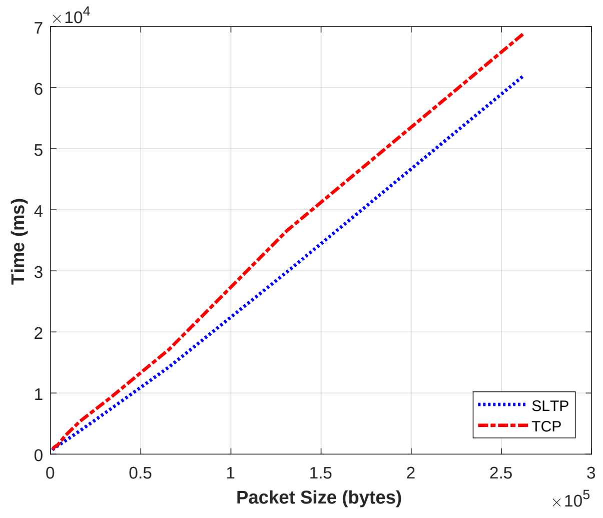


Fig. 4.9 RSU to VANET Server - Network Time: SLTP vs TCP

RSU to RSU Link Performance

The results for RSU to RSU are given in Figure 4.10. These results show that with two RSUs, it has very limited resources at user level and hence, TCP in the kernel slightly outperforms SLTP as more resources are available in the kernel when running on smaller systems, compared to running in user space.

RSU to RSU Link Performance Under Different Load Conditions

Since, SLTP runs in user space, it is important to understand how its performance is affected by different load characteristics of the system. In order to explore this, a flexible hog program was used to remove idle CPU cycles at user level from the system. Hence, we were able to obtain readings with the system being under various loads, including 25%, 50%, 75% and 100%.

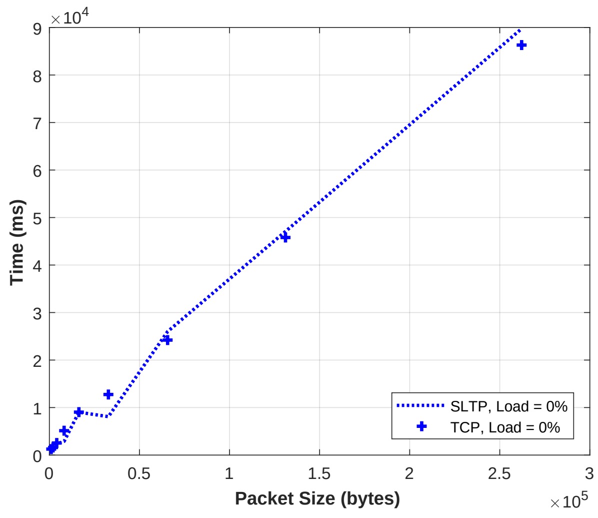


Fig. 4.10 RSU to RSU - Network Time on Load = 0%: SLTP vs TCP

The bandwidth results under different loads as shown in Figure 4.11 reveal that there are only significant differences for small packet sizes. However, after around 2KBs the bandwidth available falls to around 2.5MB/s. This is important for applications needing large packet transfers sizes such as multimedia applications.

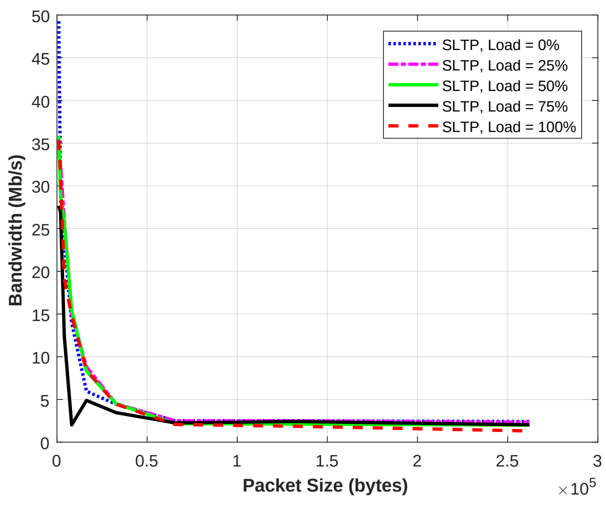


Fig. 4.11 RSU to RSU - Bandwidth on Load: SLTP

The latency results as measured by SLTP using different packet sizes under different loads are shown in Figure 4.12. It shows that the latency increases with increasing load especially after 50KBs.

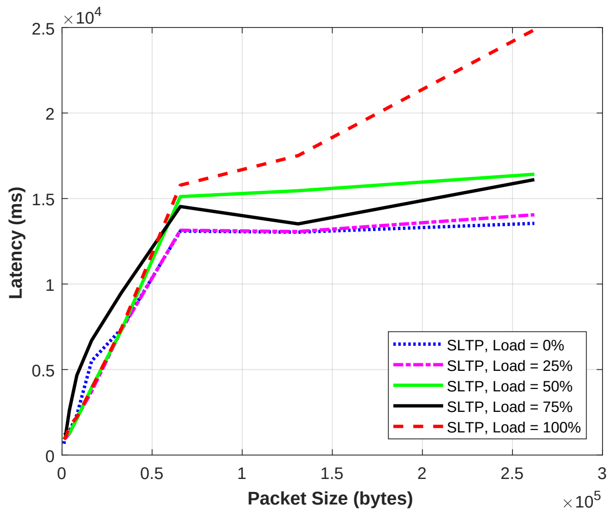


Fig. 4.12 RSU to RSU - Latency on Load: SLTP

The burst results as shown in Figure 4.13 clearly show that the system is affected by high loads especially for small packets. After around 10KBs the burst size is severely reduced.

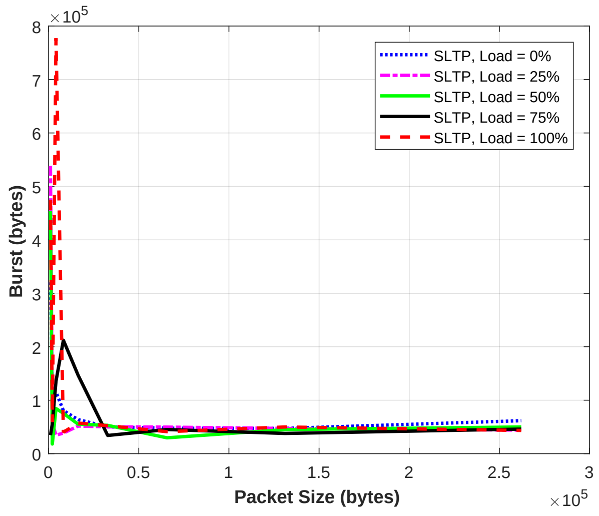


Fig. 4.13 RSU to RSU - Burst on Load: SLTP

Finally, Figure 4.14 shows the time taken to transfer for different buffer sizes under different loads and it clearly shows as the load increases SLTP underperforms TCP as less cycles are available in user space. However, this effect is only significant at very high loads.

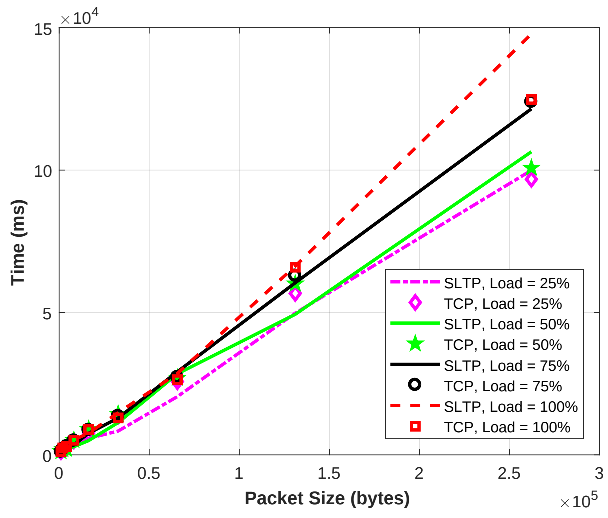


Fig. 4.14 RSU to RSU - Network Time on Load = 25% to 100%: SLTP vs TCP

4.5 Chapter Summary

The implementation framework of the Y-Comm architecture gave a better understanding of the discovery of the New Application Framework. This Application Framework provided a low latency, QoS-aware environment that would enable intelligent service migration. Furthermore, a comprehensive analysis of SLTP was shown, performance tests were carried out to show its validity. The results reveal by SLTP were compared with the standard TCP. Also, they both went through a stress test with different loads on the RSUs network and VANET server in the Middlesex University. The results showed SLTP had better performance but also in order to do intelligent service migration, a detailed analytic model which looks at the communication dynamics of vehicular networks must first be developed. Therefore, the next chapter focuses on that.

Chapter 5

Modelling Handovers in Vehicular Networks for Intelligent Service Migration

5.1 Brief Introduction

In order for a system to implement intelligent service migration it must understand the communication dynamics especially in the case of handover, because if the MN moves away from a communication region without implementing a handover, service handover would be void. This chapter looks at Y-Comm's classical approach to handover. This chapter will introduce the parameters derived from analysing the vehicular environment. Lastly, it will develop an analytical model for handover in vehicular networks.

5.2 Classification of Handover

Y-Comm has developed a very sophisticated classification of handover which is shown in Figure 5.1. Imperative handovers take place due to technical reasons alone, thereby allowing the mobile node to change its network attachment because it has been determined by technical analysis that handover must shortly occur. This could be based on parameters such as the QoS, signal strength, and coverage offered by the new network. These handovers are imperative because there could be a serious loss of connection or loss of performance if they are not executed. By contrast, alternative handovers [54] arise due to reasons other than technical issues. Hence there is no serious loss of connection or loss of performance if an alternative handover does not occur. Alternative handovers therefore may be performed due to a preference for a given network based on incentives or price. The user preferences based on promotions or features as well as contextual issues might also affect handover. Finally, there could be other network services that are being offered by other networks. This research concentrates on imperative handovers.

Imperative handovers are divided into two types. The first is known as reactive handover. This reacts to the changes in low-level wireless interfaces as to the availability or non-availability of specific networks. Reactive handovers are subdivided into unanticipated and anticipated handovers [62]. Anticipated handovers are soft handovers that illustrate the situation where there are alternative base-stations to which the mobile node may be able to handover. In unanticipated handover, the mobile is heading out of range of the current attachment and there is no other base-station to which to handover. These handovers are therefore instances of hard handovers.

The next type of imperative handover is called proactive handover. These handovers use soft handover techniques. Proactive handovers attempt to know the condition of the networks at a certain geographical location before the MN reaches that point. It allows the MN to decide when the best time to handover. Proactive handover policies allow mobiles nodes to calculate the Time Before Vertical Handover (TBVH) which will allow the mobile node to minimize packet loss, jitters and latency experienced for the MN during handovers. This, therefore, represents a mechanism that could be used to support seamless handover as it allows the system and applications more time to deal with handover issues. Currently, there are two types of proactive handovers are being developed. Firstly, is the knowledge-based which attempts to know by measuring beforehand the signal strengths of available wireless networks over a given geographical location. This might involve the act of physically driving around and taking these measurements [16]. The second proactive policy is based on a mathematical model which calculates the point when vertical handover [76] should occur and the time it takes a MN to reach that point, based on its velocity and direction.

5.2.1 Proactive Handovers in Y-Comm

As stated by the Y-Comm Framework, in order to perform proactive handover using a mathematical model approach, it is best to know the topology of local networks and how the Network Management Layer in the Core Framework manages this information. Then, the MN polls this layer to obtain information about all nearby wireless network QoS characteristics and topologies as shown in Figure 5.2. This information derived will be used by the Mobility Management Layer along with the speed and direction of the MN to determine where and when handover should occur. The Mobility Management Layer estimates the time before handover and NDT and this information is transferred to the Handover Management Layer

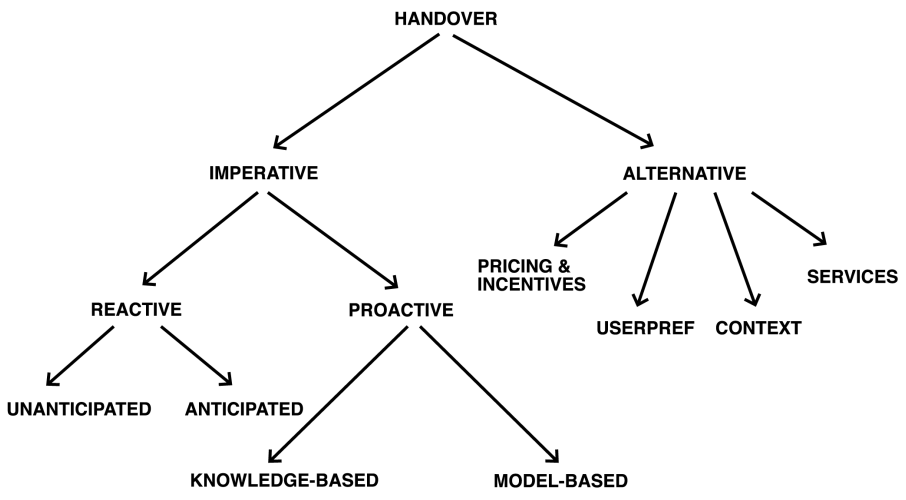


Fig. 5.1 Vertical Handover Classification

which instantly requests resources to start a handover, though resources are reserved early, handover will be started only when the time before handover elapses.

Furthermore, when the decision to execute handover is taken by the Mobility Management Layer, the new QoS, IP address, the time before handover, and estimated NDT is communicated to the upper layers. These layers are expected to take the necessary steps to avoid any slow adaptation, packet loss or latency, For example, the End System Transport Layer may signal an impending change in the QoS on current transport connections and therefore, the packets can be buffered ahead of the handover.

5.2.2 Proactive Handovers in Heterogeneous Environments

Authors in [51] presented a paper that looks at a set of mechanisms that helped to build a comprehensive framework that would support proactive handover in heterogeneous networks. They began by developing a mathematical framework and tools to guarantee its accuracy utilizing contextual information and location were examined. It demonstrated how using an ontology for proactive handover provides an implementation path for future systems and how these mechanisms can be incorporated into the Y-Comm architecture.

By using a location-based technique, it was possible to demonstrate that the NDT as well as the TBVH can be accurately estimated. These techniques are dependent on accurately estimating the handover radius, whereby the handover techniques used was mostly relevant in outdoor environments which had no irregular coverage spaces.Therefore, this

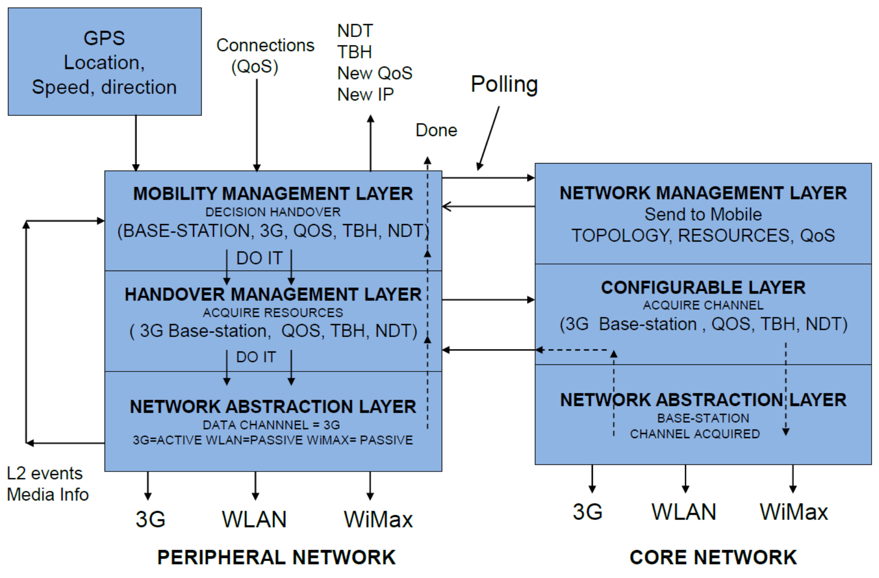


Fig. 5.2 The Proactive Handover

paper demonstrates how to calculate TBVH and NDT for any geographical location. This information is key to this research.

5.3 The use of Markov-based Model

5.3.1 Vishnu's model

The author in [88] worked on a detailed analysis of queueing models for a proactive system using Markov-based model whereby the users are able to request resources before they reached the target network. These network resource allocation issues were explored in the author's thesis. These models have generated TBVH and NDT, which were then used to great effect in the author's proactive approach. It was shown that if the parameters are acknowledged then it is possible to ensure that each user in the system will be served effectively by at least one of the available networks. This work demonstrated the need to do a vertical handover to another network, if a user is about to experience a full contention in the target network. The results showed that the proposed approach outperforms the classical resource allocation model. This helped to understand the application of proposed approach in a highly mobile real network such as VANET.

In order to look at providing QoS in resource allocation in vehicular networks, it was necessary to break down NDT to be the Resource Hold Time (RHT) and the handover time (h). The approach also looked at how much service the mobile user actually acquires. The

Markov-based model and the author's model both looked at the network performance but did not obtain the performance as seen by individual users and applications. The key issue is the lack of handover analysis in the models that were used. This issue is addressed in this chapter.

5.4 The use of Markov-based Frameworks

Markov-based models have been used to study various aspects of networking in particular network performance and performability. In [38], the author looked at performability and optimal analysis for repairmen in large-scale networks. The authors in [5] looked at a Markov-based system to analyse the Handover Process. Furthermore in [1], the authors looked at a 2-dimensional Markov Chain to analyse the performance of Multi-Path Transmission Control Protocol (MPTCP) in heterogeneous networks. Though these efforts are important, they were focused on the network level. However, in this chapter we are looking at the effect of handover on the ability to provide services to mobile users. Finally, in his PhD [73], Sardis looked at a service migration model for heterogeneous networking but did not incorporate the handover process in his model. This work addresses this issue by the introduction of a Zero-Server Markov Chain.

5.4.1 The Zero-Server Markov Chain

It is essential to model the two periods, RHT and h, to develop an analytical model for vehicular networks to investigate the QoS problems facing mobile users as previously discussed. The system probabilities can achieve a steady state through the Markov Chains which employ the concept of arrival and service rates. Nonetheless, while an ordinary Markov chain can be employed to analyse the period, another type of Markov Chain is needed to analyse the handover period owing to the lack of service during handover. Hence, a Zero-Server Markov Chain which accepts solely arrivals can be used as illustrated in Figure 5.3.

A Zero-Server Markov Chain (ZSMC) is naturally unsteady because after an extended period, the length of the queue goes to infinity. To regularise the stability, a ZSMC is combined with a Service-Based Markov Chain (SBMC) to serve customers/packets giving room to exit the system.

In order to fully understand this technique, we will look at a cyclic queuing system in which the server is serving a distributed number of queues as shown in Figure 5.4. When the

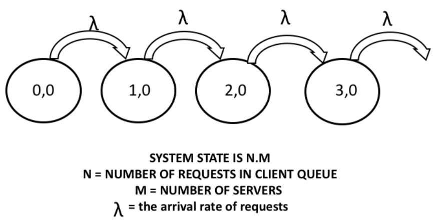

λ
λ
λ
λ
0,0
1,0
2,0
3,0
SYSTEM STATE IS N.M
N = NUMBER OF REQUESTS IN CLIENT QUEUE
M = NUMBER OF SERVERS
λ = the arrival rate of requests

Fig. 5.3 A Zero-Server Markov Chain

server arrives at the queue, it serves exhaustively until there is no one left in the queue before it moves to the other queues.

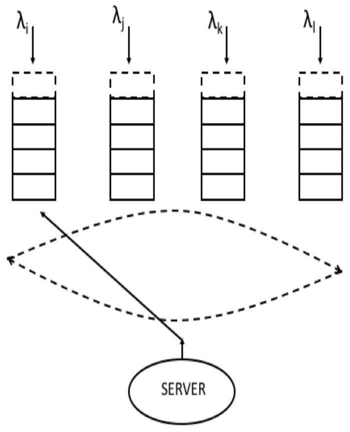


Fig. 5.4 A Cyclic Service System

As such, it is paramount to leave the ZSMC with an exit velocity μ_v. For instance, explored in [55] and illustrated in Figure 5.5 is an example of the model of exhaustive cyclic service.

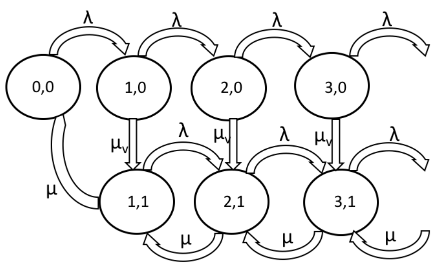


Fig. 5.5 Markov Chain showing Exhaustive Cyclic Service

The Markov Chain 1, denoted by $K,1$, shows what happens when the server is at the queue. The server employs an exhaustive service discipline and hence only leaves the queue when the queue is empty. This is shown by the transition from $P(1,1)$ to $P(0,0)$. The system returns to Markov Chain 0, denoted by $K,0$, which represents when the server is not at the queue. When the server arrives at the queue again there is a transition back to Markov Chain 1 with rate μ_v. This represents the vacation rate which is the inverse of the vacation time T_v, the period when the server is not at the queue. Using this analysis it was possible to show that the average number of people in the system, which in the original work is denoted as L_{EH}, is given as:

$$L_{EH} = \lambda T_v + \frac{\rho}{1-\rho} \tag{5.1}$$

This result clearly agrees with normal queuing analysis because when the server is always at the queue, T_v is zero, and hence the average number of the people is given as:

$$L_{EH} = \frac{\rho}{1-\rho} \tag{5.2}$$

This is the same formula for the average number of outstanding requests in M/M/1 analysis [80] and this shows that the Zero Markov approach yields the correct results for cyclic systems as well as normal M/M/1 models. Furthermore, when compared to a discrete simulation of a real system using Exhaustive Cyclic Service, the results from this approach were very accurate for operational loads [55]. The difference between the theoretical results and the simulation at higher utilisations was due to the fact that the estimation of T_v in cyclic systems with exhaustive service gets difficult for very high loads. These results show that this technique can also be used to look at the effect of handover on service delivery in Vehicular Edge Clouds. However, what is needed is an SBMC that can be used to represent how service is delivered in vehicular networks.

5.5 Analysis of Vehicular Environment

With wireless networking, the network coverage area is a region with an irregular shape where signals from a given Point of Attachment (PoA) i.e., Access Point or Base Station can be detected by an MN. The signals from the PoA are unreliable at the boundary and beyond the coverage area signals from the PoA cannot be detected. For seamless communication, the handover should be finished before the coverage boundary is reached.The list of terms used is shown in Table 5.1.

Table 5.1 Understanding Symbols used in Vehicular Networks

Symbol	Description	Meaning
R_H	Handover Radius	When handover should be completed
R_E	Exit Radius	When the handover process should start
T_{EH}	Handover Time	Handover starts at R_E and is done by R_H
v	Velocity	Average velocity of the mobile node

Therefore, two circles were presented by the handover radius (R_H) and exit radius (R_E) were defined in [52] to ensure a smooth handover. The work states that the handover must begin at the exit radius and should be completed before reaching the handover radius boundary as shown in Figure 5.6.

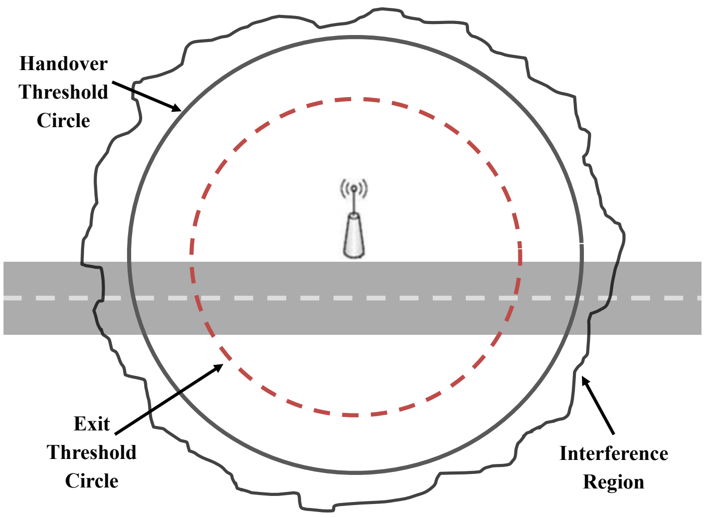


Fig. 5.6 Network Coverage

The exit radius will therefore be dependent on the velocity, v, of the MN. If we represent the time taken to execute a handover by T_{EH}, then:

$$T_{EH} \leq \frac{(R_H - R_E)}{v} \tag{5.3}$$

Hence, the exit radius can be given as shown in Equation (5.4)

$$R_E \leq R_H - (v * T_{EH}) \tag{5.4}$$

So, the faster an MN moves the smaller the R_E at which handover must begin. Given that we know the time is taken to execute a handover, the velocity of the MN and handover radius, then we can calculate the exit radius which is dependent on the handover radius. A good estimation of the handover radius is required for the proposed approach which is dependent on the propagation models being used. The time taken to effect a handover was shown to be dependent on various factors such as Detection Time (t_{det}), Configuration Time (t_{con}), Registration Time (t_{reg}) and Adaptation Time (t_{adp}) as discussed in [52].

Since reactive handovers respond to network conditions all four times must be added together because the MN knows nothing beforehand about the characteristics of the various networks. For proactive handover techniques, there is no detection time since the MN would know where all the local networks are located. This is particularly valid for vehicular networks as the route is fixed and therefore, the location of the next (target) network is likely to be known. Configuration time is also negligible since the MN will know the IP address of the target network. Registration Time is still valid. In addition, for proactive networks, the need for the transport protocol to adapt can be signalled before or during handover and not after the handovers occur. Therefore, it means that the adaptation time can be done in parallel with the registration time. So, for proactive handover, the time for handover is given by:

$$T_{EH} = MAX(t_{reg}, t_{adp}) \tag{5.5}$$

The previous work on proactive handover in [50] showed that the above-mentioned coverage parameters can be segmented into communication ranges and presented an in-depth analysis of such segmentation and their importance in order to achieve a seamless handover in vehicular networks as shown in Figure 5.7. This segmentation can be put into effective use for achieving proactive handover, resource allocation, and service migration in highly mobile environments.

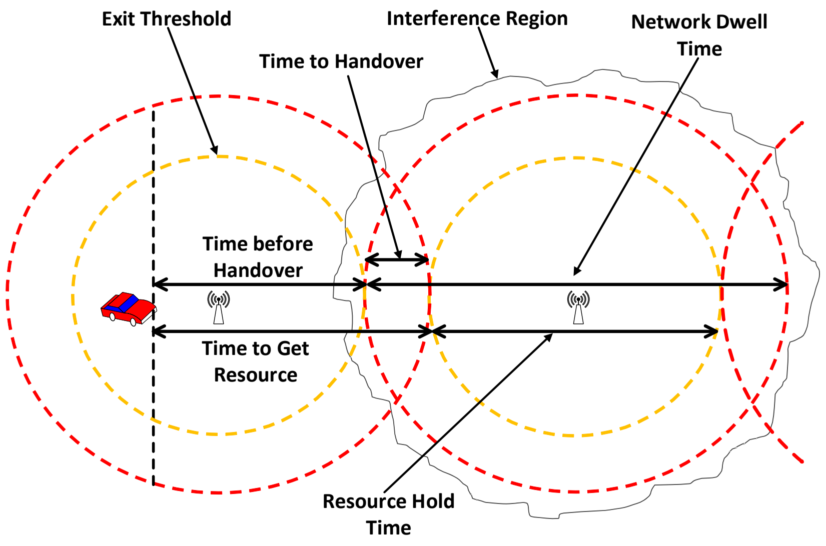


Fig. 5.7 Communication Range Segmentation

Figure 5.7 shows a more advanced scenario in which three consecutive overlapping wireless networks are segmented based on various key time variables which can be used to enhance handover and resource allocation. An example of such a scenario has been developed in London using the Middlesex (MDX) VANET testbed where three RSUs were deployed in an overlapping fashion on the A41 road Middlesex University to study and test vehicular communication.

5.6 Developing an Analytical Model for Vehicular Networks

In order to develop an analytical model, we use a simplified diagram of a vehicular network as shown in Figure 5.8. This consists ot three adjacent RSUs with overlapping coverage in order to provide seamless communication. The coverage area is therefore modelled using two parameters, N, the RHT and the handover time, h. RHT is the amount of time in the coverage area of the RSU when the vehicle has access to the communication channel. However, as the vehicle approaches the next RSU, it begins to handover to gain access to communication channel of the next RSU and so releases the channel of the current RSU. Therefore, services can only be delivered during the RHT and no services can be accessed during the handover period.

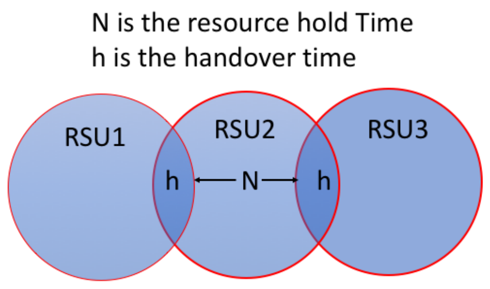


Fig. 5.8 Analytical Model

5.6.1 The Analysis

The model for the vehicular network is shown in Figure 5.9. The arrival rate of requests by the application is given by λ while the service rate is given by μ. The ZSMC is given as Chain 0 and is represented by states $P_{n,0}$. The Markov Service Chain, denoted Chain 1, is

Table 5.2 Understanding the Symbols for the Analysis

Symbol	Description	Meaning
λ	Arrival rate of requests	The rate at which requests are made
μ	Service Rate	The rate at which requests are served
$P_{n,0}$	State probability in Chain 0	System in ZSMC
$P_{n,1}$	State probability in Chain 1	System in SBMC
v_1	Transition rate from Chain 1 to Chain 0	Moving from SBMC to ZSMC
v_2	Transition rate from Chain 0 to Chain 1	Moving from ZSMC to SBMC

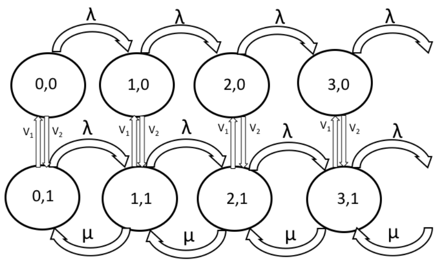


Fig. 5.9 Markov Chain for Vehicular Networks

given by $P_{n,1}$. The transitions between the two Markov Chains are represented by rates v_1 and v_2 respectively. A list of terms used in this analysis is given in Table 5.2.

Looking at State (0,0) our first equations is:

$$(\lambda + v_2)P_{0,0} = v_1 P_{0,1} \tag{5.6}$$

Hence:

$$P_{0,1} = ((\lambda + v_2)/v_1)P_{0,0} \tag{5.7}$$

Looking at State (0,1),

$$v_2 P_{0,0} + \mu_1 P_{1,1} = (\lambda + v_1)P_{0,1} \tag{5.8}$$

Substituting for $P_{0,1}$ and simplifying:

$$P_{1,1} = (\lambda/\mu)(P_{0,1} + P_{0,0}) \tag{5.9}$$

Looking at State (1,0)

$$\lambda P_{0,0} + v_1 P_{1,1} = (\lambda + v_2) P_{1,0} \tag{5.10}$$

Substituting for $P_{1,1}$ and simplifying:

$$P_{1,0} = (\lambda/(\mu(\lambda + v_2)))((\mu + v_1)P_{0,0} + v_1 P_{0,1}) \tag{5.11}$$

Looking at State (1,1)

$$\lambda P_{0,1} + v_2 P_{1.0} + \mu P_{2,1} = (\lambda + v_1 + \mu) P_{1,1} \tag{5.12}$$

Substituting for $P_{1,1}$ and simplifying:

$$P_{2,1} = (\lambda/\mu)(P_{1,1} + P + 1, 0) \tag{5.13}$$

Looking at State (2,0)

$$\lambda P_{1,0} + v_1 P_{2,1} = (\lambda + v_2) P_{2,0} \tag{5.14}$$

Again by substitution and rearranging we get:

$$P_{2,0} = (\lambda/\mu(\lambda + v_2))((\mu + v_1)P_{1,0} + v_1 P_{1,1}) \tag{5.15}$$

Therefore based on these results the general equations of the model are given by:

$$P_{n,0} = (\lambda/(\mu(\lambda + v_2)))((\mu + v_1)P_{n-1,0} + v_1 P_{n-1,1}) \tag{5.16}$$

$$P_{n,1} = (\lambda/\mu)(P_{n-1,1} + P_{n-1,0}) \tag{5.17}$$

5.6.2 Handling a system with capacity K

For a system with a limited capacity, K, Equation 5.17 is valid, but for $P_{K,0}$, Equation 5.16 must be modified as there is no forward rate (i.e. $\lambda P_{K,0}$). Hence, we get:

$$P_{K,0} = (\lambda/(\mu v_2))((\mu + v_1)P_{K-1,0} + v_1 P_{K-1,1}) \tag{5.18}$$

5.6.3 Parameters from this model

From this analytical model, it is possible to calculate the average number of requests. According to basic Queuing Theory [40], this is given by:

$$N_{avg} = \sum_{n=1}^{\infty} nP_n \tag{5.19}$$

for a finite system,

$$N_{avg} = \sum_{n=1}^{K} nP_n \tag{5.20}$$

However, in this model we must consider both Markov chains, hence

$$N_{avg} = \sum_{n=1}^{\infty} nP_{n,0} + \sum_{n=1}^{\infty} nP_{n,1} \tag{5.21}$$

For a system with limited capacity, K; this is give as:

$$N_{avg} = \sum_{n=1}^{K} nP_{n,0} + \sum_{n=1}^{K} nP_{n,1} \tag{5.22}$$

Hence we can use the above equation to calculate N_{avg}. Thus, by using Little's Law, the response time of the system, given by T, can be calculated as shown by the equation for Little's law in Equation 5.23.

$$N_{avg} = \lambda T \tag{5.23}$$

For a system with limited capacity, K, the blocking probability P_B is given by:

$$P_B = P_{K,0} + P_{K,1} \tag{5.24}$$

5.6.4 New Parameter: Probability of lost service

Because we can now calculate the steady state probabilities even when the server is not at the queue, in the context of vehicular networks, it is possible to calculate the probability of lost service due to handover. Lost service is defined as the situation when there are customers/packets in the queue waiting to be served but the server is not at the queue. Hence this given by:

$$P_{lostservice} = \sum_{n=1}^{\infty} P_{n,0} \tag{5.25}$$

or for a system with limited capacity, K,

$$P_{lostservice} = \sum_{n=1}^{K} P_{n,0} \tag{5.26}$$

5.6.5 The Middlesex Testbed Summary

In order to understand how the analysis could be applied to a real system, the VANET testbed deployed at Middlesex University was used. The experimental setup is shown in chapter 3 in Figure 3.5. It consists of 7 RSUs with three RSUs: 5, 6 and 7, located along the A41 which runs behind the Hendon Campus.

The Middlesex testbed uses IEEE 802.11p (DSRC WAVE) American Standard which enables WSMP messages to be sent between the RSU and the OBU. This includes Basic Safety Messages (BSMs) that contain the GPS co-ordinates, the speed and direction of the vehicles. These messages are sent by the OBUs in the vehicles to the RSUs and are then forwarded by the RSUs to a VANET Server located on the Hendon Campus at Middlesex University. BSMs are transmitted at a rate of 10 Hz and so these messages enable vehicles to be tracked very accurately. The testbed uses a hybrid backhaul scheme in which the RSUs on the Hendon Campus are backhauled using the University's Gigabit Ethernet Network. This is shown in Figure 5.10. An LTE backhaul system was used to backhaul the data from the RSUs on the A41, to the same VANET server.

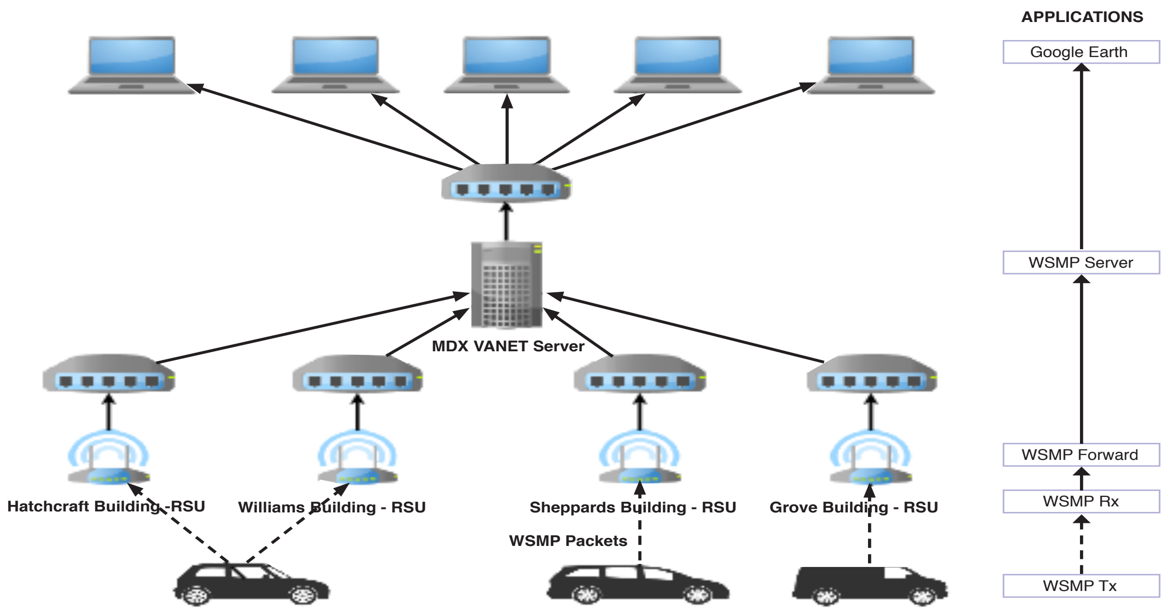


Fig. 5.10 Backhaul for Hendon Campus RSUs

Hence using this setup, it is possible to accurately measure the Network Dwell Distance (*NDD*) or coverage area of each RSU. We then divide *NDD* by the speed of the vehicle, v, to get the effective time that the mobile node will be in the coverage of the RSU which is equal to $N+h$ where N is the RHT and h is the handover time.

$$N+h = NDD/v \tag{5.27}$$

$$N = NDD/v - h \tag{5.28}$$

According to a detailed analysis of vehicular networks in [15], [51], h is set to 4 seconds. Thus we can determine N, the RHT for each RSU. Two speeds are considered, 30 mph and 50 mph. The results are shown in Table 6.1.

5.6.6 Video Service Scenario

To explore the application of these formulas, a video service scenario is examined. The video server serves videos to mobile users by the aid of a PULL mechanism for which the application on the mobile device receives 25 frames per second from the server. The server

Table 5.3 Communication Coverage Segmentation Distance and Time ($h = 4s$).

RSU No.	NDD	30 Mph	50 Mph
		N	N
RSU 1	300 m	18.37 s	9.42 s
RSU 2	456 m	30.00 s	16.40 s
RSU 3	517 m	34.55 s	19.13 s
RSU 4	248 m	14.49 s	7.09 s
RSU 5	974 m	68.63 s	39.57 s
RSU 6	1390 m	99.64 s	58.19 s
RSU 7	1140 m	81.00 s	47.00 s

utilises 5G network slicing and is therefore able to serve video frames at 50 frames per second, in this set-up, $\lambda = 25$ and $\mu = 50$. Also, $v_1 = 1/N$ where N is the RHT for the diverse RSUs as illustrated in Table 6.1, finally $v_2 = 1/h$ where h is the handover time. In line with complete analysis of vehicular networks in [15], [51], h is put at 4 seconds so $v_2 = 0.25$.

The Parameters to be Evaluated

It is commonly known that in mobile and vehicular systems, the resources on the mobile node are less than the resources on main servers and therefore there is a need to ensure that resources, such as memory, are efficiently used on mobile devices. Hence, it is important to see how well the system works when different numbers of video frame buffers are provided for the video connection. Thus, the results are shown for different values of K, the capacity of the system. This algorithm will allow us to find out the average number of outstanding requests in the system and the average response time for each service request for different values of K. In addition, it also will enable us to obtain the probability of blocking, P_B as well as the probability of lost service given by P_{LS}. These results were obtained for each RSU in the Middlesex VANET testbed.

5.6.7 Generation of Results

A C program was written to generate the results for this model. However it is necessary to check that the C program itself is accurate; so it was decided to do some calculations manually and compare them with the results of the program. We therefore used the data for RSU2 to generate results for P_{LS}. As shown in Table 5.4, the difference is extremely small which indicates that the program used to generate the results is accurate.

Table 5.4 Accuracy of Results for P_{LS} for RSU2 at 30 mph

K	**Manual**	**Program Results**	**Difference %**
1	0.11687135	0.1168698211	0.0013082
2	0.11698219	0.11698219	0.0008774
3	0.117027060883	0.1170260340	0.0008775

5.6.8 Results from Standard Analysis

In an attempt to fully comprehend the work, results will be presented from a standard queueing analysis by looking at a scenario with the absence of handover within a fixed network, which allows the server to consistently be at the queue. By employing a simple M/M/1 analysis with infinite buffering, the equation is:

$$N_{avg} = \rho/(1-\rho) \tag{5.29}$$

$$T_{resp} = 1/(\mu - \lambda) \tag{5.30}$$

where $\rho = \lambda/\mu$; hence $N_{avg} = 0.5/0.5 = 1$
and $T_{resp} = 1/(50-25) = 0.04$ seconds

Limited Capacity

A system with limited capacity, K will be looked at here.

For these systems:

$$N_{avg} = \rho/(1-\rho) - ((K+1)\rho^{K+1}/(1-\rho^{K+1}) \tag{5.31}$$

Response times will be found using Little's Law in Equation 5.23. The results for the same scenario will be presented here with different values of K. The Average Number of Requests in the system will be presented as N_{avg} and the corresponding Response Time, T_{resp} with the probability of Blocking P_B. Table 5.5 illustrates the results, for which, a queueing system with $K = 25$ practically performs similar to a M/M/1 system.

Table 5.5 No Handover with Limited Capacity

K	N_{avg}	T_resp	P_B
5	0.905	0.0362	0.0159
10	0.995	0.0398	0.000489
15	1.000	0.040	0.0000153
20	1.000	0.040	0.000005
25	1.000	0.040	0.000000
30	1.000	0.040	0.000000

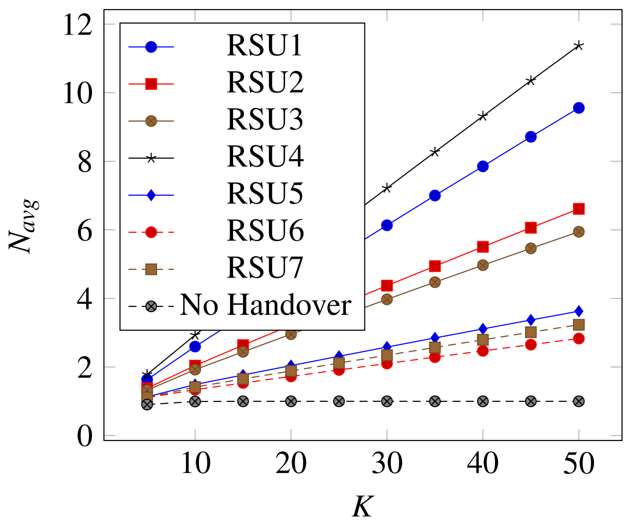


Fig. 5.11 N_{avg} vs K at 30 mph

Results from Testbed

Here, the results from the testbed will be used to estimate the parameters such as the average number of requests, the Response time, the Probability of Blocking P_B and the Probability of Lost Service, P_{LS} for each RSU for systems capacities from $K = 5$ to $K = 50$.

Figure 5.11 shows the N_{avg} for each RSU. Basically the more time, the vehicle can spend in the Markov Chain 1, the service chain, the more requests will be served and hence the smaller the value of N_{avg}. Since we said that the rate out of the service chain is the inverse of N, the RHT, the greater the value of N, the smaller the value of v_1 and hence the greater the level of service. Thus since the RSUs on the A41, RSU5, RSU6, RSU7 have the largest values of N, they have the lowest values of N_{avg} In contrast, RSU4 and RSU1 have the smallest values of N and so have largest values of N_{avg}. The same reasoning explains Figure 5.12 which shows the Blocking Probability P_B as K increases. Both graphs, however, also show

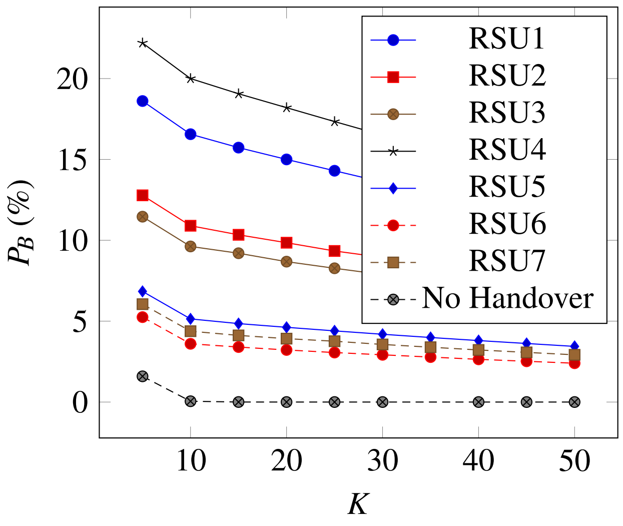


Fig. 5.12 P_B vs K at 30 mph

that compared to the no handover case, handovers to the different RSUs significantly affect the performance of the system due to services becoming unavailable during handovers. The ability to quantify this lost service is therefore very important.

Figure 5.13 shows the values of the probability of lost service denoted by P_{LS} for each RSU. The figure appears to show that for each RSU, P_{LS} is the same and is thus independent of the value of K. This is a very interesting result.

To explore this further, we now calculate the values of P_{LS} for 2 RSUs: the most effective RSU, RSU6, and the least effective RSU, RSU4.

Table 5.6 shows how P_{LS} varies with the system capacity K for RSU4 and RSU6 for ρ at 0.5 and speed at 30 mph. The difference between the least and maximum values of each RSU is quite small. For RSU4, that difference is 0.596% and for RSU6, it is 0.007%.

Table 5.7 shows how P_{LS} varies with utilisation ρ for RSU4 and RSU6 for $K = 100$ and speed at 30 mph. Again, the difference between the least and maximum values of each RSU is quite small. For RSU4 , that difference is 0.8984% and for RSU6, it is 0.1646%.

These results show that P_{LS} only marginally changes when there are changes in system capacity K and utilisation ρ. This also means that if we are able to calculate P_{LS} for the simplest setup then we know that the maximum values will be very close to the results for the simple values which is a significant contribution..

Figure 5.14 compares P_{LS} with different values of K for both 30 mph and 50 mph. The results show that there are higher values of P_{LS} for 50 mph compared to values at 30 mph both for RSU4 and RSU6. As shown in Table 6.1, for the same speeds, the Resource Hold

Table 5.6 Detailed Look at P_{LS} in % and System Capacity K

K	**RSU4**	**RSU6**
1	21.4906	3.8340
2	21.5112	3.8376
3	21.5195	3.8391
4	21.5234	3.8397
5	21.5254	3.8401
10	21.5284	3.8404
20	21.5309	3.8405
30	21.5331	3.8405
50	21.5369	3.8406
100	21.5435	3.8408
200	21.5502	3.8410

Table 5.7 Detailed Look at P_{LS} in % and Utilisation ρ

ρ	**RSU4**	**RSU6**
0.1	20.7343	3.6948
0.2	21.2375	3.7849
0.3	21.4081	3.8158
0.4	21.4930	3.8314
0.5	21.5435	3.8408
0.6	21.5767	3.8471
0.7	21.6000	3.8516
0.8	21.6168	3.8550
0.9	21.6283	3.8576
0.94	21.6312	3.8584
0.98	21.6327	3.8591

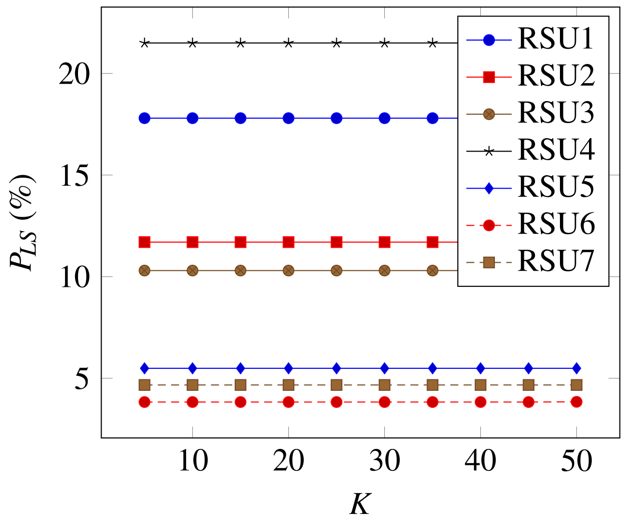


Fig. 5.13 P_{LS} vs K at 30 mph

time of RSU4 is much less that RSU6, so RSU4 will leave the Service Chain much quicker than RSU6 as so experience a greater loss of service.

For different speeds, the rate out of the service chain, Markov Chain 1, increases for the 50 mph scenario because the RHT for 50 mph is less than the RHT for 30 mph for both RSUs. Hence the transition rates out of the Service Chain is much greater at 50 mph than at 30 mph for both RSUs, thus P_{LS} values are greater for 50 mph compared to 30 mph. This difference in the rates at the different speeds are much larger for RSU4 ($1/7.09 - 1/14.49 = 0.07411$) compared with the differences for RSU6 ($1/58.19 - 1/99.64 = 0.00715$) hence there will be a greater difference between the values for the different speeds for RSU4 compared to RSU6 as shown in Figure 5.14.

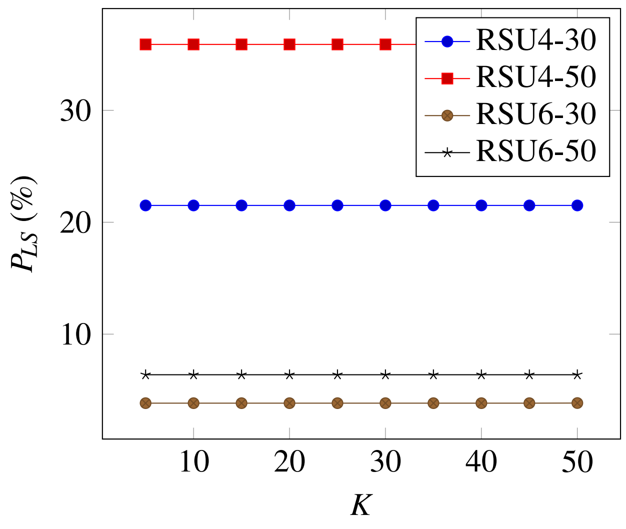


Fig. 5.14 P_{LS} vs K at 30 mph and 50 mph

5.7 Chapter Summary

This chapter presented the Y-Comm classical approach and showed the analysis of the effect of handover on service delivery in a vehicle network. This was done using the ZSMC technique. The analytical model allowed a new parameter call the probability of lost service, P_{LS}, to be obtained. The analytical model was developed and then applied to the MDX VANET testbed. An investigation of the results showed that P_{LS} was marginally dependent on the system capacity (K) and utilisation ρ. P_{LS} appears to be only significantly dependent on the rates of transition between the ZSMC and the SBMC. The method of approach used in this chapter will also be used to develop our model for service migration in the next chapter.

Chapter 6

Analytical Model for Service Migration in Vehicular Network

6.1 Brief Introduction

The chapter begins by deriving the equations for reactive and proactive service migration. It then looks at how service migration mechanisms that support service migration can be analysed in terms of reactive and proactive service migration in a similar way that the reactive and proactive handover was analysed in the previous chapter. Finally, the ZSMC technique is applied to analyse proactive and reactive service migration using different mechanisms in vehicular networks.

6.2 Reactive Service Migration

The analytical model for reactive service migration is shown in Figure 6.1. This figure shows that there are two key issues, the communications parameters and the service migration parameters. The communication parameters are given by the N and h. We now need to consider the service migration model. We first define the service migration time (SMT) as the time it takes to migrate the service from one RSU to another RSU. The service is not available during SMT. We also define the Service Hold Time (SHT) as the period in the RSU that the service is available to mobile users.

However, we need to link these parameters. This is given by:

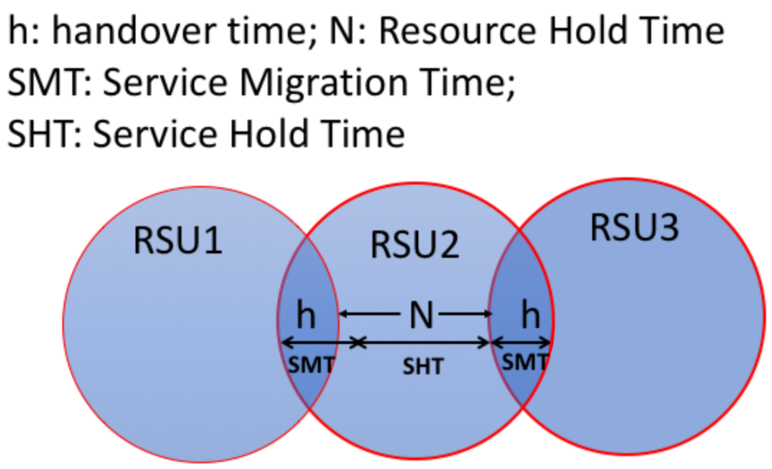

h: handover time; N: Resource Hold Time
SMT: Service Migration Time;
SHT: Service Hold Time
RSU1
RSU2
RSU3
h
N
h
SMT
SHT
SMT

Fig. 6.1 Analytical Model to explore Reactive Service Migration

$$SHT + SMT \leq N + h \tag{6.1}$$

$$SHT \leq (N + h) - SMT \tag{6.2}$$

$$SHT \geq 0 \tag{6.3}$$

Hence if we know SMT, N and h then we can calculate SHT. In addition, $SHT \geq 0$. This is because if this is not case, then service migration to the next network cannot occur because by the time the migration is completed the MN would have already left the upcoming network. This is the key equation for reactive service migration in highly mobile environments such as vehicular networks.

6.3 Proactive Service Migration

In addition to reactive service migration where service migration begins when at the same time that handover occurs, we will consider starting handover X seconds before reactive service migration should begin. This is called proactive service migration. This is shown in Figure 6.2.

In this diagram X is the proactive time; hence, the service begins to migrate X seconds before handover occurs. It is assumed therefore that the Mobile Node still has access to the service for the X seconds.

This therefore means that the breakof service due to service migration goes from SMT to $SMT - X$ hence our equations now become:

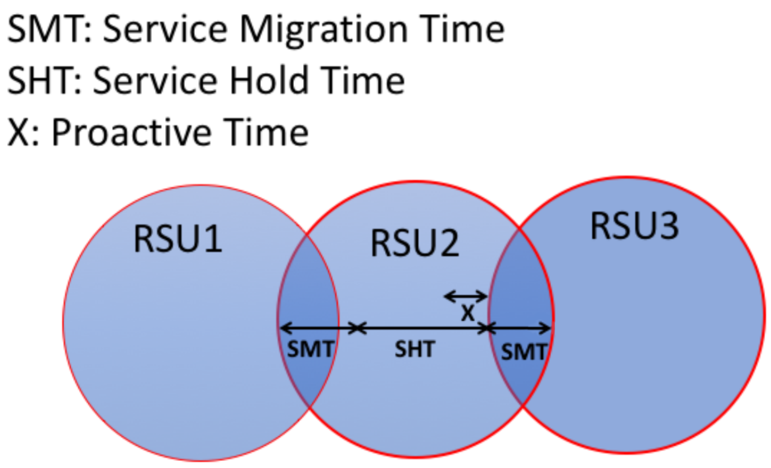


Fig. 6.2 Proactive Service Migration

$$SHT + (SMT - X) \leq N + h \tag{6.4}$$
$$SHT \leq (N + h) - (SMT - X) \tag{6.5}$$
$$SHT \geq 0 \tag{6.6}$$
$$X_{min} = N + h - SMT \tag{6.7}$$
$$X_{min} \geq 0 \tag{6.8}$$

The last two equations are obtained because the minimum value of SHT is zero. Hence we can work out what is the minimum value of X.

We can also work out the maximum value of X using the following equations:

$$X \leq N \tag{6.9}$$
$$(SMT - X) \geq h \tag{6.10}$$

The first equation takes into account that in order to begin a service migration proactively, the mobile node should have access to the channel in the current network. So, we are saying that X should be less than or equal to the RHT in the current network. However, the second equation tells us that because handover will come into play which will affect service migration so having values of X such that $(SMT - X) < h$ would not be of benefit. Hence we can say that the maximum value of X is the minimum of these two values.

$$X_{max} = min(X \leq N, (SMT - X) \geq h) \tag{6.11}$$

Hence we can now summarise all the equations about proactive migration time X, which are given below:

$$X_{min} \geq 0 \tag{6.12}$$

$$X_{min} = N + h - SMT \tag{6.13}$$

$$X_{max} = min(X \leq N, (SMT - X \geq h)) \tag{6.14}$$

$$X_{max} \geq Xmin \tag{6.15}$$

6.4 Service Migration as a Mechanism

Under the Y-Comm architecture, due to the mobility of the user, the heterogeneity of the edge device while considering dynamics of the network resources such as the computing resources, QoS, security, and the network bandwidth, we should offer the service in accordance with the personalized demands of the user. The computational power is particularly large, so the computing resources are required to be more flexible when deploying services on the edge. Therefore, different migration mechanisms such as KVM, LXD, Docker, and Unikernels will be explored for a better understanding of service migration.

The next section starts by showing the preliminary results gotten at the early stage of this research from a mobile node moving at two different velocities i.e., 30 Mph and 50 Mph going through RSUs within a VANET environment using different applications as a service.

6.5 Application-Focused Service Migration

Table 6.1 shows the NDT, denoted by $\aleph$, and $\mathbb{N}$ for two different velocities i.e., 30 Mph and 50 Mph for all the RSUs. We know from [52] that the handover execution time i.e., $\hbar$ is 4s and therefore, the $\mathbb{N}$ is:

$$\mathbb{N} = \aleph - \hbar \tag{6.16}$$

Table 6.1 Communication Coverage Segmentation Distance and Time ($\hbar = 4s$).

RSU No.	**NDD**	**30 Mph**		**50 Mph**	
		$\aleph$	$\mathbb{N}$	$\aleph$	$\mathbb{N}$
RSU 1	300 m	22.37 s	18.37	13.42 s	9.42 s
RSU 2	456 m	34.00 s	30.00 s	20.40 s	16.40 s
RSU 3	517 m	38.55 s	34.55 s	23.13 s	19.13 s
RSU 4	248 m	18.49 s	14.49 s	11.09 s	7.09 s
RSU 5	974 m	72.63 s	68.63 s	43.57 s	39.57 s
RSU 6	1390 m	103.64 s	99.64 s	62.19 s	58.19 s
RSU 7	1140 m	85.00 s	81.00 s	51.00 s	47.00 s

6.5.1 Live Service Migration Use Case Scenario Results

We are using the Middlesex VANET tested as already discussed in Chapter 5. The authors in [48] detail a layered framework for migrating active service applications which are condensed in virtual machines and containers. Containers are developing technology and they consume less storage space compared to VMs, therefore, will be appropriate for service migration. In order to reduce service downtime and overall migration time, they categorised the instances in layers. The first is a base layer which includes the guest OS, kernel, and so on, but with no applications installed. Secondly, the application layer which contains different services such as the game server, face detection and so on. And lastly, the third layer that contains instances. For this experiment a 2-layer approach refers to the use of the base layer and the instance layer. While a 3-layer approach makes use of all three layers. Under the given framework, the migration performance in terms of SMT for both VM and Containers were examined in a controlled test-bed environment. They have considered five different applications for migration; No Application, Game Server, RAM Simulation, Video Streaming, and Face Detection. The results on the performance of two i.e., 2-layer and 3-layer approaches for different application with container technology as shown in Table 6.2 have been used in this work for the evaluation of our model.

Given the measurement results for service migration as listed in Table 6.2, we consider two use-cases for two different velocity i.e., 30Mph and 50Mph:

- Reactive Approach where the service migration starts once the MN reaches the next RSUs coverage region.
- Proactive Approach where the service migration starts before the MN reaches the next RSUs coverage region at point X called as the proactive service migration time.

Table 6.2 Migration results for two-layer and three layer configurations

Services	**SMT / Data Transferred**	
	2 Layer	**3 Layer**
No Application	6.5 s /1.4 MB	11.0 s/ 1.9 MB
Game Server	7.3 s /2.2 MB	10.9 s/ 2.7 MB
RAM Simulation	20.2 s /97.1 MB	27.2 s/ 97.6 MB
Video Streaming	27.5 s /180.2 MB	37.3 s/ 184.6 MB
Face Detection	52.0 s /363.1 MB	70.1 s/ 365.0 MB

The graphs presented in Figure. 6.3 and 6.4 show the ratio for reactive and proactive service migration derived from Equation 6.3 and 6.15 respectively for different application services presented in Table 6.2.

We can look at the equation 6.1 as follows:

$$N + h \geq SHT + SMT \tag{6.17}$$

Since SHT must be greater than 0, it means that the ratio:

Hence if SHT is zero:

$$N + h \geq SMT \tag{6.18}$$

Thus, for reactive handover, this represents the maximum value of SMT. If SMT is greater than this value the service migration cannot occur because by the time the migration occurs the mobile user would have left the targeted network. We can also express this as a fraction by dividing both sides by SMT.

For reactive service migration where we start the service migration at the same time as the communication handover,

$$(N + h)/SMT > 1 \tag{6.19}$$

For a service to be successfully migrated using reactive service migration, the ratio must be above 1.

For proactive service migration, where service migration starts X seconds before the communication handover

$$(N+h)/(SMT-X) > 1 \tag{6.20}$$

In case 1, we can observe that the service migration in a mobile environment is successful for both No application, and Game server with 2-layer and 3-layer approach for both 30Mph and 50Mph as shown in Figure 6.3. This is due to the fact that size of the service are small compared to others. RAM Simulation service cannot be successfully migrated for 3-layer approach at 50Mph, this is due to the high speed and rest of the cases for RAM Simulation service can be successfully migrated. The other two services i.e, Video streaming and Face detection cannot be migrated using reactive service migration.

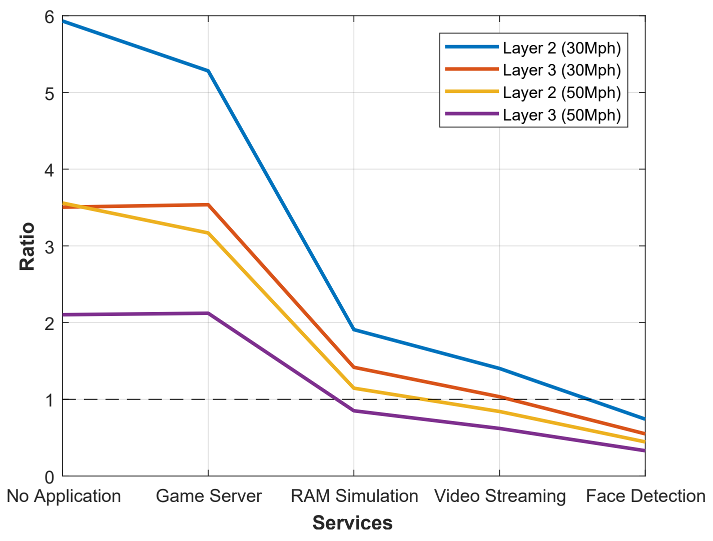


Fig. 6.3 Service Migration for Reactive Handover

In case 2, Figure 6.4 shows the results of proactive approach i.e, service is migrated at the proactive service migration time, X. This is not a constant value and it changes according to the type of the service. We consider the values of X (in seconds) for No Application and Game server as 5s, RAM simulation and video streaming to be 15s and then finally face detection as 45s. The results show that the No Application, Game server, and RAM Simulation are successfully migrated for all cases. Video streaming, and Face detection are almost nearing the threshold 1 for 3-layer approach at 50Mph and all other cases can be successfully migrated. This shows the need for a proactive service migration approach in a highly mobile edge environment. In addition, proactive service migration time, X has to be explored in the future to develop efficient algorithms for such service migration. Here, the SMT was measured in a controlled environment and only one mechanism was considered.

However, in order to have intelligent service migration we need to get accurate values of SMT for different mechanisms.

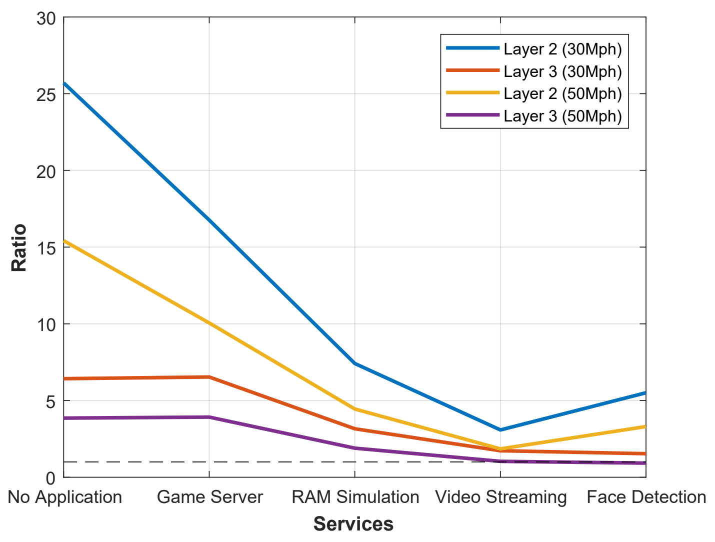


Fig. 6.4 Service Migration for Proactive Handover

6.6 Developing a Prototype Environment for the New Framework

6.6.1 NMS and FUSE as a Service

Using this approach, we will now consider a Network Memory Server (NMS) that manages blocks of memory on behalf of its clients. The NMS creates, reads, writes and deletes blocks of memory using a simple socket interface. We have incorporated the NMS as a back-end to the FUSE file system which is a user-space file system commonly employed in Linux environment as shown in Figure 6.5. The diagram in Figure 6.6 depicts how the NMS and FUSE services are used in a Vehicular Network. FUSE runs on the MN and communicates with the NMS server running in the local network. As the MN moves within the vehicular network the NMS is migrated to a nearby RSU.

6.7 Investigating Different Migration Mechanisms for NMS

It is necessary to look at how the NMS will be migrated using different migration techniques. Four state-of-the-art migration techniques, such as KVM, LXD, Docker and Unikernels were

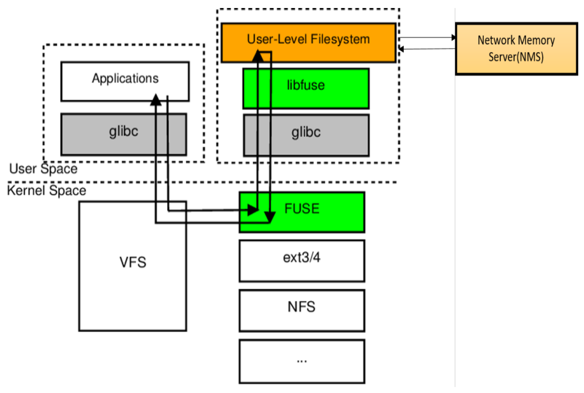


Fig. 6.5 FUSE and NMS integration

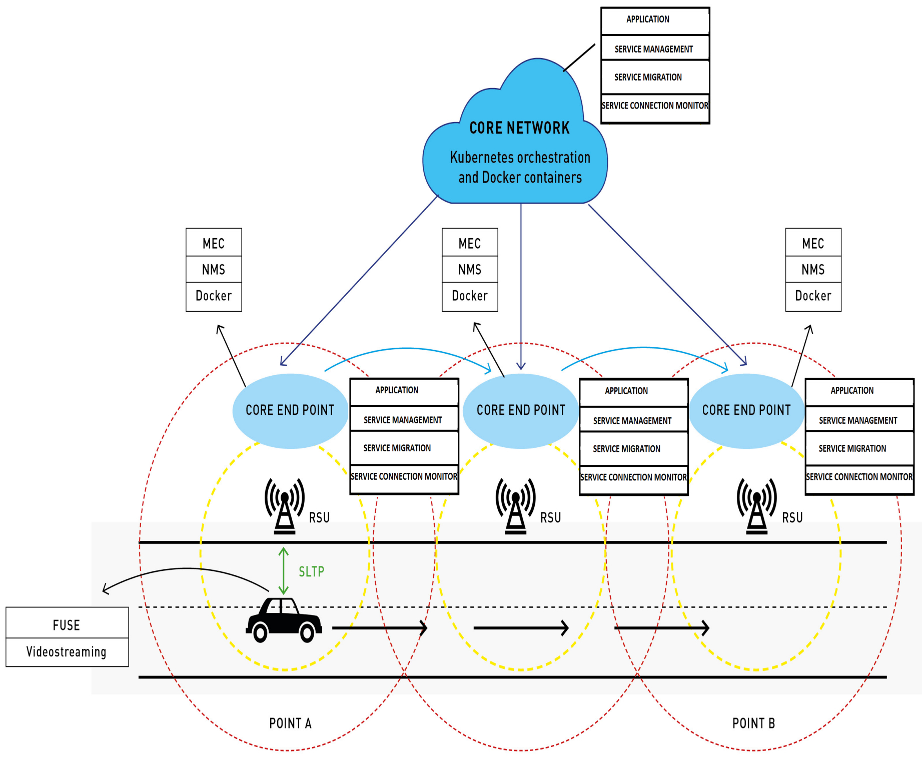


Fig. 6.6 NMS migration scenario within VANET

deployed in order to test their performance. A logical diagram of the network implementation is shown in Figure 6.7. Two physical interfaces and one virtual bridge are attached to the host computer: wireless adapter (wlp0s1), Ethernet adapter (enp0s1), and one virtual bridge is created as part of the VM virtual network (br0). Two virtual Ethernet adapters are created per VM to represent the connections from the VM to the host computer in the form of *veth* interfaces. Two interfaces are required, since one of them will be used for service traffic

Table 6.3 Linux host used for live migration without shared storage

Linux-host (HWs)	Features
Device	Dell Inspiron 5559
CPU	Intel Core i5 6200U 2.4 GHz (4 cores)
RAM	16 GB RAM DDR3
Storage	256 SSD drive 1TB HDD drive
Host OS	Archlinux (Kernel version 5.2)
Network	Realtek 1Gbps Ethernet, Intel802.11ac wireless adapter

(enp0s8) and the second one (enp0s3) for management traffic. These *veth* interfaces are connected to the enp0s8 and enp0s3 interfaces inside the VM guest operating system.

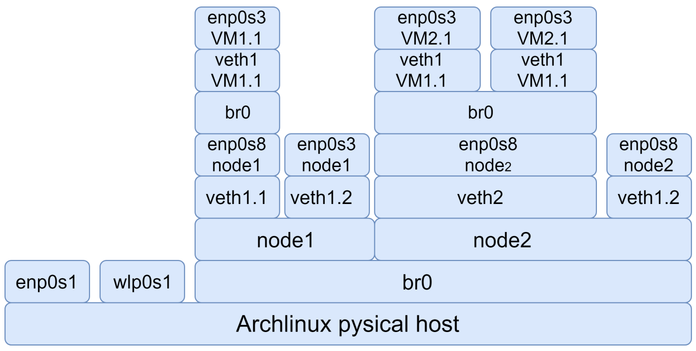


Fig. 6.7 Logical Implementation Diagram

LXD live migration is dependent on the Checkpoint/Restore in user-space (CRIU) library, since CRIU must be aware of the particularities of the process to checkpoint and the way it utilizes the underlying resources. Thus, checkpoint creation has several restrictions. A manual compilation of the CRIU is required to guarantee a successful migration. In addition, even when using the latest compiled version available, live migration is not possible in all cases. In particular, any guest OS that uses systemd as the method to manage user processes fails to checkpoint due to systemd's use of shared filesystem mounts that interfere with the CRIU checkpoint process. Non-systemd distributions, such as Devuan or Alpine are available and can be migrated using this library.

6.7.1 KVM

Migration of the VM was done using the CLI with qemu+ssh. However, the virtual hard disk (vHDD) needs to be migrated in advance.

6.7.2 LXD

Migration over LXD [46] is done using CRIU and an embedded functionality called LXC move. A preshared SSH key is required between two hosts to attempt migration. Then the container information and snapshots are migrated as a delta of the original image. If the migration is successful, the container is deleted from the source and started on the target server.

At the time of initiating the migration, the local LXD daemon checks for the existence of the declared container. A token is created by the local LXD daemon and sent to the remote target daemon, with the source URL and the local certificate identifying LXD local daemon. Then, the remote LXD daemon connects to the local daemon via a control websocket, using the provided token and the transfer mechanism is then negotiated depending on the backend storage being used.

6.7.3 Docker

Docker migration is not defined as a feature in Docker documentation. A checkpoint and restore capability using CRIU and runc is available under experimental conditions. This capability is not production-ready and it is evaluated for comparison purposes with other technologies. A checkpoint of the present state of the container is done in this scenario, which requires the container to be stopped in advance. Docker migration scripts were created using Bash scripting language to provide migration capabilities based on the work in [58].

6.7.4 Unikernels

A Unikernel was compiled as part of this research to evaluate the performance of live migration of Unikernel images over KVM. In order to compile a Unikernel, a target application needs to be defined, for this purpose, a simple NMS was used. OSv was successfully used to compile a Unikernel with the NMS. The migration algorithm for Unikernel images is identical to KVM images, since the result of the Unikernel compilation process is a Quick Emulator copy-on-write (QCOW2) KVM disk image. A VM with similar vCPU and vRAM was created using the QCOW2 image compiled.

6.7.5 Migration Results

The specification for the VM being used to house the NMS is given in Table 6.4. The four service migration methods were evaluated in terms of NMS migration time and the results are listed in Table 6.5.

Table 6.4 VM Specifications

Name	ID	HDD	RAM	vCPU	Network	OS
VM	122	32 GB	2 GB	2	1Gbps	Ubuntu 18.04

Table 6.5 Service Migration Results

Migration Mechanism	**NMS Migration Time**
Unikernel KVM	11.99 s
LXD CRIU	24.73 s
Docker Container	73.00 s
KVM	824.00 s

In order to understand the impact of the service migration time within a dynamic high mobility network environment, the VANET testbed deployed at Middlesex University was used. It consists of 7 RSUs with three RSUs: 5, 6 and 7, located along the A41 which runs behind the Hendon Campus. Using OBUs in vehicles, it was possible to measure the different coverage parameters at 30 mph and at 50 mph. This is shown in Table 6.1.

Using the results from Tables 6.5 and 6.1, we can evaluate the performance of the four service migration methods under two scenarios: reactive and proactive service migration of NMS within a real dynamic VANET environment.

6.8 Using the ZSMC to Look at Service Migration

Service migration will cause a loss of service. In order for us to analyse this further. We can use the same Markov Chain model as shown in Figure 5.9 on page 61. Hence, in order to study the service migration, then v_1 the outward rate from Markov Chain 1 to Markov Chain 0 is the inverse of the Service Hold Time, given by $v_1 = 1/SHT$ where SHT is the Service Hold Time in the current network. The inward rate from Markov Chain 0 to Markov Chain 1, $v_2 = 1/SMT$.

6.8.1 Results to be evaluated

Given the setup of the Middlesex testbed, it was decided to evaluate valid transitions that would occur on the testbed. Those transitions were divided into 2 parts. The first set was transition between the Hendon-based RSUs (RSU1, RSU2, RSU3, RSU4). The second set was between the RSUs on the A41 (RSU5, RSU6, RSU7). The system capacity K is set to 100, the arrival rate λ is set to 25 and the service rate μ is 50.

Table 6.6 Average Number of Requests for Reactive Service Migration for 30 MPH

Transition	Unikernels	LXD	Docker	KVM
RSU1 to RSU2	54.1623	NP1	NP2	NP2
RSU2 to RSU3	35.6863	73.3825	NP2	NP2
RSU3 to RSU4	31.5495	NP2	NP2	NP2
RSU5 to RSU6	17.1508	34.6469	NP1	NP2
RSU6 to RSU7	12.3054	24.5609	70.7721	NP2

Table 6.7 Average Number of Requests for Reactive Service Migration for 50 MPH

Transition	Unikernels	LXD	Docker	KVM
RSU1 to RSU2	92.8323	NP2	NP2	NP2
RSU2 to RSU3	59.4877	NP2	NP2	NP2
RSU3 to RSU4	NP2	NP2	NP2	NP2
RSU5 to RSU6	27.9998	57.2495	NP2	NP2
RSU6 to RSU7	19.8738	40.3166	NP2	NP2

The program developed for analysing handover was therefore modified to evaluate service migration for the different migration mechanisms for each of the transitions given above. The results obtained were the average number of requests, the Response time, the Probability of Blocking P_B and the Probability of lost service, P_{LS} for each transfer mechanism from one RSU to another RSU. $NP1$ is the condition where it is not possible to migrate from the first network because the Service Hold Time in that network is less than zero. Hence it is not possible to migrate to that first network in the first place. $NP2$ is the condition where it is not possible to migrate to the second network because the Service Hold Time in that second network is less than zero.

6.8.2 Reactive Service Migration Results

The main equation for reactive migration is given by Equation 6.3. For a number of mechanisms, we first need to find out if reactive service migration is possible according to this equation. In the tables below, Table 6.6 and Table 6.7 show the average number of requests for 30mph and 50 mph, while Table 6.8 and Table 6.9 show the Blocking Probability, while Table 6.10 and Table 6.11 show the Probability of Lost Service.

6.8.3 Evaluation of the results

These results are very mixed. Unikernels showed that it was the fastest migration mechanism. LXD also showed good results for most transitions. Docker showed mixed results and was unsuitable for RSUs with relatively small coverage ranges while KVM could not be used for

Table 6.8 Blocking Probability % for Reactive Service Migration for 30 MPH

Transition	Unikernels	LXD	Docker	KVM
RSU1 to RSU2	40.7124	NP1	NP2	NP2
RSU2 to RSU3	26.1052	68.8911	NP2	NP2
RSU3 to RSU4	22.9275	NP2	NP2	NP2
RSU5 to RSU6	12.0263	29.2422	NP1	NP2
RSU6 to RSU7	8.4009	20.4439	66.9771	NP2

Table 6.9 Blocking Probability % for Reactive Service Migration for 50 MPH

Transition	Unikernels	LXD	Docker	KVM
RSU1 to RSU2	79.6230	NP2	NP2	NP2
RSU2 to RSU3	45.1153	NP2	NP2	NP2
RSU3 to RSU4	NP2	NP2	NP2	NP2
RSU5 to RSU6	20.2193	49.1825	NP2	NP2
RSU6 to RSU7	14.0727	34.2081	NP2	NP2

Table 6.10 Probability of Lost Service % for Reactive Service Migration for 30 MPH

Transition	Unikernels	LXD	Docker	KVM
RSU1 to RSU2	53.5342	NP1	NP2	NP2
RSU2 to RSU3	35.2143	72.6958	NP2	NP2
RSU3 to RSU4	31.0569	NP2	NP2	NP2
RSU5 to RSU6	16.4823	34.0240	NP1	NP2
RSU6 to RSU7	11.5503	23.8431	70.4191	NP2

Table 6.11 Probability of Lost Service % for Reactive Service Migration for 50 MPH

Transition	Unikernels	LXD	Docker	KVM
RSU1 to RSU2	89.3312	NP2	NP2	NP2
RSU2 to RSU3	57.7092	NP2	NP2	NP2
RSU3 to RSU4	NP2	NP2	NP2	NP2
RSU5 to RSU6	27.4778	56.7216	NP2	NP2
RSU6 to RSU7	19.2496	39.7363	NP2	NP2

reactive migration mechanisms. The main reason for the long delay in KVM migration is because KVM attempts to migrate the whole Virtual Hard Disk (vHDD) rather than just the amount of disk and memory being used.

6.9 Proactive Service Migration

For proactive migration, the results are shown below.

Figure 15: RSU1 to RSU2

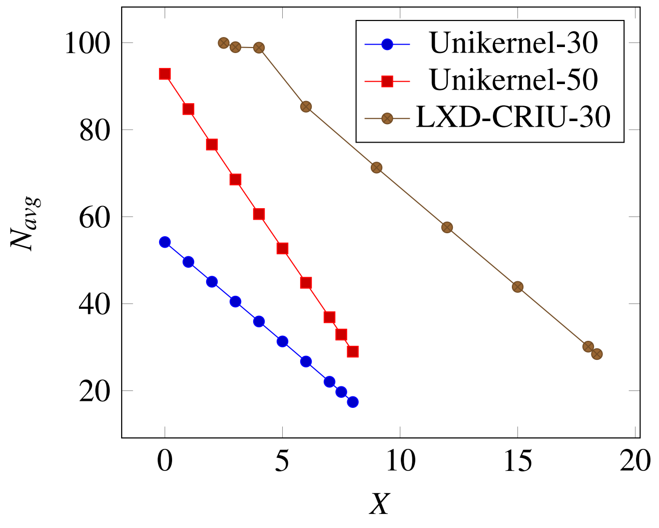


Figure 16: RSU2 to RSU3

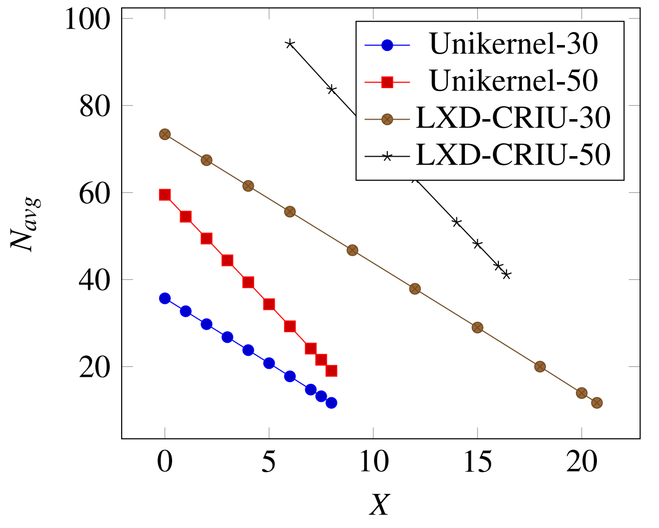


Figure 17: RSU3 to RSU4

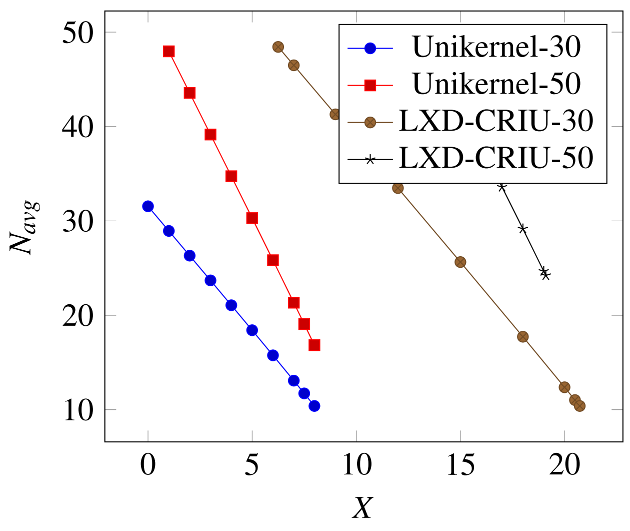


Figure 18: RSU5 to RSU6

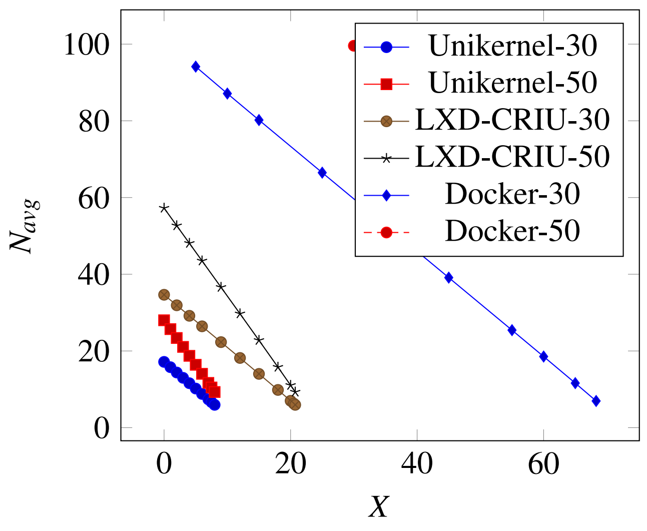
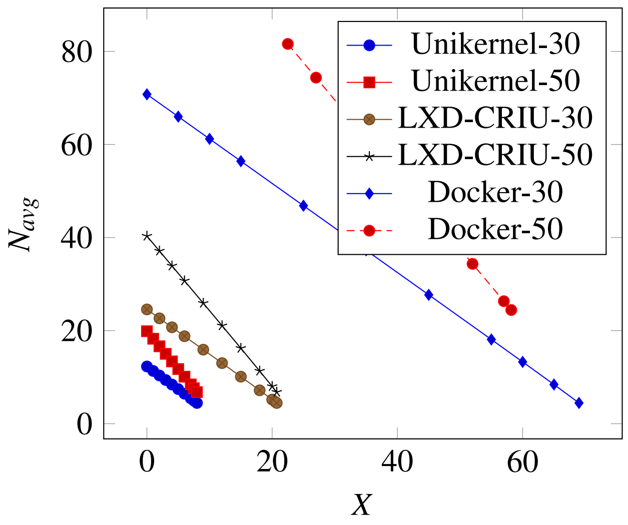


Figure 19: RSU6 to RSU7



The results for the average number of packets in the system for each transition are shown Figures 15-19. What is interesting is that for the transfers on the Hendon Campus, (RSU1 to RSU4) only Unikernel and LXD-CRIU are able to do the migration for all transitions. Docker and KVM are unable to do service migration for any of these transition. However, for the RSUs on the A41, Docker was able to do all the transitions. This is because the Service Hold Time is larger in these RSUs compared to the Hendon RSUs. This enables the Docker Service Migration time which is 73 seconds to be handled by the proactive time X in the RSUs.

6.9.1 Results for the Probability of Blocking

The results for the blocking probability for each transition are shown in Figures 20-24. Again the graphs are very linear. For the RSUs on the Hendon Campus only Unikernels and LXD-CRIU showed a good set of results. Docker and KVM did not have any readings due to their large migration times. For the transitions on the A41, Docker was able to generate results.

Figure 20: RSU1 to RSU2

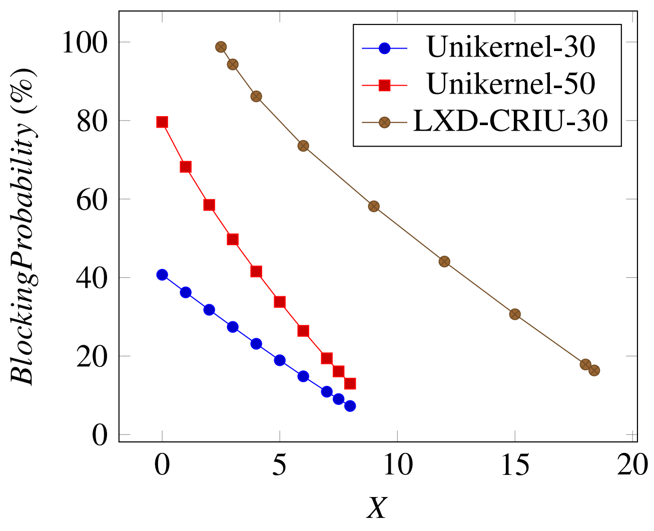


Figure 21: RSU2 to RSU3

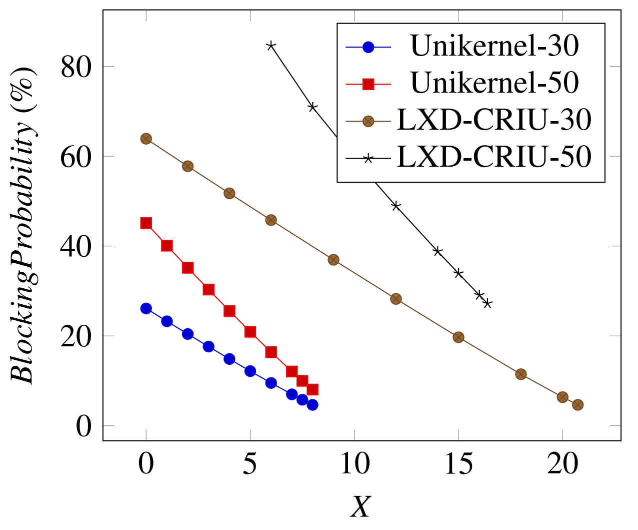


Figure 22: RSU3 to RSU4

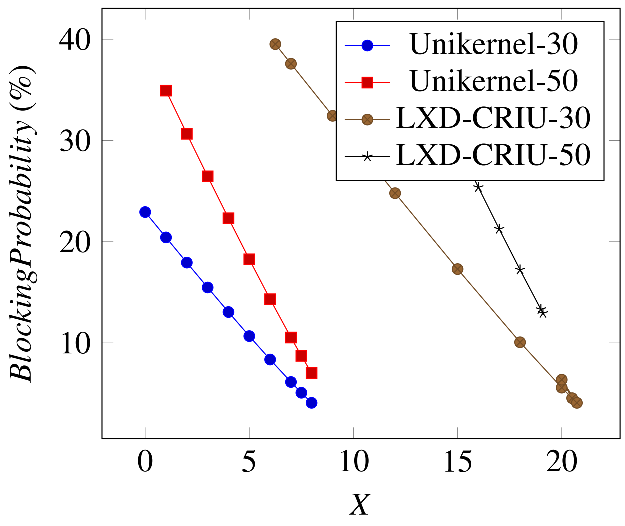


Figure 23: RSU5 to RSU6

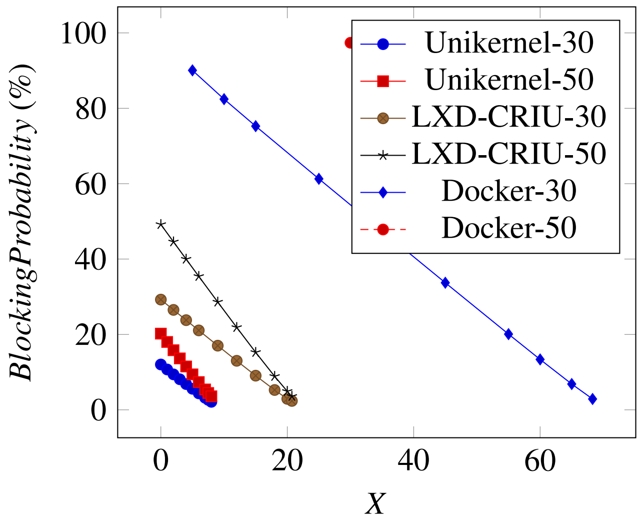
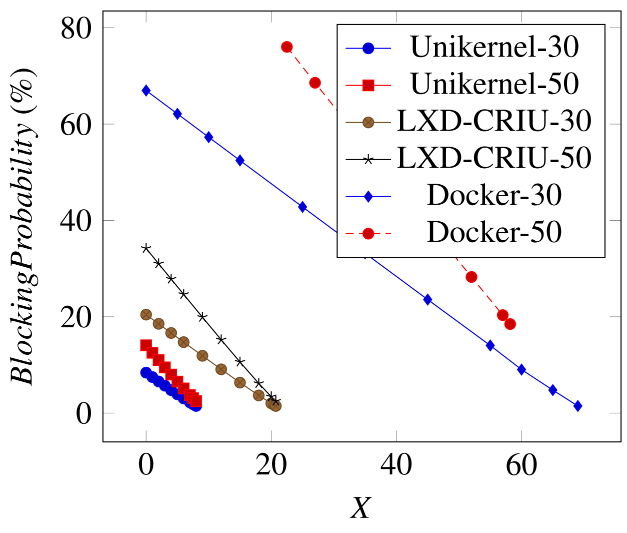


Figure 24: RSU6 to RSU7



6.9.2 Results for Lost Service

Figure 25: RSU1 to RSU2

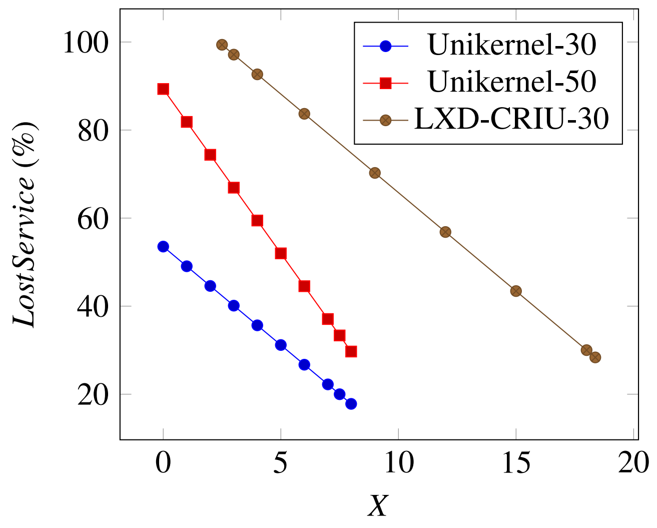


Figure 26: RSU2 to RSU3

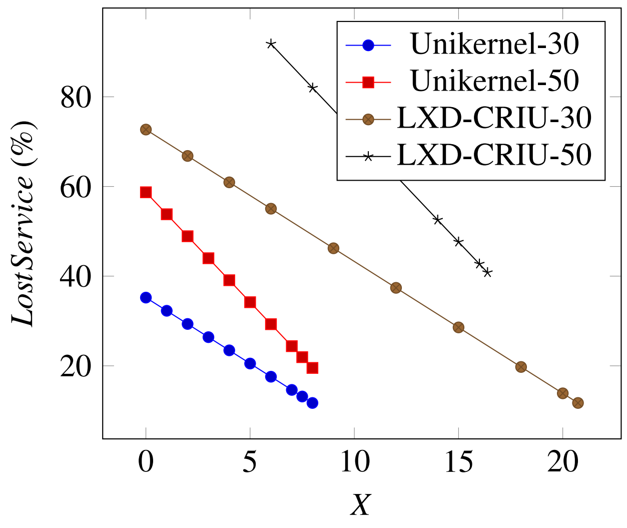


Figure 27: RSU3 to RSU4

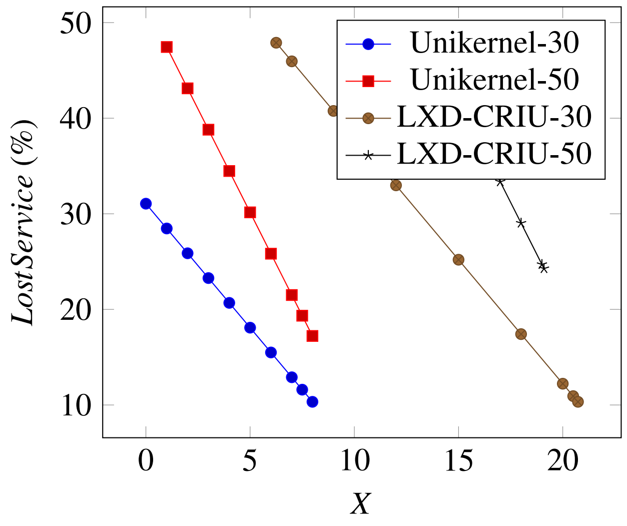


Figure 28: RSU5 to RSU6

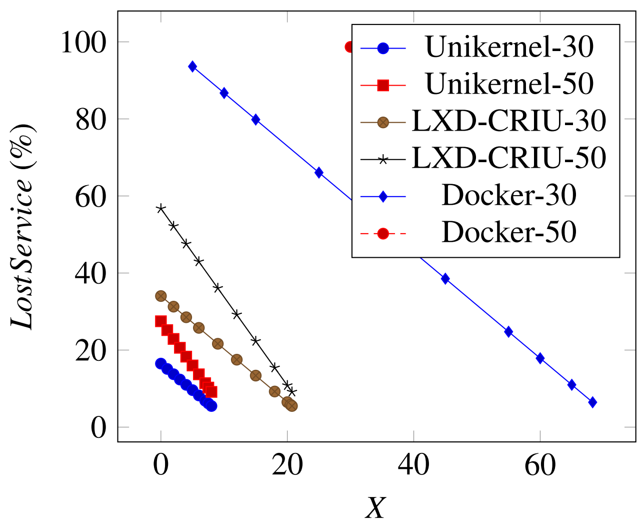
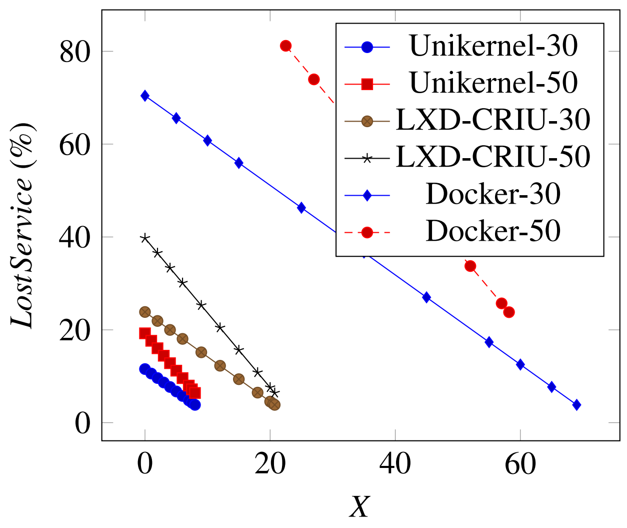


Figure 29: RSU6 to RSU7



Again these results show that by using proactive service, there is a very linear relationship between the Proactive Time (X) and the Probability of Lost Service. This means that it should be possible to use Proactive Service migration to do intelligent service migration.

For 50mph, the effect due to service migration is more severe as the rate of leaving the Service Based Markov Chain is greatly increased due to a much smaller Service Hold Time. This is clearly reflected in the results in terms of higher average number of packets, greater Blocking Probability and greater Probability of Lost Service for 50mph.

Proactive Service Migration will also have a greater effect on the 50mph results as showed by the steeper curves for the 50mph results compared to the 30mph results. However, at 50mph the possible values of X, the proactive service migration time, also decreases and so the range of X will also be mimimised.

Table 6.12 Minimum and Maximum Values of X for Unikernels

Transition	Code	X_{min30}	X_{min50}	X_{max30}	X_{max50}
RSU1 to RSU2	Y	0	0	7.99	7.99
RSU2 to RSU3	Y	0	0	7.99	7.99
RSU3 to RSU4	Y	0	1.0	7.99	7.99
RSU5 to RSU6	Y	0	0	7.99	7.99
RSU6 to RSU7	Y	0	0	7.99	7.99

Table 6.13 Minimum and Maximum Values of X for LXD-CRIU

Transition	Code	X_{min30}	X_{min50}	X_{max30}	X_{max50}
RSU1 to RSU2	Y	2.5	NP1	18.37	NP1
RSU2 to RSU3	Y	0	6.0	20.73	16.39
RSU3 to RSU4	Y	6.25	14.0	20.73	19.1
RSU5 to RSU6	Y	0	0	20.73	20.73
RSU6 to RSU7	Y	0	0	20.73	20.73

6.9.3 Looking at Minimum and Maximum Values for Proactive Service

In order to explore this further the following tables are shown for the maximum and minimum values of X for each mechanism and for each transition. This is shown in Table 6.12 for Unikernels, Table 6.13 for LXD-CRIU, Table 6.14 for Docker and Table 6.15 for KVM.

In the KVM case all the minimum required proactive service migration time X_{min} is always greater the maximum possible value of X given as X_{max}. Thus, we have the condition known as NP3 and service migration cannot be performed using this mechanism.

As previously noted, for the Hendon-Based RSUs, Docker is also adversely affected, but does much better with the A41 RSUs. However X_{min} increases and the X_{max} decreases for 50mph compared to 30mph.

LXD-CRIU could generally be used in most cases. However, the transition from RSU1 to RSU2 for 50mph could not be done because the time in the first network is too small (9.42s)

Table 6.14 Minimum and Maximum Values of X for Docker

Transition	Code	X_{min30}	X_{min50}	X_{max30}	X_{max50}
RSU1 to RSU2	NP3	39.00	52.6	18.37	9.42
RSU2 to RSU3	NP3	34.45	49.87	30.00	16.4
RSU3 to RSU4	NP3	54.51	61.91	34.55	19.13
RSU5 to RSU6	Y	5	30.0	68.299	39.57
RSU6 to RSU7	Y	0	22.5	69.0	58.18

Table 6.15 Minimum and Maximum Values of X for KVM

Transition	Code	X_{min30}	X_{min50}	X_{max30}	X_{max50}
RSU1 to RSU2	NP3	790	803.6	18.37	9.42
RSU2 to RSU3	NP3	785.45	800.87	30.00	16.4
RSU3 to RSU4	NP3	805.51	812.91	34.55	19.13
RSU5 to RSU6	NP3	720.36	761.81	68.63	39.57
RSU6 to RSU7	NP3	739	773	99.64	58.19

compared to the service migration time (24.73) and hence $SHT < 0$, and thus error code, $NP1$ was returned.

The Unikernel mechanism showed the best results mainly because of its speed. It worked quite well in all circumstances except for the RSU3 to RSU4 transition for 50mph because of a reduced RHT for RSU4 at 50 mph (7.09s) which resulted a slight increase in X_{min}. Thus going forward we believe it is imperative to look at using the Unikernel system for service migration in vehicular networks.

6.10 Chapter Summary

This chapter explored a detailed view of service migration and showed how it can be managed using the analytical modelling. Both reactive and proactive service migration were studied and a comprehensive set of results were produced. It revealed that Unikernels were great: LXC-CRIU was generally OK, Docker gave mixed results while KVM cannot be used for practical service migration. This work will be incorporated into a new Service Management Framework in the next chapter to provide mobile users with a sustained QoS in vehicular networks.

Chapter 7

Developing a Practical Intelligent Service Management System

7.1 Brief Introduction

This Chapter starts by revealing a flow diagram based on the work of the previous Chapters of this thesis of how intelligent service migration could be implemented. It then discusses Sardis's framework as a Reference Model and then examines an implementation model for Intelligent Service Migration as well as key routines and structures needed to implement a Service Management Framework. A Simple prototype of the system is then developed and evaluated.

7.2 Flow Chart for Intelligent Service Migartion

Based on our previous analysis in chapter 5 and chapter 6, Figure 7.1 reveals the flow diagram that would be needed to implement intelligent service migration in a vehicular network.

As previous discussed, the MN polls the core network about the network topology and will receive information about the target RSU including its NDT, Time to Handover, and services that can be migrated via the 802.11p or 5G network using different migration mechanisms. When this information gets to the MN, it transfers them to the Service data unit along with the car's velocity, Application QoS and its bandwidth and latency information derived from SLTP. When this data is received by the Service data unit it selects the best transfer technique, either through Unikernel or LXD-CRIU. This is transferred to the migration execution unit. Service migration decisions are then executed for a proactive handover, reactive handover, etc.

If we can do a Proactive service migration, then the system checks if the Reactive service migration is also possible. This is true if Xmin is 0. If yes, the system checks if the QoS for the service in the current network is better than the QoS for service in the next network. If yes, it informs the Service Management Framework (SMF) to begin Service Migration (SM) at the same time as the communication handover is done by the lower layers, which will allow the MN to stay as long as possible in the current network to experience the better QoS. If not, then SMF is told to start SM at Xmax in order to leave the current network early and hence experience the better QoS in the next network.

If it is not possible to do Reactive service migration, but the QoS on the current network is still better than the QoS in the next network, then the system will tell the SMF to start SM at Xmin, else start at Xmax.

Finally, if neither Proactive nor Reactive service migration can be done to the next RSU, the SMF is told to migrate the service via a VANET Server back to an Edge Cloud System which can be accessed by the MN through the next RSU. Hence service can be maintained though service migration to the next RSU cannot occur.

7.2.1 Service-Oriented Framework for Mobile Services

The work done by Sardis [72] showed that in order to migrate a service, it is necessary to compare the time taken to migrate the service with the amount of time the user will be in that specific region. This is a compelling framework that can be used to allow services to migrate in a MEC environment. Although Sardis used a simple queuing model to represent the user's mobility in mobile networks, his framework can be considered as a reference framework that can be used as a starting point to look at practical service frameworks for vehicular networks.

7.3 New Service Management Framework

Figure 7.2 shows the old environment where the server had to manage replications of itself on different systems, below it reveals the new environment where this is managed by the SMF.

The main challenge that needs to be addressed in order to build a viable service management framework for vehicular networks is the need to guarantee seamless communication in such environments [25], since the performance of any service architecture must depend on the performance of the underlying communication dynamics [21].

The new proposed implementation framework is based on the service framework developed by Sardis and has four layers. The first layer is called an Application Layer that runs

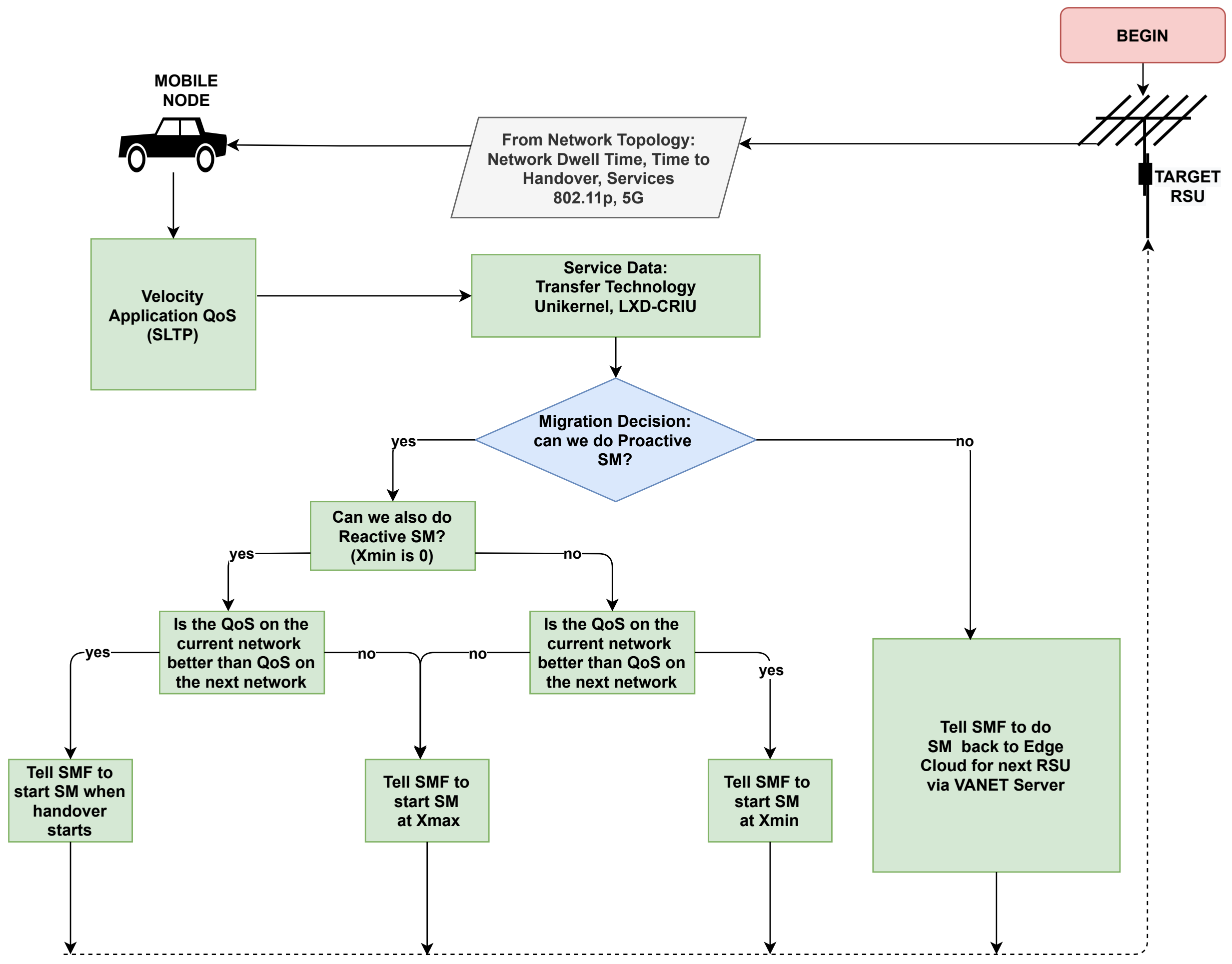

BEGIN
TARGET RSU
MOBILE NODE
From Network Topology: Network Dwell Time, Time to Handover, Services 802.11p, 5G
Velocity Application QoS (SLTP)
Service Data: Transfer Technology Unikernel, LXD-CRIU
Migration Decision: can we do Proactive SM?
yes
no
Can we also do Reactive SM? (Xmin is 0)
yes
no
Is the QoS on the current network better than QoS on the next network
yes
no
no
Is the QoS on the current network better than QoS on the next network
yes
Tell SMF to start SM when handover starts
Tell SMF to start SM at Xmax
Tell SMF to start SM at Xmin
Tell SMF to do SM back to Edge Cloud for next RSU via VANET Server

Fig. 7.1 Flow Chart for Intelligent Service Migartion

on the MN and invokes the service through the Service Management Layer (SManL) giving the service name, Service id and required QoS. The SManL administrates the service and is also responsible for Service Subscription and Service Delivery. The Service Migration Layer (SML) migrates the service as requested by SManL. It uses a secure Resource Allocation Security Protocol (RASP) [35] to ensure that the transfer is securely done. RASP in turn, will use standard migration mechanisms such as Docker, KVM, LXD and Unikernels to do the actual migration. SML informs SManL when the migration is completed. The Service Connection Layer (SCL) monitors the connection between the mobile node and the server

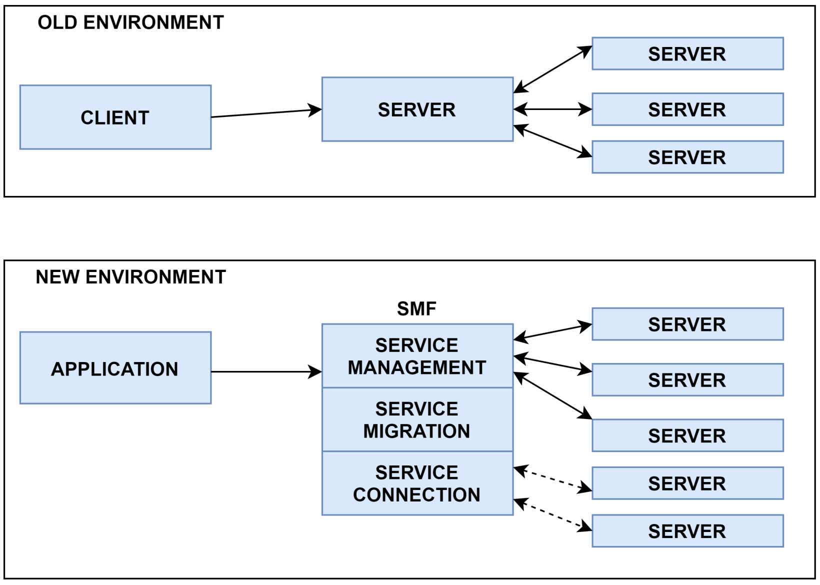


Fig. 7.2 Structure of the New Service Management Framework

and the reports to the SManL when the mobile node is no longer contactable due to handover to another network. The overall structure is shown in Figure 7.3.

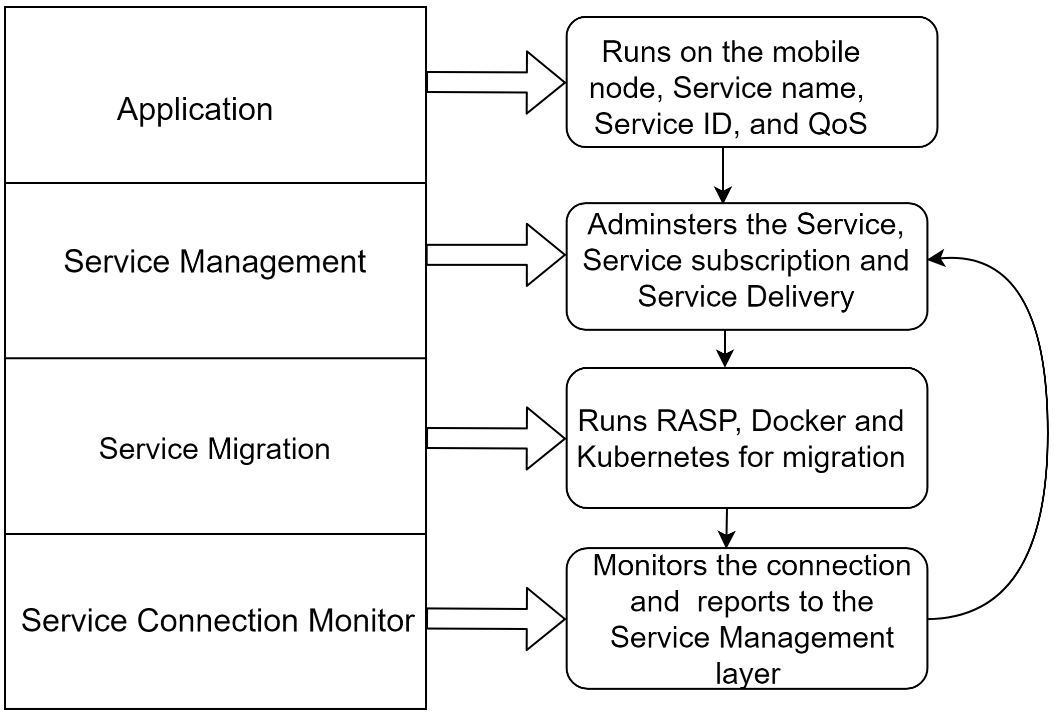

Application
Service Management
Service Migration
Service Connection Monitor
Runs on the mobile node, Service name, Service ID, and QoS
Adminsters the Service, Service subscription and Service Delivery
Runs RASP, Docker and Kubernetes for migration
Monitors the connection and reports to the Service Management layer

Fig. 7.3 Service Migration Prototype

Table 7.1 General Notations

Notations	Name
AL	Application Layer
SManL	Service Management Layer
SML	Service Migration Layer
SCL	Service Connection Layer
S1, S2... Sn	Server
QoS(Req)	Quality of Service (Required)
MC	Mobile Client

7.3.1 Protocols for Intelligent Service Migration in a Network Scenario for a Vehicular Environment

Interaction between Servers and the Service Management Framework

- **Step 1:** All servers must register with the SManL using the steps below.
- **Step 2:** The Cloud Management system which is responsible for running a number of services calls SManL to register a service with the following parameters: [Server name, Server version, Resource requirements (CPU, Memory, Network, Storage), Restriction list, Security level, QoS, Location Restrictions, Maximum Replicas, Actual binary of the service, and Container Images.]
- **Step 3:** SManL allocates a new service structure and Service ID and adds the new service structure to the list of services it supports.
- **Step 4:** SManL tells the Cloud Management System that the service has been registered. It returns the Service ID and the Service Capability for the new service.

Interaction between MC and Service Management Framework

- **Step 1:** All MCs must be connected to the SManL by using SCL, via a special port to talk to the SManL.
- **Step 2:** Application Layer (AL) requests a service from SManL: AL Sends the following information to SManL [Node ID, Service name, Service version, QoS Requirements, Location, and Network Interfaces.] Then SManL takes the service name and service version from the AL's request, then scans through a list of services

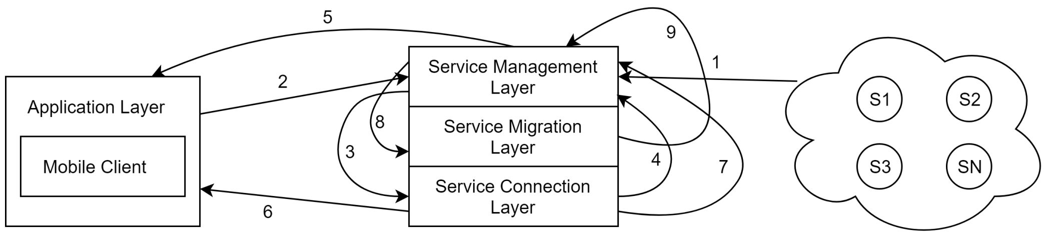


Fig. 7.4 Interaction between MC and Service Management Framework

already stored in its database, if there is a valid service for the AL request then it will return the Service Structure for that service which contains a list of servers that is currently running that particular service.

- **Step 3:** SManL to SCL: SManL sends the following MC information to SCL [Node ID, Service name, QoS Requirements, Service ID, Location, and Node IP address.] The SCL has a list of nodes that it is already pinging in order to keep track of different network IDs and services.

  [Case 1:] If the SCL is already pinging MC, it goes to Step 4.

  [Case 2:] If SCL is not pinging MC, it starts pinging it. If pinging is successful, SCL adds the Node ID and IP address to its database and go to Step 4

  [Case 3:] If SCL cannot ping MC, it informs the SManL it cannot connect.

- **Step 4:** SCL to SManL: The SCL sends a connection successful message to the SManL. The SManL updates its lists of MCs using that service and moves on to Step 5.

- **Step 5:** SManL to MC: The SManL sends [Node ID, Service name, Service ID, Location, QoS Requirements and Server IP address] to the MC. MC communicates with the servers [S1, S2... Sn] using the network interface which it received from SManL and goes to Step 6.

- **Step 6:** SCL to MC: The SCL pings the MC to monitor its presence within the network.

  [Case 1:] An internal loop is created (pinging between SCL and MC), this continues to run in a loop.

  [Case 2:] If ping fails or MC moves away from the network region go to Step 7, else go back to the inner loop in case 1.

  [Case 3:] If the MC finishes with the service, the AL on MC sends a message to SManL to close down the services (Socket to server is closed). Then SManL talks to SCL to stop pinging and exit.

- **Step 7:** SCL to SManL: The SCL informs the SManL that the pinging failed given following information [Node ID, IP address] SManL finds the MC's location and finds a new server given the direction and location of MC.

  [Case 1:] If the chosen location already has a server on the SManL database, it updates the database with the new server for the client and moves to Step 9.

  [Case 2:] Else, we need to migrate the service to a chosen server location and move on to Step 8.

- **Step 8:** SManL to SML: SManL informs SML to migrate the services to new server location by using a RASP protocol then heads on to Step 9.

- **Step 9:** SML to SManL: The SML informs SManL that the service migration has been completed. Since all MCs are required to contact SManL after handover or when in new network, it moves to Step 10.

- **Step 10:** MC to SManL: The MC will inform the SManL of the services that is still running on the MC and hence the process is repeated from Step 2 in the new network.

7.3.2 C Code results

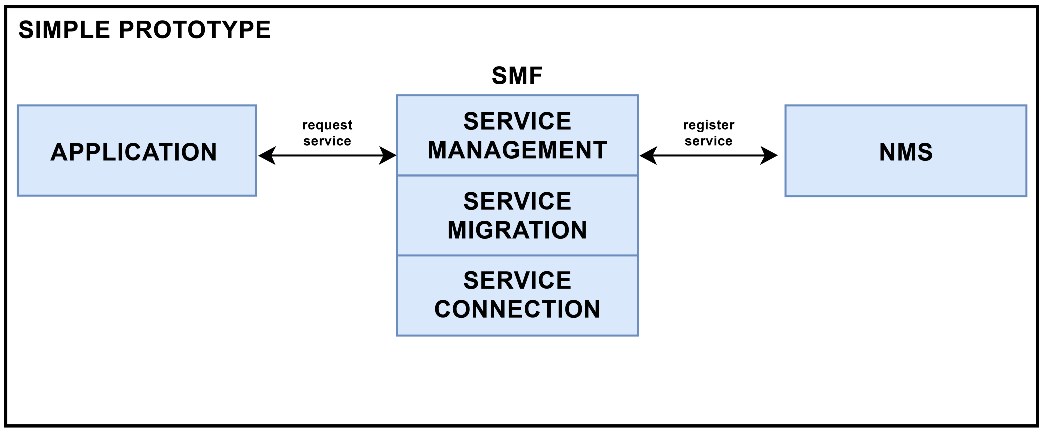


Fig. 7.5 Simple Prototype in C language

Figure 7.5 shows a simple prototype which was written in the C language, the source code can be found in the appendix of this thesis. We wrote routines to register a service with the SMF and to request a service.

The three separate programs (a basic SManL, Register Service and Request Service) showed how we got a basic SMF to work. Results were obtained when all 3 programs are run on the same machine. The machine is a Core i7 processor running a 3,5.3 Linux Operating system.

When registering a Network Memory Server service with the SMF we get the following results:

- **Registering an NMS Service:** User Time spent in total: 0.168s and System Time taken: 0.028s
- **Requesting an NMS Service:** User Time spent in total: 0.180s and System Time taken: 0.032s

7.4 Chapter Summary

This chapter explored the development of a new service management framework for vehicular environments. Then it revealed the functions for intelligent service migration in VANET. Lastly, a basic prototype was developed using the C programming language and preliminary performance results were obtained.

Chapter 8

Conclusions and Future work

In this chapter, a summary of the accomplished work is offered and the major contributions to knowledge are highlighted. This is followed by conclusions derived from the work done so far and proposed future work for the research is offered.

8.1 Contribution of the Thesis

The contributions of this thesis can be summarised as follows:

Chapter 1- In this thesis, we began by motivating the need for intelligent service migration in a highly mobile environments such as vehicular networks. It showed the key research questions that needed to be addressed in the thesis as well as its research benefits with key contributions. Furthermore, it showed the list of papers published to validate the researchers claims.

Chapter 2- This chapter was broken down into two parts. The first part focused on the technical background which introduced us to a number of technologies starting with the Y-Comm Reference Framework which was developed for future communication, then Multi-access edge computing that helped in providing the network architecture needed for VANET and a list of migration technologies that were useful for this research progression. The second part displayed the related work section which represented an intensive analysis of different research efforts, that investigated MEC, computational offloading decisions and existing migration models as well as a solution approach to developing intelligent service migration.

Chapter 3- This chapter mainly concentrated on three specific methods to address the key methodology used in the thesis. Firstly, this chapter introduced analytical modelling, by discussing how the analytical model is developed and showed its key parameters to create an estimated solution. Secondly, it detailed the construction of a testbed that will enable us to perform experiments. Lastly, the use of NMS in a file system known as FUSE as a service to show how we could intelligently migrate services across the network in the VANET testbed.

Chapter 4- In this chapter, it revealed the implementation framework of Y-Comm which led to the design of SLTP which is a new experimental transport protocol. Furthermore, a comprehensive analysis of SLTP was revealed, performance tests were carried out to show its validity. SLTP was compared with the standard TCP. They both went through a stress test with different loads on the VANET RSUs located the Middlesex University, results showed SLTP had better performance.

Chapter 5- The analysis of the effect of handover on service delivery in a vehicular network was displayed in this chapter. It was achieved by the application of the Zero Service Markov chain technique. This analytical model allowed for the development of a new parameter called the Probability of Lost Service. This model was applied in the MDX VANET testbed and interesting results were obtained.

Chapter 6- This chapter developed a model for service migration. The system used different migration techniques, such as KVM, Docker, LXD and Unikernels to migrate a network memory server. Both reactive and proactive service migration were investigated, and a comprehensive set of results was produced. They showed that presently, Unikernels is the best service migration mechanism for vehicular networks.

Chapter 7- This chapter revealed a flow diagram of how to integrate intelligent service migration. A newly developed service management framework for vehicular networking was created. New functions and routines were developed to implement a framework for intelligent service migration. An intelligent service migration prototype was implemented and evaluated. The C programming language was used to code the protocols and results were achieved to prove its validity. This new approach appeared to be much better than the current approaches because it addressed the key issues in intelligent service migration in highly mobile environments.

8.2 Contribution to Knowledge

This thesis presents and explores intelligent service migration in a highly mobile network such as VANET as a means of improving the QoS and managing network traffic. The proposed mechanisms for intelligent service migration framework consider user and service requirements and characteristics and combine them with network conditions and user mobility to determine the best way to migrate a service. The uniqueness of the presented framework lies in the convergence of MEC technology, user mobility and future networks.

8.3 Conclusion

This thesis presented an analysis of the effect of handover on service delivery in a vehicular network. It was attained by applying the ZSMC technique. This analytical model allowed for the development of a new parameter called the Probability of Lost Service(P_{LS}). This model was applied in the MDX VANET testbed. An investigation of the results showed that P_{LS} was slightly dependent on the system capacity (K) and utilisation ρ. P_{LS} appears to be only significantly dependent on the rates of transition between the ZSMC and the SBMC.

This method of approach helped to develop our model for service migration, we used different migration techniques, such as KVM, Docker, LXD and Unikernels to migrate a network memory server. Reactive and proactive service migration were investigated, and a comprehensive set of results was produced. It showed that Unikernels is the best service migration mechanism for vehicular networks.

Furthermore, a flow diagram of how to integrate intelligent service migration was developed as well as a service management framework for the vehicular environment was created. Innovative functions and routines were developed to implement a framework for intelligent service migration. An intelligent service migration prototype was implemented and evaluated using simple C programs, results were achieved to prove its validity. This new approach presented in this thesis appeared to be much better than the current approaches because it addressed the key issues in intelligent service migration in a vehicular network.

8.4 Future Work

This thesis highlights a few areas that are subject to future research. This section proposes future work and work currently in progress in the fields of Security and machine learning.

8.4.1 Security

Even though the security of service migration in vehicular network falls outside the scope of this thesis and is currently being explored by researchers [36], there are several other aspects of the proposed system that present potential security weaknesses. The main security concern is that of ensuring that performance data for an individual MN and service are exchanged in a way that prevents impersonation or altering. Creating and securing a telecommunication channel between the client device and the service is a necessity for preventing malicious users from inputting false data into the system causing false migrations. There are several different attacks that can be used against the system involving false prevention or triggering of a migration, altering with the destination of the migration by giving false user location and overloading a MEC by a synchronised attack that forces services to migrate to it. To prevent such attacks there are three proposed guidelines for future research.

Firstly, the security mechanisms on the MEC should prevent any services from migrating in or out until they first establish an agreement between the source and destination MEC. This would be covered in the proposed service delivery framework as part of the migration layer which receives information from the Service Management layer to confirm which services can move depending on their requirements and the capabilities of the target destination. Secondly, the mechanisms that handle this data on the MN and MEC side should be validated to avoid malicious programmers from tampering with the data at the point where it is gathered and processed. Then the final requirement is to secure the communication channel between the device and the service so that man-in-the-middle attacks cannot be used to disrupt the performance data.

8.4.2 Machine Learning

This is a method of data analysis that helps to automate analytical model building. It is a branch of artificial intelligence [99] [79] that is based on the concept of systems being able to learn from data, identify certain patterns and make choices with minimum human interference. Live service migration based on Machine learning can be applied in this area. Statistical, probabilistic, and learning based regression models are a new dimension to enhance live migration. The Migration System should be able to automatically adjust its behaviour depending on the workload patterns displayed by the applications. To address this problem, future research on intelligent service migration should apply machine learning algorithms to the estimation of future CPU utilization.

Lastly, the development of an Internet Service Platform using Software-Defined Vehicular Networking [31] techniques, would greatly enhance the development and deployment of servers and services for the Future Internet. This would combine Y-Comm, the work of Frank Sardis, SLTP, the analytical models as well as machine learning and artificial intelligence to enable intelligent service management on a global scale.

References

[1] Abd-Elrahman, E. and Said, A. (2019). Two dimensional markov chain approximation for mptcp over hetnets: Performance evaluation. In *15th International Wireless Computing and Mobile Computing Conference (IWCMC)*.

[2] Aiash, M., Mapp, G., Lasebea, A., and Augusto, M. (2012). A qos framework for heterogeneous network. In *2012 Third Asian Himalayas International Conference on Internet*, pages 1–6.

[3] Badraa, T. and Kinoshita, K. (2018). An energy efficient non-live virtual machine migration. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pages 1–3.

[4] Balakrishnan, K. (2019). *Exponential Distribution: Theory, Methods and Applications*. CRC Press.

[5] Balouch, S. F. and Ahmad, M. (2019). A markov-based framework for handover process in heterogeneous cellular networks. In *2nd International Conference on Computing, Mathematics and Engineering Techniques (iCoMET)*.

[6] Barbarossa, S., Sardellitti, S., and Lorenzo, P. D. (2013). Joint allocation of computation and communication resources in multiuser mobile cloud computing. In *2013 IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 26–30.

[7] Bjorklund, M. (2010). *RFC 6020: YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)*. IETF.

[8] Bolch, G., Greiner, S., de Meer, H., and Trivedi, K. (2006). *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley.

[9] Cao, S., Tao, X., Hou, Y., and Cui, Q. (2015). An energy-optimal offloading algorithm of mobile computing based on hetnets. In *2015 International Conference on Connected Vehicles and Expo (ICCVE)*, pages 254–258.

[10] Chen, M., Dong, M., and Liang, B. (2016a). Joint offloading decision and resource allocation for mobile cloud with computing access point. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3516–3520.

[11] Chen, M., Liang, B., and Dong, M. (2015). A semidefinite relaxation approach to mobile cloud offloading with computing access point. In *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 186–190.

[12] Chen, S. and Zhou, M. (2021). Evolving container to unikernel for edge computing and applications in process industry. *Processes*, 9(2):1–19.

[13] Chen, X., Jiao, L., Li, W., and Fu, X. (2016b). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808.

[14] Cheng, Y. and Li, X. (2020). A compute-intensive service migration strategy based on deep reinforcement learning algorithm. In *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, volume 1, pages 1385–1388.

[15] Cottingham, D. (2009). *Vehicular wireless communication*. PhD thesis, Computer Laboratory, University of Cambridge.

[16] Cottingham, D. N., Wassell, I. J., and Harle, R. K. (2007). Performance of IEEE 802.11a in vehicular contexts. In *Proceedings of the 65th IEEE Vehicular Technology Conference, VTC Spring 2007, 22-25 April 2007, Dublin, Ireland*, pages 854–858. IEEE.

[17] Deng, M., Tian, H., and Fan, B. (2016). Fine-granularity based application offloading policy in cloud-enhanced small cell networks. In *2016 IEEE International Conference on Communications Workshops (ICC)*, pages 638–643.

[18] Docker (2021). Docker company. https://www.docker.com/company.

[19] Elbay, S. K., Hegazy, I., and El-Horabty, E. M. (2017). Analyzing live migration energy overhead in cloud computing: A survey. In *2017 Eighth International Conference on Intelligent Computing and Information Systems (ICICIS)*, pages 356–359.

[20] Ezenwigbo, A., Paranthaman, V. V., Trestian, R., Mapp, G., and Sardis, F. (2018). Exploring a new transport protocol for vehicular networks. In *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pages 287–294.

[21] Ezenwigbo, O. A., Paranthaman, V. V., Mapp, G., and Trestian, R. (2019). Exploring intelligent service migration in vehicular networks. In Gao, H., Yin, Y., Yang, X., and Miao, H., editors, *Testbeds and Research Infrastructures for the Development of Networks and Communities*, pages 41–61, Cham. Springer International Publishing.

[22] Fang, P., Zhao, Y., Liu, Z., Gao, J., and Chen, Z. (2020). Resource allocation strategy for mec system based on vm migration and rf energy harvesting. In *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, pages 1–6.

[23] Farahbakhsh, R. (2009). Smooth handover by synchronizing context transfer protocol and fast mobile ipv6. In *2009 IEEE International Conference on Internet Multimedia Services Architecture and Applications (IMSAA)*, pages 1–5.

[24] Gao, J. and Tang, G. (2013). Virtual machine placement strategy research. In *2013 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pages 294–297.

[25] Ghosh, A., Paranthaman, V. V., Mapp, G., and Gemikonakli, O. (2014). Exploring efficient seamless handover in VANET systems using network dwell time. *EURASIP Journal on Wireless Communications and Networking*, 2014(1).

[26] Gozalvez, J., Sepulcre, M., and Bauza, R. (2012). Ieee 802.11p vehicle to infrastructure communications in urban environments. *IEEE Communications Magazine*, 50(5):176–183.

[27] Han, Z., Zhang, Y., and Wang, R. (2018). A locality live migration strategy based on docker containers. In *2018 IEEE 4th International Conference on Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing, (HPSC) and IEEE International Conference on Intelligent Data and Security (IDS)*, pages 51–54.

[28] He, X., Long, Y., and Zheng, L. (2016). A transparent file encryption scheme based on fuse. In *2016 12th International Conference on Computational Intelligence and Security (CIS)*, pages 642–645.

[29] Hendrawan, Rachmana, N., and Iskandar (2016). Network management system (nms) using local collector mediation devices. In *2016 10th International Conference on Telecommunication Systems Services and Applications (TSSA)*, pages 1–4.

[30] Hussain, R., Son, J., Eun, H., Kim, S., and Oh, H. (2012). Rethinking vehicular communications: Merging vanet with cloud computing. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 606–609.

[31] Isaia, P. and Guan, L. (2016). Performance benchmarking of sdn experimental platforms. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 116–120.

[32] J. Kennedy, R. C. E. (1997). A discrete binary version of the particle swarm algorithm. In *Systems, Man, and Cybernetics*, volume 5, pages 4104–4108. IEEE International.

[33] Jain, R. (2015). *Art of Computer Systems Performance Analysis: Techniques For Experimental Design Measurements Simulation and Modeling*. Wiley.

[34] Kamoun, M., Labidi, W., and Sarkiss, M. (2015). Joint resource allocation and offloading strategies in cloud enabled cellular networks. In *2015 IEEE International Conference on Communications (ICC)*, pages 5529–5534.

[35] Karthick, G., Mapp, G., Kammueller, F., and Aiash, M. (2018). Formalization and analysis of a resource allocation security protocol for secure service migration. In *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, pages 207–212.

[36] Karthick, G., Mapp, G., Kammüller, F., and Aiash, M. (2017). Exploring a security protocol for secure service migration in commercial cloud environments.

[37] Kikuchi, J., Wu, C., Ji, Y., and Murase, T. (2017). Mobile edge computing based vm migration for qos improvement. In *2017 IEEE 6th Global Conference on Consumer Electronics (GCCE)*, pages 1–5.

[38] Kirsal, Y. (2017). Performability evaluation and optimization analysis of repairmen for large-scale networks. In *25th Signal Processing and Communications Applications Conference (SIU)*.

[39] Kirsal, Y., Mapp, G., and Sardis, F. (2019). Using advanced handover and localization techniques for maintaining quality-of-service of mobile users in heterogeneous cloud-based environment. *J. Netw. Syst. Manag.*, 27(4):972–997.

[40] Kleinrock, L. (1975). *Queueing Systems, Volume 1: Theory*. Wiley Interscience.

[41] Kurek, T. (2019). Unikernel network functions: A journey beyond the containers. *IEEE Communications Magazine*, 57(12):15–19.

[42] Labidi, W., Sarkiss, M., and Kamoun, M. (2015a). Energy-optimal resource scheduling and computation offloading in small cell networks. In *2015 22nd International Conference on Telecommunications (ICT)*, pages 313–318.

[43] Labidi, W., Sarkiss, M., and Kamoun, M. (2015b). Joint multi-user resource scheduling and computation offloading in small cell networks. In *2015 IEEE 11th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, pages 794–801.

[44] Lertsinsrubtavee, A., Ali, A., Molina-Jimenez, C., Sathiaseelan, A., and Crowcroft, J. (2017). Picasso: A lightweight edge computing platform. In *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, pages 1–7.

[45] Liu, J., Mao, Y., Zhang, J., and Letaief, K. B. (2016). Delay-optimal computation task scheduling for mobile-edge computing systems. In *2016 IEEE International Symposium on Information Theory (ISIT)*, pages 1451–1455.

[46] Ma, J., Kim, H., and Kim, Y. (2016). The virtualization and performance comparison with lxc-lxd in arm64bit server. In *2016 6th International Conference on IT Convergence and Security (ICITCS)*, pages 1–4.

[47] Mach, P. and Becvar, Z. (2017). Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys Tutorials*, 19(3):1628–1656.

[48] Machen, A., Wang, S., Leung, K. K., Ko, B. J., and Salonidis, T. (2018). Live service migration in mobile edge clouds. *IEEE Wireless Communications*, 25(1):140–147.

[49] Mao, Y., Zhang, J., and Letaief, K. B. (2016). Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605.

[50] Mapp, G., Gosh, A., Paranthaman, V. V., Iniovosa, V. O., Loo, J., and Vinel, A. (2016a). Exploring seamless connectivity and proactive handover techniques in vanet systems. In *Intelligent Transportation Systems*, pages 195–220. Springer.

[51] Mapp, G., Katsriku, F., Aiash, M., Chinnam, N., Lopes, R., Moreira, E., Vanni, R. P., and Augusto, M. (2012a). Exploiting Location and Contextual Information to Develop a Comprehensive Framework for Proactive Handover in Hetegeneous Environments. *Journal of Computer Networks and Communications*.

[52] Mapp, G., Katsriku, F., N., M. A., Chinnam, Lopes, R., Moreira, E., Vanni, R. M. P., and Augusto, M. (2012b). Exploiting location and contextual information to develop a comprehensive framework for proactive handover in heterogeneous environments. *Journal of Computer Networks and Communications*, 2012(Article ID 748163):1–17.

[53] Mapp, G., Sardis, F., and Crowcroft, J. (2016b). Developing an implementation framework for the future internet using the y-comm architecture, sdn and nfv. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 43–47.

[54] Mapp, G., Shaikh, F., Aiash, M., Vanni, R. P., Augusto, M., and Moreira, E. (2009). Exploring efficient imperative handover mechanisms for heterogeneous wireless networks. In *2009 International Conference on Network-Based Information Systems*, pages 286–291.

[55] Mapp, G., Thakker, D., and Gemikonakli, O. (2010). Exploring a new markov chain model for multiqueue systems. In *Computer Modelling and Simulation (UKSim), 2010 12th International Conference*, pages 592–597.

[56] Mapp, G., Thakker, D., and Silcott, D. (2007a). The Design of a Storage Architecture for Mobile Heterogeneous Devices. *ICNS2007*, 0:41.

[57] Mapp, G. E., Shaikh, F., Cottingham, D., Crowcroft, J., and Baliosian, J. (2007b). Y-comm: a global architecture for heterogeneous networking. In *Proceedings of the 3rd international conference on Wireless internet*, page 22. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[58] Marathe, N., Gandhi, A., and Shah, J. M. (2019). Docker swarm and kubernetes in cloud computing environment. In *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, pages 179–184.

[59] Medel, V., Rana, O., Bañares, J. ., and Arronategui, U. (2016). Modelling performance amp; resource management in kubernetes. In *2016 IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC)*, pages 257–262.

[60] N, P. E., Mulerickal, F. J. P., Paul, B., and Sastri, Y. (2015). Evaluation of docker containers based on hardware utilization. In *2015 International Conference on Control Communication Computing India (ICCC)*, pages 697–700.

[61] Narayanan, G. G. and Saravanaguru, R. K. (2018). Securing vm migration through ipsec tunneling and onion routing algorithm. In *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 364–370.

[62] Patanapongpibul, L., Mapp, G., and Hopper, A. (2006). An end-system approach to mobility management for 4g networks and its application to thin-client computing. *SIGMOBILE Mob. Comput. Commun. Rev.*, 10(3):13–33.

[63] Putra, J. P., Nugroho, S. M. S., and Pratomo, I. (2017). Live migration based on cloud computing to increase load balancing. In *2017 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, pages 286–290.

[64] Ramirez, J., Ezenwigbo, O. A., Karthick, G., Trestian, R., and Mapp, G. (2020). A new service management framework for vehicular networks. In *2020 23rd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*, pages 162–164.

[65] Ren, J., Guo, H., Xu, C., and Zhang, Y. (2017). Serving at the edge: A scalable iot architecture based on transparent computing. *IEEE Network*, 31(5):96–105.

[66] Rimal, B. P., Van, D. P., and Maier, M. (2017a). Mobile edge computing empowered fiber-wireless access networks in the 5g era. *IEEE Communications Magazine*, 55(2):192–200.

[67] Rimal, B. P., Van, D. P., and Maier, M. (2017b). Mobile edge computing empowered fiber-wireless access networks in the 5g era. *IEEE Communications Magazine*, 55(2):192–200.

[68] SAEInternational (2010). Dsrc implementation guide - a guide to users of sae j2735message sets over dsrc.

[69] Sardellitti, S., Scutari, G., and Barbarossa, S. (2014). Distributed joint optimization of radio and computational resources for mobile cloud computing. In *2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)*, pages 211–216.

[70] Sardellitti, S., Scutari, G., and Barbarossa, S. (2015). Joint optimization of radio and computational resources for multicell mobile-edge computing. *IEEE Transactions on Signal and Information Processing over Networks*, 1(2):89–103.

[71] Sardis, F. (2014a). *Exploring Traffic and QoS Management Mechanisms to Support Mobile Cloud Computing using Service Localisation in Heterogeneous Environments*. PhD thesis.

[72] Sardis, F. (2014b). *Exploring Traffic and QoS Management mechanisms to support mobile cloud computing using service localization in heterogeneous environments*. School of Science and Technology, Middlesex University. PhD Thesis.

[73] Sardis, F. (2015). *Exploring traffic and QoS management mechanisms to support mobile cloud computing using service localisation in heterogeneous environments*. PhD thesis, Middlesex University.

[74] Sassi, A., Charfi, F., Kamoun, L., Elhillali, Y., and Rivenq, A. (2015). Experimental measurement for vehicular communication evaluation using obu arada system. In *2015 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1358–1364.

[75] Sengole Merlin, S., Arunkumar, N. M., and Angela, M. A. (2018). Automated intelligent systems for secure live migration. In *2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT)*, pages 1360–1371.

[76] Shaikh, F., Lasebae, A., and Mapp, G. (2008). Proactive policy management for heterogeneous networks. In *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, pages 1–6.

[77] SlideShare (2015). Understanding DPDK.

[78] Sun, X. and Ansari, N. (2016). Edgeiot: Mobile edge computing for the internet of things. *IEEE Communications Magazine*, 54(12):22–29.

[79] Sun, Z., Liu, J., Li, Z., Wang, T., Wang, Z., Jia, F., and Lai, C. (2020). Csr-im: Compressed sensing routing-control- method with intelligent migration-mechanism based on sensing cloud-computing. *IEEE Access*, 8:28437–28449.

[80] Sztrik, J. (2012, 2021). *Basic Queueing Theory*. Sztrik online.

[81] Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., and Sabella, D. (2017). On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys Tutorials*, 19(3):1657–1681.

[82] Tan, D., Liu, F., Xiao, N., and Xie, Y. (2019). Optimizing virtual machine live migration in distributed edge servers based on hybrid memory. In *2019 International Conference on High Performance Big Data and Intelligent Systems (HPBD IS)*, pages 29–34.

[83] Tang, L. (2018). Dynamic migration optimization algorithm for virtual machine under trusted computing environment. In *2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS)*, pages 48–52.

[84] Toutov, A. V., Vorozhtsov, A. S., and Toutova, N. V. (2018). Estimation of total migration time of virtual machines in cloud data centers. In *2018 IEEE International Conference "Quality Management, Transport and Information Security, Information Technologies" (IT QM IS)*, pages 389–393.

[85] Tran, T. X., Hajisami, A., Pandey, P., and Pompili, D. (2017). Collaborative mobile edge computing in 5g networks: New paradigms, scenarios, and challenges. *IEEE Communications Magazine*, 55(4):54–61.

[86] u. R. Khan, A., Othman, M., Madani, S. A., and Khan, S. U. (2014). A survey of mobile cloud computing application models. *IEEE Communications Surveys Tutorials*, 16(1):393–413.

[87] Ullah, A., Yao, X., Shaheen, S., and Ning, H. (2020). Advances in position based routing towards its enabled fog-oriented vanet–a survey. *IEEE Transactions on Intelligent Transportation Systems*, 21(2):828–840.

[88] Vardhan, P. V. (2018). *Exploiting User Contention toOptimize Proactive ResourceAllocation in Future Networks*. PhD thesis, Middlesex University.

[89] Wan, T. and Ashwood, P. (2015). A performance study of CPRI over Ethernet. *IEEE1904*, 3.

[90] Wang, D., Tian, X., Cui, H., and Liu, Z. (2020). Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware mec network. *China Communications*, 17(8):31–44.

[91] Xu, B., Wu, S., Xiao, J., Jin, H., Zhang, Y., Shi, G., Lin, T., Rao, J., Yi, L., and Jiang, J. (2020). Sledge: Towards efficient live migration of docker containers. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*, pages 321–328.

[92] Yang, B., Li, J., and Li, Y. (2019). Research on qos-oriented virtual machine migration strategy in mobile edge computing. In *2019 12th International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pages 227–231.

[93] Yingchi, M., Ziyang, X., Longbao, W., and Jiulong, W. (2015). An optimal web services migration framework in the cloud computing. In *2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pages 153–156.

[94] Yu, Y. (2016). Mobile edge computing towards 5g: Vision, recent progress, and open challenges. *China Communications*, 13(Supplement2):89–99.

[95] Zhang, K., Mao, Y., Leng, S., Vinel, A., and Zhang, Y. (2016a). Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks. In *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, pages 288–294.

[96] Zhang, K., Mao, Y., Leng, S., Zhao, Q., Li, L., Peng, X., Pan, L., Maharjan, S., and Zhang, Y. (2016b). Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks. *IEEE Access*, 4:5896–5907.

[97] Zhang, S., Wang, L., and Han, X. (2014). A kvm virtual machine memory forensics method based on vmcs. In *2014 Tenth International Conference on Computational Intelligence and Security*, pages 657–661.

[98] Zhao, Y., Zhou, S., Zhao, T., and Niu, Z. (2015). Energy-efficient task offloading for multiuser mobile cloud computing. In *2015 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 1–5.

[99] Zou, H., Li, M., Li, Z., and Gao, J. (2018). Design of multi-intelligent data migration strategy based on sdn secondary mode. In *2018 International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 76–81.

Appendix A

The Server Migration Results

A.1 Reactive Handover Results for 30MPH

Where:

K = 100 ; lambda = 25; mu = 50

N = Resource Hold Time for different RSUs

For 30 mph and 50 mph

RSU No.	**30 Mph**	**50 Mph**
	N	*N*
RSU 1	18.37 s	9.42 s
RSU 2	30.00 s	16.40 s
RSU 3	34.55 s	19.13 s
RSU 4	14.49 s	7.09 s
RSU 5	68.63 s	39.57 s
RSU 6	99.64 s	58.19 s
RSU 7	81.00 s	47.00 s

Migration Results

Technology	**Time**
Uni-kernel	11.99
LXD CRIU	24.73
Docker	73.00
KVM	824.00

Result Codes

Table A.1 **Uni-kernel** Reactive for 30 MPH

Uni-kernel	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	54.1623	2.1665	40.7124	53.5342
RSU2->RSU3	35.6863	1.4275	26.1052	35.2143
RSU3->RSU4	31.5495	1.2620	22.9275	31.0569
RSU5->RSU6	17.1508	0.6860	12.0263	16.4823
RSU6->RSU7	12.3054	0.4922	8.4009	11.5503

Table A.2 **LXD-CRIU** Reactive for 30 MPH

LXD-CRIU	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	NP1			
RSU2->RSU3	73.3825	2.9353	63.8911	72.6958
RSU3->RSU4	NP2			
RSU5->RSU6	34.6469	1.3859	29.2422	34.0240
RSU6->RSU7	24.5609	0.9824	20.4439	23.8431

- **NP1 :** Service migration cannot occur because you cannot migrate anything **from t**he first network because SHT <0 in the first network
- **NP2:** Service migration cannot occur because you cannot migrate anything **to** the second network because SHT <0 on the second network

Table A.3 **Docker** Reactive for 30 MPH

Docker	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	NP2			
RSU2->RSU3	NP2			
RSU3->RSU4	NP2			
RSU5->RSU6	NP1			
RSU6->RSU7	70.7721	2.8309	66.9771	70.4191

Table A.4 **KVM** Reactive for 30 MPH

KVM	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	NP2			
RSU2->RSU3	NP2			
RSU3->RSU4	NP2			
RSU5->RSU6	NP2			
RSU6->RSU7	NP2			

A.2 Proactive Handover Results for 30MPH

Where:

K = 100 ; lambda = 25; mu = 50

N = Resource Hold Time for different RSUs

For 30 mph and 50 mph

RSU No.	**30 Mph**	**50 Mph**
	N	*N*
RSU 1	18.37 s	9.42 s
RSU 2	30.00 s	16.40 s
RSU 3	34.55 s	19.13 s
RSU 4	14.49 s	7.09 s
RSU 5	68.63 s	39.57 s
RSU 6	99.64 s	58.19 s
RSU 7	81.00 s	47.00 s

Migration Results

Technology	**Time**
Uni-kernel	11.99
LXD CRIU	24.73
Docker	73.00
KVM	824.00

A.2.1 Result Codes

- **NP1:** Service migration cannot occur because you cannot migrate anything **from t**he first network because SHT $<$ 0 in the first network

Table A.5 **Uni-kernel** Service Migration Time = 11.99 Xmin = 0 Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	0	54.1623	2.1665	40.7124	53.5342
RSU1->RSU2	1	49.6052	1.9842	36.2207	49.0618
RSU1->RSU2	2	45.0473	1.8019	31.7901	44.5897
RSU1->RSU2	3	40.4837	1.6193	27.4237	40.1177
RSU1->RSU2	4	35.9097	1.4369	23.1307	35.6460
RSU1->RSU2	5	31.3185	1.2527	18.9272	31.1742
RSU1->RSU2	6	26.7025	1.0681	14.8412	26.7027
RSU1->RSU2	7	22.0500	0.8820	10.9198	22.2310
RSU1->RSU2	7.5	19.7057	0.7882	9.0447	19.9951
RSU1->RSU2	7.99	17.3938	0.6958	7.2814	17.8040

Table A.6 **LXD-CRIU** Service Migration Time = 24.73, Resource Hold Time RSU1 = 18.37, Resource Hold Time RSU2 = 30.0

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	1	NP1			
RSU1->RSU2	2	NP1			
RSU1->RSU2	2.5	99.9348	3.9974	98.7483	99.2742
RSU1->RSU2	3	98.9611	3.9584	94.2897	97.1387
RSU1->RSU2	4	98.8462	3.7938	86.1511	92.6588
RSU1->RSU2	6	85.2868	3.4115	73.5462	83.6948
RSU1->RSU2	9	71.2783	2.8511	58.1437	70.2649
RSU1->RSU2	12	57.5364	2.3015	44.0761	56.8439
RSU1->RSU2	15	43.8614	1.7545	30.6483	43.4269
RSU1->RSU2	18	30.1211	1.2048	17.8520	30.0116
RSU1->RSU2	18.37	28.4141	1.1366	16.3370	28.3572

- **NP2:** Service migration cannot occur because you cannot migrate anything **to** the second network because SHT $<$ 0 on the second network
- **NP3:** Xmax $<$ Xmin This maximum value of X is less that the minimum required value

Table A.7 **Docker** Service Migration Time = 73.00, Resource Hold Time RSU1 18.37, Resource Hold Time RSU2 30.00

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	0	NP3			
RSU1->RSU2	1	NP3			
RSU1->RSU2	2	NP3			
RSU1->RSU2	3	NP3			
RSU1->RSU2	4	NP3			
RSU1->RSU2	6	NP3			
RSU1->RSU2	9	NP3			
RSU1->RSU2	12	NP3			
RSU1->RSU2	15	NP3			
RSU1->RSU2	18	NP3			
RSU1->RSU2	18.37	NP3			

Table A.8 **KVM** Service Migration Time = 824.00, Resource Hold Time RSU1 18.37, Resource Hold Time RSU2 30.00, Xmin = 790.0, Xmax = 18.37

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	0	NP3			
RSU1->RSU2	1	NP3			
RSU1->RSU2	2	NP3			
RSU1->RSU2	3	NP3			
RSU1->RSU2	4	NP3			
RSU1->RSU2	6	NP3			
RSU1->RSU2	9	NP3			
RSU1->RSU2	12	NP3			
RSU1->RSU2	15	NP3			
RSU1->RSU2	18	NP3			
RSU1->RSU2	18.37	NP3			

Table A.9 **RSU2 -> RSU3 Unikernel**, Service Migration Time = 11.99, Resource Hold Time RSU1 30.00, Resource Hold Time RSU2 34.55

Unikernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU2->RSU3	0	35.6863	1.4275	26.1052	35.2143
RSU2->RSU3	1	32.7217	1,3089	23.2513	32.2729
RSU2->RSU3	2	29.7510	1,1900	20.4219	29.3312
RSU2->RSU3	3	26.7730	1,0709	17.6233	26.3896
RSU2->RSU3	4	23.7851	0.9514	14.8643	23.4481
RSU2->RSU3	5	20.7837	0.8313	12.1579	20.5065
RSU2->RSU3	6	17.7647	0.7106	9.5246	17.5650
RSU2->RSU3	7	14.7212	0.5888	6.9971	14.6234
RSU2->RSU3	7.5	13.1878	0.5275	5.7892	13.1525
RSU2->RSU3	7.99	11.6761	0.4670	4.6543	11.7111

Table A.10 **RSU2 -> RSU3 LXD-CRIU**, Service Migration Time = 24.73, Resource Hold Time RSU1 30.00, Resource Hold Time RSU2 34.55, Xmin = 0, Xmax = 20.73

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU2->RSU3	0	73.3825	2.9353	63.8911	72.6958
RSU2->RSU3	2	67.4387	2.6975	57.7659	66.8103
RSU2->RSU3	4	61.5160	2.4606	51.7366	60.9258
RSU2->RSU3	6	55.6047	2.2242	45.7761	55.0416
RSU2->RSU3	9	46.7454	1.8698	36.9405	46.2162
RSU2->RSU3	12	37.8776	1.5151	28.2307	.3912
RSU2->RSU3	15	28.9778	1.1591	19.6910	28.5663
RSU2->RSU3	18	20.0001	0.8000	11.4652	19.7417
RSU2->RSU3	20.0	13.9249	0.5570	6.3644	13.8586
RSU2->RSU3	20.73	11.6761	0.4670	4.6543	11.7112

Table A.11 **RSU2 -> RSU3 Docker**, Service Migration Time = 73.00, Resource Hold Time RSU1 30.00, Resource Hold Time RSU2 34.55, Xmin = 34.45, Xmax = 30.00

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU2->RSU3	0	NP3			
RSU2->RSU3	2	NP3			
RSU2->RSU3	4	NP3			
RSU2->RSU3	6	NP3			
RSU2->RSU3	9	NP3			
RSU2->RSU3	12	NP3			
RSU2->RSU3	15	NP3			
RSU2->RSU3	18	NP3			
RSU2->RSU3	20.0	NP3			
RSU2->RSU3	20.73	NP3			

Table A.12 **RSU2 -> RSU3 KVM**, Service Migration Time = 824.00, Resource Hold Time RSU1 30.00, Resource Hold Time RSU2 34.55, Xmin = 785.45, Xmax = 30.00

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU2->RSU3	0	NP3			
RSU2->RSU3	2	NP3			
RSU2->RSU3	4	NP3			
RSU2->RSU3	6	NP3			
RSU2->RSU3	9	NP3			
RSU2->RSU3	12	NP3			
RSU2->RSU3	15	NP3			
RSU2->RSU3	18	NP3			
RSU2->RSU3	20.0	NP3			
RSU2->RSU3	20.73	NP3			

Table A.13 **RSU3 ->** **RSU4**, **Unikernel** Service Migration Time = 11.99, Resource Hold Time RSU1 34.55, Resource Hold Time RSU2 14.49, Xmin = 0, Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU3->RSU4	0	31.5495	1.2620	22.9275	31.0569
RSU3->RSU4	1	28.9380	1.1575	20.4208	28.4627
RSU3->RSU4	2	26.3210	1.0528	17.9350	25.8683
RSU3->RSU4	3	23.6972	0.9479	15.4757	23.2739
RSU3->RSU4	4	21.0645	0.8426	13.0510	20.6797
RSU3->RSU4	5	18.4200	0.7368	10.6724	18.0854
RSU3->RSU4	6	15.7598	0.6304	8.3582	15.4912
RSU3->RSU4	7	13.0781	0.5231	6.1374	12.8968
RSU3->RSU4	7.5	11.7272	0.4691	5.0765	11.5996
RSU3->RSU4	7.99	10.3954	0.4158	4.0799	10.3284

Table A.14 **RSU3 ->** **RSU4**, **LXD-CRIU**, Service Migration Time = 24.73, Resource Hold Time RSU1 34.55, Resource Hold Time RSU2 14.49, Xmin = 6.24, Xmax = 20.7

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU3->RSU4	6.25	48.4314	1.9373	39.5182	47.8942
RSU3->RSU4	7.00	46.4837	1.8593	37.5798	45.9485
RSU3->RSU4	9.00	41.2870	1.6515	32.4341	40.7597
RSU3->RSU4	12.00	33.4793	1.3392	24.7937	32.9767
RSU3->RSU4	15.00	25.6395	1.0256	17.2927	25.1937
RSU3->RSU4	18.0	17.7300	0.7092	10.0635	17.4108
RSU3->RSU4	20	12.3766	0.4951	5.5808	12.2227
RSU3->RSU4	20.5	11.0215	0.4409	4.5417	10.9251
RSU3->RSU4	20.73	10.3954	0.4158	4.0780	10.3284

Table A.15 **RSU3 ->** **RSU4**, **Docker**, Service Migration Time = 73.00, Resource Hold Time RSU3 34.55,Resource Hold Time RSU4 14.49 Xmin = 54.51, Xmax = 34.549999

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4.	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				

Table A.16 **RSU3 -> RSU4**, **KVM**, Service Migration Time = 824.00, Resource Hold Time RSU1 34.55, Resource Hold Time RSU2 14.49 Xmin = 805.51, Xmax = 34.549999

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4.	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				
RSU3->RSU4	NP3				

Table A.17 **RSU5->RSU6 Unikernel**, Service Migration Time = 11.99, Resource Hold Time RSU5 = 68.63, Resource Hold Time RSU6 = 99.64, Xmin = 0, Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU5->RSU6	0	17.1501	0.6860	12.0263	16.4823
RSU5->RSU6	1	15.7675	0.6307	10.7080	15.1055
RSU5->RSU6	2	14.3819	0.5753	9.4004	13.7286
RSU5->RSU6	3	12.9924	0.5197	8.1067	12.3517
RSU5->RSU6	4	11.5893	0.4640	6.8315	10.9749
RSU5->RSU6	5	10.1980	0.4080	5.5810	9.5980
RSU5->RSU6	6	8.7894	0.3516	4.3651	8.2211
RSU5->RSU6	7	7.3699	0.2948	3.1997	6.8442
RSU5->RSU6	7.5	6.6550	0.2662	2.6436	6.1557
RSU5->RSU6	7.99	5.9506	0.2380	2.1218	5.4810

Table A.18 **RSU5->RSU6 LXD-CRIU**, Service Migration Time = 24.73, Resource Hold Time RSU5 68.63, Resource Hold Time RSU6 99.64, Xmin = 0, Xmax = 20.73

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU5->RSU6	0	34.6469	1.3859	29.2422	34.0240
RSU5->RSU6	2	31.9074	1.2763	26.5122	31.2700
RSU5->RSU6	4	29.1665	1.1667	23.7888	28.5162
RSU5->RSU6	6	26.4235	1.0569	21.0735	25.7625
RSU5->RSU6	9	22.3031	0.8921	17.0214	21.6319
RSU5->RSU6	12	18.1716	0.7269	13.0076	17.5012
RSU5->RSU6	15	14.021	0.5608	9.0625	13.3706
RSU5->RSU6	18	9.8350	0.3934	5.2624	9.2430
RSU5->RSU6	20	6.9987	0.2799	2.9079	6.4862
RSU5->RSU6	20.5	6.2818	0.2513	2.3635	5.7978
RSU5->RSU6	20.73	5.9506	0.2380	2.1283	5.4811

Table A.19 **RSU5->RSU6 Docker**, Service Migration Time = 73.00, Resource Hold Time RSU5 68.63, Resource Hold Time RSU6 99.64, Xmin = 0.00, Xmax = 68.29997

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU5->RSU6	0	NP1			
RSU5->RSU6	5	94.1356	3.7654	90.0643	93.6131
RSU5->RSU6	10	87.1125	3.4845	82.4261	86.7224
RSU5->RSU6	15	80.2056	3.2082	75.2651	79.8355
RSU5->RSU6	25	66.4858	2.6594	61.2852	66.0646
RSU5->RSU6	35	52.8027	2.1121	47.4595	52.2952
RSU5->RSU6	45	39.1228	1.5649	33.7161	38.5262
RSU5->RSU6	55	25.4217	1.0169	20.0850	24.7575
RSU5->RSU6	60	18.5441	0.7418	13.3667	17.8730
RSU5->RSU6	65	11.6123	0.4645	6.8441	10.9886
RSU5->RSU6	68.299	6.9572	0.2783	2.8757	6.4462

Table A.20 **RSU5->RSU6 KVM**, Service Migration Time = 824.00, Resource Hold Time RSU5 68.63, Resource Hold Time RSU6 99.64, Xmin = 720.36, Xmax = 68.629997

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU5->RSU6	NP3				
RSU5->RSU6	NP3				
RSU5->RSU6.	NP3				
RSU5->RSU6	NP3				
RSU5->RSU6	NP3				
RSU5->RSU6	NP3				
RSU5->RSU6	NP3				
RSU5->RSU6	NP3				
RSU5->RSU6	NP3				
RSU5->RSU6	NP3				

Table A.21 **RSU6->RSU7 Unikernel**, Service Migration Time = 11.99, Resource Hold Time RSU6 = 99.64, Resource Hold Time RSU7 = 81.00, Xmin = 0, Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU6->RSU7	0	12.3054	0.4922	8.4010	11.5503
RSU6->RSU7	1	11.3369	0.4535	7.4787	10.5855
RSU6->RSU7	2	10.3662	0.4146	6.5641	9.6206
RSU6->RSU7	3	9.3930	0.3757	5.6594	8.6557
RSU6->RSU7	4	8.4165	0.3367	4.7677	7.6908
RSU6->RSU7	5	7.4356	0.2974	3.8934	6.7259
RSU6->RSU7	6	6.4491	0.2579	3.0438	5.7610
RSU6->RSU7	7	5.4550	0.2182	2.2300	4.7961
RSU6->RSU7	7.5	4.9544	0.1982	1.8414	4.3137
RSU6->RSU7	7.99	4.4613	0.1785	1.4773	3.8409

Table A.22 **RSU6->RSU7 LXD-CRIU**, Service Migration Time = 24.73, Resource Hold Time RSU6 99.64, Resource Hold Time RSU7 81.00, Xmin = 0, Xmax = 20.73

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU6->RSU7	0	24.5609	0.9824	20.4439	23.8432
RSU6->RSU7	2	22.6423	0.9057	18.5345	21.9133
RSU6->RSU7	4	20.7225	0.8290	16.6300	19.9838
RSU6->RSU7	6	18.8011	0.7520	14.7300	18.0536
RSU6->RSU7	9	15.9151	0.6366	11.8954	15.1591
RSU6->RSU7	12	13.0210	0.5208	9.0874	12.2644
RSU6->RSU7	15	10.1134	0.4045	6.3278	9.3696
RSU6->RSU7	18	7.1797	0.2872	3.6698	6.4750
RSU6->RSU7	20	5.1950	0.2078	2.0259	4.5452
RSU6->RSU7	20.5	4.6930	0.1877	1.6459	4.0628
RSU6->RSU7	20.73	4.4613	0.1785	1.4773	3.8409

Table A.23 **RSU6->RSU7 Docker**, Service Migration Time = 73.0, Resource Hold Time RSU5 99.64, Resource Hold Time RSU6 81.00, Xmin = 0, Xmax = 69.00

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU6->RSU7	0	70.7721	2.8309	66.9771	70.4191
RSU6->RSU7	5	65.9839	2.6394	62.1289	65.5944
RSU6->RSU7	10	61.1973	2.4479	57.2889	60.7696
RSU6->RSU7	15	56.4124	2.2565	52.4557	55.9453
RSU6->RSU7	25	46.8429	1.8737	42.8038	46.2959
RSU6->RSU7	35	37.2726	1.4909	33.1731	36.6472
RSU6->RSU7	45	27.6954	1.1078	23.5721	26.9982
RSU6->RSU7	55	18.0995	0.7240	14.0386	17.3494
RSU6->RSU7	60	13.2819	0.5313	9.03396	12.5248
RSU6->RSU7	65	8.4262	0.3370	4.7765	7.70044
RSU6->RSU7	69	4.4613	0.1785	1.4773	3.84089

Table A.24 **RSU6->RSU7 KVM**, Service Migration Time = 824.00, Resource Hold Time RSU5 99.64, Resource Hold Time RSU6 81.00, Xmin = 739.0 Xmax = 99.639999

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU6->RSU7	NP3				
RSU6->RSU7	NP3				
RSU6->RSU7.	NP3				
RSU6->RSU7	NP3				
RSU6->RSU7	NP3				
RSU6->RSU7	NP3				
RSU6->RSU7	NP3				
RSU6->RSU7	NP3				
RSU6->RSU7	NP3				
RSU6->RSU7	NP3				

A.3 Reactive Handover Results for 50MPH

Where:

K = 100 ; lambda = 25; mu = 50

N = Resource Hold Time for different RSUs

For 30 mph and 50 mph

RSU No.	**30 Mph**	**50 Mph**
	N	*N*
RSU 1	18.37 s	9.42 s
RSU 2	30.00 s	16.40 s
RSU 3	34.55 s	19.13 s
RSU 4	14.49 s	7.09 s
RSU 5	68.63 s	39.57 s
RSU 6	99.64 s	58.19 s
RSU 7	81.00 s	47.00 s

Migration Results

Technology	**Time**
Uni-kernel	11.99
LXD CRIU	24.73
Docker	73.00
KVM	824.00

Result Codes

- **NP1 :** Service migration cannot occur because you cannot migrate anything **from t**he first network because SHT <0 in the first network
- **NP2:** Service migration cannot occur because you cannot migrate anything **to** the second network because SHT <0 on the second network

Table A.25 **Uni-kernel** Reactive for 50 MPH

Uni-kernel	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	92.8323	3.7133	79.6230	89.3312
RSU2->RSU3	59.4877	2.3795	45.1153	58.7092
RSU3->RSU4	NP2			
RSU5->RSU6	27.9998	1.1200	20.2193	27.4778
RSU6->RSU7	19.8738	0.7950	14.0727	19.2496

Table A.26 **LXD-CRIU** Reactive for 50 MPH

LXD-CRIU	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	NP2			
RSU2->RSU3	NP2			
RSU3->RSU4	NP2			
RSU5->RSU6	57.2495	2.2900	49.1825	56.7216
RSU6->RSU7	40.3166	1.6127	34.2081	39.7363

Table A.27 **Docker** Reactive for 50 MPH

Docker	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	NP2			
RSU2->RSU3	NP2			
RSU3->RSU4	NP2			
RSU5->RSU6	NP2			
RSU6->RSU7	NP2			

Table A.28 **KVM** Reactive for 50 MPH

KVM	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	NP2			
RSU2->RSU3	NP2			
RSU3->RSU4	NP2			
RSU5->RSU6	NP2			
RSU6->RSU7	NP2			

A.4 Proactive Handover Results for 50MPH

Where:

K = 100 ; lambda = 25; mu = 50

N = Resource Hold Time for different RSUs

For 30 mph and 50 mph

RSU No.	30 Mph *N*	50 Mph *N*
RSU 1	18.37 s	9.42 s
RSU 2	30.00 s	16.40 s
RSU 3	34.55 s	19.13 s
RSU 4	14.49 s	7.09 s
RSU 5	68.63 s	39.57 s
RSU 6	99.64 s	58.19 s
RSU 7	81.00 s	47.00 s

Migration Results

Technology	Time
Uni-kernel	11.99
LXD CRIU	24.73
Docker	73.00
KVM	824.00

A.4.1 Result Codes

- **NP1:** Service migration cannot occur because you cannot migrate anything **from t**he first network because SHT $<$ 0 in the first network
- **NP2:** Service migration cannot occur because you cannot migrate anything **to** the second network because SHT $<$ 0 on the second network
- **NP3:** Xmax $<$ Xmin This maximum value of X is less that the minimum required value

Table A.29 **RSU1->RSU2 Unikernel**, Service Migration Time = 11.99, Resource Hold Time RSU 1 9.42, Resource Hold Time RSU2 16.40, Xmin = 0, Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	0	92.8323	3.7133	79.6230	89.3312
RSU1->RSU2	1	84.7357	3.3894	68.2152	81.8564
RSU1->RSU2	2	76.5878	3.0635	58.4985	74.3853
RSU1->RSU2	3	68.5519	2.7421	49.7294	66.9188
RSU1->RSU2	4	60.6029	2.4241	41.5513	59.4556
RSU1->RSU2	5	52.7001	2.1080	33.8012	51.9948
RSU1->RSU2	6	44.8063	1.7923	26.4205	44.5356
RSU1->RSU2	7	36.8846	1.4754	19.4250	37.0773
RSU1->RSU2	7.5	32.9009	1.3160	16.0984	33.3483
RSU1->RSU2	7.99	28.9753	1.1590	12.9760	29.6940

Table A.30 **RSU1->RSU2 LXD-CRIU**, Service Migration Time = 24.73, Resource Hold Time RSU1 9.42, Resource Hold Time RSU2 16.40, Xmin = 4.33, Xmax = 9.42

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	NP1				
RSU1->RSU2	NP1				
RSU1->RSU2	NP1				
RSU1->RSU2	NP1				
RSU1->RSU2	NP1				
RSU1->RSU2	NP1				
RSU1->RSU2	NP1				
RSU1->RSU2	NP1				

Table A.31 **RSU1->RSU2 Docker** Service Migration Time = 73.00, Resource Hold Time RSU1 9.42, Resource Hold Time RSU2 16.40, Xmin = 52.6, Xmax = 9.42

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	0	NP3			
RSU1->RSU2	1	NP3			
RSU1->RSU2	2	NP3			
RSU1->RSU2	3	NP3			
RSU1->RSU2	4	NP3			
RSU1->RSU2	6	NP3			
RSU1->RSU2	9	NP3			
RSU1->RSU2	12	NP3			
RSU1->RSU2	15	NP3			
RSU1->RSU2	18	NP3			
RSU1->RSU2	18.37	NP3			

Table A.32 **RSU1->RSU2 KVM** Service Migration Time = 824.00, Resource Hold Time RSU1 9.42, Resource Hold Time RSU2 16.40, Xmin = 803.6, Xmax = 9.42

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU1->RSU2	0	NP3			
RSU1->RSU2	1	NP3			
RSU1->RSU2	2	NP3			
RSU1->RSU2	3	NP3			
RSU1->RSU2	4	NP3			
RSU1->RSU2	6	NP3			
RSU1->RSU2	9	NP3			
RSU1->RSU2	12	NP3			
RSU1->RSU2	15	NP3			
RSU1->RSU2	18	NP3			
RSU1->RSU2	18.37	NP3			

Table A.33 **RSU2->RSU3 Unikernel** Service Migration Time = 11.99, RSU2 16.40, RSU3 19.13, Xmin = 0, Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU2->RSU3	0	59.4877	2.3795	45.1153	58.7092
RSU2->RSU3	1	54.4608	2.1784	40.0981	53.8041
RSU2->RSU3	2	49.4388	1.9776	35.1681	48.8994
RSU2->RSU3	3	44.4153	1.7766	30.3229	43.9950
RSU2->RSU3	4	39.3836	1.5753	25.5684	39.0910
RSU2->RSU3	5	34.3355	1.3734	20.9192	34.1870
RSU2->RSU3	6	29.2619	1.1705	16.4041	29.2833
RSU2->RSU3	7	24.1492	0.9660	12.0730	24.3795
RSU2->RSU3	7.5	21.5732	0.8629	10.0023	21.9276
RSU3->RSU3	7.99	19.0328	0.76131	8.05480	19.5247

Table A.34 **RSU2->RSU3 LXD-CRIU** Service Migration Time = 24.73, RSU2 16.40, RSU3 19.13, Xmin = 1.60, Xmax = 16.4

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU2->RSU3	2.0	NP1			
RSU2->RSU3	4.0	NP1			
RSU2->RSU3	6.0	94.1721	3.7669	84.5833	91.8022
RSU2->RSU3	8.0	83.6588	3.3464	70.8672	81.9726
RSU2->RSU3	10.0	73.3509	2.9340	59.4401	72.1531
RSU2->RSU3	12.0	63.2150	2.5286	48.8901	62.3394
RSU2->RSU3	14.0	53.1549	2.1262	38.8082	52.5288
RSU2->RSU3	15.0	48.1330	1.9253	33.9000	47.6241
RSU2->RSU3	16.0	43.1081	1.7243	29.0776	42.7199
RSU2->RSU3	16.39	41.1462	1.6458	27.2216	40.8075

Table A.35 **RSU2->RSU3 Docker** Service Migration Time = 73.00, RSU2 16.40, RSU3 19.13, Xmin = 49.87, Xmax = 16.4

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU2->RSU3	0	NP3			
RSU2->RSU3	1	NP3			
RSU2->RSU3	2	NP3			
RSU2->RSU3	3	NP3			
RSU2->RSU3	4	NP3			
RSU2->RSU3	6	NP3			
RSU2->RSU3	9	NP3			
RSU2->RSU3	12	NP3			
RSU2->RSU3	15	NP3			
RSU2->RSU3	18	NP3			
RSU2->RSU3	18.37	NP3			

Table A.36 **RSU2->RSU3 KVM** Service Migration Time = 824.00, RSU2 16.40, RSU3 19.13, Xmin = 800.87, Xmax = 16.4

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU2->RSU3	0	NP3			
RSU2->RSU3	1	NP3			
RSU2->RSU3	2	NP3			
RSU2->RSU3	3	NP3			
RSU2->RSU3	4	NP3			
RSU2->RSU3	6	NP3			
RSU2->RSU3	9	NP3			
RSU2->RSU3	12	NP3			
RSU2->RSU3	15	NP3			
RSU2->RSU3	18	NP3			
RSU2->RSU3	18.37	NP3			

Table A.37 **RSU3->RSU4 Unikernel** Service Migration Time = 11.99, RSU3 19.13, RSU4 7.09, Xmin = 0.9, Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU3->RSU4	1	47.9640	1.9186	34.9284	47.4485
RSU3->RSU4	2	43.5617	1.7425	30.6612	43.1235
RSU3->RSU4	3	39.1530	1.5661	26.4529	38.7986
RSU3->RSU4	4	34.7334	1.3893	22.3133	34.4740
RSU3->RSU4	5	30.2968	1.2119	18.2586	30.1493
RSU3->RSU4	6	25.8356	1.0334	14.3163	25.8247
RSU3->RSU4	7	21.3390	0.8536	10.5324	21.5000
RSU3->RSU4	7.5	19.0732	0.7629	8.7231	19.3377
RSU3->RSU4	7.99	16.8389	0.6736	7.0217	17.2186

Table A.38 **RSU3->RSU4 LXD-CRIU** Service Migration Time = 24.73, RSU3 19.13, RSU4 7.09, Xmin = 13.64, Xmax = 19.13

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU3->RSU4	14.0	46.8199	1.8728	33.8137	46.3241
RSU3->RSU4	15.0	42.4161	1.6966	29.5610	41.1999
RSU3->RSU4	16.0	38.0051	1.5202	25.3694	37.6741
RSU3->RSU4	17.0	33.5818	1.3433	21.2500	33.3495
RSU3->RSU4	18.0	29.1395	1.1656	17.2212	29.0248
RSU3->RSU4	19.0	24.6703	0.9868	13.3147	24.7002
RSU3->RSU4	19.1	24.2215	0.9689	12.9326	24.2678

Table A.39 **RSU3->RSU4 Docker** Service Migration Time = 73.00, RSU3 19.13, RSU4 7.09, Xmin = 61.91, Xmax = 19.13

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU3->RSU4	0	NP3			
RSU3->RSU4	1	NP3			
RSU3->RSU4	2	NP3			
RSU3->RSU4	3	NP3			
RSU3->RSU4	4	NP3			
RSU3->RSU4	6	NP3			
RSU3->RSU4	9	NP3			
RSU3->RSU4	12	NP3			
RSU3->RSU4	15	NP3			
RSU3->RSU4	18	NP3			
RSU3->RSU4	18.37	NP3			

Table A.40 **RSU3->RSU4 KVM** Service Migration Time = 824.00, RSU3 19.13, RSU4 7.09, Xmin = 812.91, Xmax = 19.13

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU3->RSU4	0	NP3			
RSU3->RSU4	1	NP3			
RSU3->RSU4	2	NP3			
RSU3->RSU4	3	NP3			
RSU3->RSU4	4	NP3			
RSU3->RSU4	6	NP3			
RSU3->RSU4	9	NP3			
RSU3->RSU4	12	NP3			
RSU3->RSU4	15	NP3			
RSU3->RSU4	18	NP3			
RSU3->RSU4	18.37	NP3			

Table A.41 **RSU5->RSU6 Unikernel** Service Migration Time = 11.99, RSU5 39.57, RSU6 58.19, Xmin = 0.00, Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU5->RSU6	0	28.0000	1.1200	20.2193	27.4778
RSU5->RSU6	1	25.6912	1.0276	18.0079	25.1825
RSU5->RSU6	2	23.3777	0.9351	15.8147	22.8872
RSU5->RSU6	3	21.0579	0.8423	13.6446	20.5918
RSU5->RSU6	4	18.7302	0.7492	11.5050	18.2965
RSU5->RSU6	5	16.3920	0.6557	9.4061	16.0011
RSU5->RSU6	6	14.0400	0.5616	7.3643	13.7058
RSU5->RSU6	7	11.6691	0.4668	5.4055	11.4104
RSU5->RSU6	7.5	10.4748	0.4190	4.4699	10.2627
RSU5->RSU6	7.99	9.2976	0.3719	3.5912	9.1380

Table A.42 **RSU5->RSU6 LXD-CRIU** Service Migration Time = 24.73, RSU5 39.57, RSU6 58.19, Xmin = 0.000, Xmax = 20.73

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU5->RSU6	0.00	57.2498	2.2900	49.1825	56.7216
RSU5->RSU6	2.0	52.6633	2.1065	44.5703	52.1304
RSU5->RSU6	4.0	48.0777	1.9231	39.9793	47.5393
RSU5->RSU6	6.0	43.4915	1.7397	35.4100	42.9485
RSU5->RSU6	9.0	36.6063	1.4643	28.6016	36.0625
RSU5->RSU6	12.0	29.7052	1.1882	21.8654	29.1763
RSU5->RSU6	15.0	22.7751	0.9110	15.2480	22.2903
RSU5->RSU6	18.0	15.7820	0.6313	8.8869	15.4043
RSU5->RSU6	20.00	11.0490	0.4420	4.9146	10.8136
RSU5->RSU6	20.73	9.2976	0.3719	3.5912	9.1380

Table A.43 **RSU5->RSU6 Docker** Service Migration Time = 73.00, RSU5 39.57, RSU6 58.19, Xmin = 10.81, Xmax = 39.57

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU5->RSU6	11	NP1			
RSU5->RSU6	14	NP1			
RSU5->RSU6	18	NP1			
RSU5->RSU6	21	NP1			
RSU5->RSU6	24	NP1			
RSU5->RSU6	27	NP1			
RSU5->RSU6	30	99.5756	3.9830	97.3861	98.6917
RSU5->RSU6	32	95.1553	3.8062	89.5907	94.0906
RSU5->RSU6	34	90.3199	3.6128	83.6653	89.4909
RSU5->RSU6	36	85.5837	3.4233	78.4036	84.8948
RSU5->RSU6	38	80.9101	3.2364	73.4200	80.3008
RSU5->RSU6	39.57	77.2671	3.0907	69.6108	76.6953

Table A.44 **RSU5->RSU6 KVM** Service Migration Time = 824.00, RSU5 39.57, RSU6 58.19, Xmin = 761.810, Xmax = 39.57

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU5->RSU6	0	NP3			
RSU5->RSU6	1	NP3			
RSU5->RSU6	2	NP3			
RSU5->RSU6	3	NP3			
RSU5->RSU6	4	NP3			
RSU5->RSU6	6	NP3			
RSU5->RSU6	9	NP3			
RSU5->RSU6	12	NP3			
RSU5->RSU6	15	NP3			
RSU5->RSU6	18	NP3			
RSU5->RSU6	18.37	NP3			

Table A.45 **RSU6->RSU7 Unikernel** Service Migration Time = 11.99, RSU6 58.19, RSU7 47.00, Xmin = 0.00, Xmax = 7.99

Uni-kernel	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU6->RSU7	0	19.8738	0.7950	14.0727	19.249
RSU6->RSU7	1	18.2586	0.7303	12.5311	17.6416
RSU6->RSU7	2	16.6399	0.6656	11.0020	16.0336
RSU6->RSU7	3	15.0168	0.6007	9.4892	14.4256
RSU6->RSU7	4	13.3881	0.5355	7.9978	12.8175
RSU6->RSU7	5	11.7520	0.4701	6.5351	11.2095
RSU6->RSU7	6	10.1064	0.4043	5.1127	9.6014
RSU6->RSU7	7	8.4479	0.3379	3.7490	7.9933
RSU6->RSU7	7.5	7.6126	0.3045	3.0981	7.1893
RSU6->RSU7	7.99	6.7895	0.2716	2.4873	6.4014

Table A.46 **RSU6->RSU7 LXD-CRIU** Service Migration Time = 24.73, RSU6 58.19, RSU7 47.00, Xmin = 0.000, Xmax = 20.73

LXD-CRIU	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU6->RSU7	0.00	40.3166	1.6127	34.2081	39.7363
RSU6->RSU7	2.0	37.1155	1.4846	31.0140	36.5200
RSU6->RSU7	4.0	33.9128	1.3565	27.8282	33.3039
RSU6->RSU7	6.0	30.7078	1.2283	24.6523	30.0878
RSU6->RSU7	9.0	25.8939	1.0358	19.9136	25.2637
RSU6->RSU7	12.0	21.0673	0.8427	15.2201	20.4396
RSU6->RSU7	15.0	16.2183	0.6487	10.6069	15.6154
RSU6->RSU7	18.0	11.3252	0.4530	6.1608	10.7914
RSU6->RSU7	20.00	8.0141	0.3206	3.4074	7.5753
RSU6->RSU7	20.73	6.7895	0.2716	2.4873	6.4014

Table A.47 **RSU6->RSU7 Docker** Service Migration Time = 73.00, RSU6 58.19, RSU7 47.00, Xmin = 22.00, Xmax = 58.19

Docker	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU6->RSU7	22.5	81.6048	3.2642	76.0250	81.180
RSU6->RSU7	27.0	74.3627	2.9745	68.5783	73.9413
RSU6->RSU7	32.0	66.3453	2.6538	60.4181	65.8999
RSU6->RSU7	37.0	58.3427	2.3337	52.3199	57.8591
RSU6->RSU7	42.0	50.3457	2.0138	44.2623	49.8187
RSU6->RSU7	47.0	42.3484	1.6939	36.2397	41.7784
RSU6->RSU7	52.0	34.3452	1.3738	28.2577	33.7380
RSU6->RSU7	57.0	26.3275	1.0531	20.3385	25.6978
RSU6->RSU7	58.18	24.4315	0.9773	18.4840	23.8004

Table A.48 **RSU6->RSU7 KVM** Service Migration Time = 824.00, RSU6 58.19, RSU7 47.00, Xmin = 773.00, Xmax = 58.19

KVM	X	Av. no of requests	Av. Resp. Time	BP	LS
RSU6->RSU7	0	NP3			
RSU6->RSU7	1	NP3			
RSU6->RSU7	2	NP3			
RSU6->RSU7	3	NP3			
RSU6->RSU7	4	NP3			
RSU6->RSU7	6	NP3			
RSU6->RSU7	9	NP3			
RSU6->RSU7	12	NP3			
RSU6->RSU7	15	NP3			
RSU6->RSU7	18	NP3			
RSU6->RSU7	18.37	NP3			

A.4.2 Prototype for Service Management Framework in C Language

We successfully made a basic prototype of the Service Management Framework. It is written in C and the key structures and routines are shown below:

```
/* Server frame work structure */


/* Server frame work structure */

/*commands to the service*/
#define REGISTER_SERVICE 1
#define REQUEST_SERVICE 2
#define MIGRATE_SERVICE 3 /* NI - NOT IMPLEMENTED */
#define DEREGISTER_SERVICE 4

struct resources_required {
unsigned int CPU;     /* Number of Cores, Clock Speed */
unsigned int memory; //MBs of Memory
unsigned int network; // Mbps requirement)
unsigned int storage;  //(Gigabits Requirement)
};

struct qos {
int latency; /* in microseconds */
int bandwidth; /* Mbps */
int jitter;    /* in microseconds */
int reliability; /* Out of 100 where 100 is total reliability */
};


struct restriction_list {
int security_level; //(minimum Security level)
struct qos qos_min;   //(minimum)
int restrict_array_count;
int location [10]; //(restricted Networks)
int maximum_replication; //(how much replica count)
};

struct client {
unsigned int client_id; //(identifies clients to the service)
unsigned int node_id;
unsigned int location_id; // (IP address)
struct client *next;
struct client *prev;

};

struct client_list{

struct client *head;
struct client *tail;
int count;
};

struct server {
char executable [100]; // file name where the binary is to be found
unsigned int server_id;
unsigned int server_location; //(IP address)
unsigned short server_port;
unsigned char server_version;
unsigned char server_status; //(running, initiated, executable)
unsigned short maximum_server_load;
unsigned short current_load;
unsigned short maximum_client_on_server;
struct client_list server_client;
struct server *next;
struct server *prev;

// define as a service { unsigned short
  list_of_current_running_servers;}
};
struct server_list{

struct server *head;
struct server *tail;
int count;
};





struct network_requirement {
unsigned char types_of_handover_support; //(Proactive or Reactive)
unsigned char signalling_from_network_layer; //(L2 Trigger)
unsigned char list_of_network_interfaces; //(e.g. Heterogeneous)
unsigned char list_of_transport_protocol; //(e.g. TCP, UDP, and SLTP)
```

```
};

struct restrictions_max {
/*unsigned short Minimum_Security_level;
unsigned short Location_Restrictions;*/
unsigned short max_CPU;
unsigned short max_memory;
unsigned short max_storage;
};

struct recovery_mechanism {
unsigned short shutdown;
unsigned short restart;
};

struct service_min {
unsigned int server_id;
unsigned int ip_address;
};

struct service {
char archive_executable[100];
char service_name [100];
unsigned char archive_executable_len;
unsigned char service_name_len;
unsigned int service_id;
unsigned int service_version; /* if its "0" - it is a latest version!
  */
unsigned int maximum_number_servers; /* 1 */
unsigned int current_number_servers; /* 0 */
struct server_list service_server_list; /* NULL*/
struct resources_required service_resources_required;
struct qos service_qos;
struct restriction_list service_restriction_list;
struct network_requirement service_network_requirments;
struct restrictions_max service_restrictions_max;
struct recovery_mechanism service_recovery_mechanism;
struct service *next;
struct service *prev;
};

struct service_list{

struct service *head;
struct service *tail;
int count;
};


struct service_management_framework {
int smf_name_len;
char smf_name [100];
unsigned int status;// status 0 = down, 1= starting , 2 = up , 3 =
 closing , 4 = closed
struct service_list list_managed_services;
int number_of_request;
unsigned int port_number;
};

struct service_management_message {

int command;
int result;
struct service serv_mec; /*mechanism*/


};
```

Listing A.1 Header File for Service Management Framework

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <linux/types.h>
#include <asm/types.h>
#include <linux/socket.h>
#include <asm/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "server_framework.h"

#define SERVER_PORT 1070
struct service_management_message smm;
char *s = "/usr/bin/fuxfs_server";
char *sn = "network_memory_server";

/* change the program to connect to any device */

int main(argc, argv)
int argc;
char **argv;

{
int n;
int socksend_fd;
struct sockaddr_in msock, csock;
int len = sizeof(struct sockaddr_in);
int count = 0;
char cbuf[100];

bzero((char *)&csock, sizeof(struct sockaddr_in));
bzero((char *)&msock, sizeof(struct sockaddr_in));

msock.sin_family = AF_INET;
msock.sin_addr.s_addr = 0;
msock.sin_port = 0;

if((socksend_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
printf("%s: unable to get a socket\n", argv[0]);
exit(-1);
}
```

```

if((bind(socksend_fd, (struct sockaddr *)&msock, len)) < 0)
{
printf("%s: unable to bind to a local socket\n",argv[0]);
exit(-1);
}

/* convert the first argument to something I can use */

csock.sin_family = AF_INET;
/* csock.sin_addr.s_addr = 0;  /* connecting to local machine */
csock.sin_addr.s_addr = inet_addr(argv[1]);
csock.sin_port = htons(SERVER_PORT);

if((connect(socksend_fd, (struct sockaddr *)&csock, sizeof(struct
  sockaddr_in))) < 0)
{
printf("%s: unable to connect to server\n", argv[0]);
exit(-1);
}
printf("connected to the server\n");
bzero((char *)&smm, sizeof(struct service_management_message));
smm.command = REGISTER_SERVICE;
smm.result = -1;
strcpy(smm.serv_mec.archive_executable, s);
smm.serv_mec.archive_executable_len = strlen(s);
strcpy(smm.serv_mec.service_name, sn);
smm.serv_mec.service_name_len = strlen(sn);
smm.serv_mec.service_version = 0;
smm.serv_mec.maximum_number_servers = 1;
smm.serv_mec.current_number_servers = 0;
smm.serv_mec.service_server_list.count=0;
smm.serv_mec.service_resources_required.CPU = 1; /* Assum programs
  does not use threads*/
smm.serv_mec.service_resources_required.memory = 400 ; /*Megabytes (
  using physical memory) */
smm.serv_mec.service_resources_required.network = 100  ;/* Mbps */
smm.serv_mec.service_resources_required.storage = 10  ;/* Gigabytes */
smm.serv_mec.service_qos.latency = 100000; /* 100 milliseconds of
  latency */
smm.serv_mec.service_qos.bandwidth = 100; /* Mbps */
smm.serv_mec.service_qos.jitter = 100000 ; /* 100 milliseconds of
  Jitter */
smm.serv_mec.service_qos.reliability = 100 ; /* Totally reliable */
```

```
send(socksend_fd, (char *)&smm, sizeof(struct
  service_management_message), 0);

n = recv(socksend_fd, (char *)&smm, sizeof(struct
  service_management_message), 0);

if (n > 0)
{
printf("Result returned: %d\n", smm.result);
printf("Service Registered:   Service_id %d\n", smm.serv_mec.
  service_id);
}
else
{
printf("Unable recive from Service Management framework");
}

close(socksend_fd);
exit(0);
}



```

Listing A.2 C file for Register Service

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <linux/types.h>
#include <asm/types.h>
#include <linux/socket.h>
#include <asm/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "server_framework.h"

#define SERVER_PORT 1070
struct service_management_message smm;
char *s = "/usr/bin/fuxfs_server";
char *sn = "network_memory_server";

/* change the program to connect to any device */

int main(argc, argv)
int argc;
char **argv;

{
int n;
int socksend_fd;
struct sockaddr_in msock, csock;
int len = sizeof(struct sockaddr_in);
int count = 0;
char cbuf[100];

bzero((char *)&csock, sizeof(struct sockaddr_in));
bzero((char *)&msock, sizeof(struct sockaddr_in));

msock.sin_family = AF_INET;
msock.sin_addr.s_addr = 0;
msock.sin_port = 0;

if((socksend_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
printf("%s: unable to get a socket\n", argv[0]);
exit(-1);
}

```

```
 if((bind(socksend_fd, (struct sockaddr *)&msock, len)) < 0)
 {
 printf("%s: unable to bind to a local socket\n",argv[0]);
 exit(-1);
 }

 /* convert the first argument to something I can use */

 csock.sin_family = AF_INET;
 /* csock.sin_addr.s_addr = 0;  /* connecting to local machine */
 csock.sin_addr.s_addr = inet_addr(argv[1]);
 csock.sin_port = htons(SERVER_PORT);

 if((connect(socksend_fd, (struct sockaddr *)&csock, sizeof(struct
   sockaddr_in))) < 0)
 {
 printf("%s: unable to connect to server\n", argv[0]);
 exit(-1);
 }
 printf("connected to the server\n");
 bzero((char *)&smm, sizeof(struct service_management_message));
smm.command = REQUEST_SERVICE;
smm.result = -1;

 //strcpy(smm.serv_mec.archive_executable, s);
 //smm.serv_mec.archive_executable_len = strlen(s);
 strcpy(smm.serv_mec.service_name, sn);
smm.serv_mec.service_name_len = strlen(sn);
smm.serv_mec.service_version = 0;

 send(socksend_fd, (char *)&smm, sizeof(struct
   service_management_message), 0);

 n = recv(socksend_fd, (char *)&smm, sizeof(struct
   service_management_message), 0);

 if (n > 0)
 {
 printf("Result returned: %d\n", smm.result);
 printf("Service Registered:  Service_id %d\n", smm.serv_mec.
   service_id);
 }
 else
 {
  printf("Unable recive from Service Management framework");
  }

  close(socksend_fd);
  exit(0);
  }

```

Listing A.3 C file for Reqeust Service