6-2014

Opportunistic Service Differentiation and Cloud Resource Management in Support of Enhanced Vehicular Applications

Mohammad Ali Salahuddin
Western Michigan University, mohammad.salahuddin@ieee.org

OPPORTUNISTIC SERVICE DIFFERENTIATION AND CLOUD RESOURCE MANAGEMENT IN SUPPORT OF ENHANCED VEHICULAR APPLICATIONS

by

Mohammad Ali Salahuddin

A dissertation submitted to the Graduate College
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
Department of Computer Science
Western Michigan University
June 2014

Doctoral Committee:

Ala Al-Fuqaha, Ph.D., Chair
Dionysios Kountanis, Ph.D.
Mohsen Guizani, Ph.D.

OPPORTUNISTIC SERVICE DIFFERENTIATION AND CLOUD RESOURCE MANAGEMENT IN SUPPORT OF ENHANCED VEHICULAR APPLICATIONS

Mohammad Ali Salahuddin, Ph.D.

Western Michigan University, 2014

An integral part of Intelligent Transportation Systems (ITS) are Vehicular Ad hoc Networks (VANETs), which consist of vehicles with on-board units (OBUs) and fixed road-side units (RSUs). Wireless Access in Vehicular Environment (WAVE) offers QoS via service differentiation by using application defined priorities. However, WAVE has unbounded delay and is oblivious to network load and severity of vehicles with respect to their environment. Our context severity metric innovatively enhances WAVE to be sensitive to vehicle and environment interactions. Our novel Opportunistic Service Differentiation (OSD) technique, dynamically readjusts the WAVE packet priorities to improve utilization of lower latency queues, prioritizing packets in order of context severity. This also overcomes the unbounded delay in WAVE, which is crucial for safety applications.

On the other end, our novel RSU Cloud is a unique approach to hosting non-safety services on specialized RSU micro-datacenters. Optimal provisioning of constrained resources is critical in RSU Clouds. Furthermore, inherently dynamic demands from the vehicles require replications, migrations and, or instantiations of new or existing services, on virtual machines (VMs) in the RSU Cloud. We leverage the deep programmability of Software Defined Networking (SDN) to dynamically reconfigure the RSU Cloud. However, frequent changes to service hosts and data flows not only result in degradation of services, but are also costly for service providers. In Mininet, we analyze this reconfiguration overhead, which is used to design and model optimal RSU Cloud resource management (CRM).

CRM will optimally select service hosts and data forwarding rules, such that, the reconfigurations in the network are minimized with varying demands. We begin by designing the Pareto Optimal Frontier of non-dominated solutions (POF), such that, each solution is a configuration that minimizes either the number of service instances or the RSU Cloud infrastructure delay. The network is a priori configured for some demand and, now, the optimal CRM selects a configuration from the POF that minimizes the reconfiguration costs for the new demand. Together, CRM and OSD can improve QoS for ITS applications.

ACKNOWLEDGEMENTS

It is only due to the will of Al-Mighty Allah, that I hold this position and write my dissertation. I would like to thank the numerous faculty, fellow students, peer-reviewers from conferences and journals, for their time, efforts, and invaluable feedback in improving the quality of my research work and its presentation. I would like to send my deepest gratitude to the staff at Department of Computer Science, Western Michigan University, for their tremendous help and support in administrative tasks and equipment setup.

I am indebted and forever grateful to my Ph.D. Committee Chair, Dr. Ala Al-Fuqaha, for his support, guidance and far-sightedness in research problems and solution methodologies. I would also like to thank Dr. Dionysios Kountanis and Dr. Mohsen Guizani for the time they spent in reviewing my work and their invaluable and critical comments and suggestions.

My parents, siblings, wife, and children have been a support system and source of inspiration. I stand here today because of the perseverance inculcated in me, by my parents Mr. and Mrs. Hashim Ali Khan and the love and support of my brother Shaukat and sister Saadia. My wife, Zill-E-Huma Kamal, has been an integral part of this journey. As always, her support and belief in my success has pushed me through the thick and thin. I would like to acknowledge my children, Iman, Shaiq and Safa, for lulling my worries and anxieties with their smiles. They empower me to be a Super Hero and helped me find courage to stay course.

My faltered steps found renewed inspiration, motivation and dedication in my in-laws Mr. and Mrs. Mohammad Anwar Kamal and I thank them for everlasting love. Lastly, I am thankful to my late brother-in-law, Muneeb Kamal, for changing my life forever. He taught me to value time and live a purposeful life.

Mohammad Ali Salahuddin

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii
LIST OF TABLES v
LIST OF FIGURES vi
CHAPTER 1 INTRODUCTION 1
I. Opportunistic Service Differentiation 2
II. RSU Cloud Resource Management 6
CHAPTER 2 RELATED WORK 10
I. Limitations of WAVE 10
II. Clouds in support of VANETs 12
CHAPTER 3 OPPORTUNISTIC SERVICE DIFFERENTIATION FOR WAVE 14
I. Fuzzy Inference System 15
II. OSD Traffic Distribution Scheme 18
III. OSD Traffic Distribution Model 19
i. Problem Statement 19
ii. Delay Model 20
IV. Linear Programming Formulation 23
V. Lemmas and Theorem 26
VI. OSD Traffic Distribution–Heuristic 28
CHAPTER 4 RESULTS–OPPORTUNISTIC SERVICE DIFFERENTIATION 31
I. Analytical Results 31
II. Simulation and Results 34
i. Scenario 34
ii. Topology 35
iii. Analysis Setup 38
iv. Results and Discussion 38

Table of Contents - continued

CHAPTER 5 RSU CLOUD RESOURCE MANAGEMENT 46
I. RSU Cloud Architecture 46
II. Reconfiguration Overhead Analysis in Mininet 48
III. RSU Cloud Resource Management Model 51
i. Problem Statement 52
ii. Delay Model 52
IV. Multi-Objective Integer Linear Programming Formulation 54
V. RSU Cloud Resource Management–Heuristic 60
VI. Markov Decision Process 62
CHAPTER 6 RESULTS–RSU CLOUD RESOURCE MANAGEMENT 64
I. Scenario–Topology and Analysis Setup 64
II. Results and Discussion 65
CHAPTER 7 CONCLUSION–ENHANCED VEHICULAR APPLICATIONS 73
I. Opportunistic Service Differentiation for Safety Applications 73
II. RSU Cloud Resource Management for Non-Safety Applications 74
III. Future Work 75
REFERENCES 76
APPENDICES 81
A. Mininet Topology 82
B. Enabling Stochastic Switching in Open vSwitch 84
C. Mininet Python Topology Script 87
D. Performing Stochastic Switching in Mininet 93

LIST OF TABLES

1. OSD and WAVE QoS Parameters 31

2. OSD Simulation Parameters 36

3. OSD Vehicle Assignment to AC with Respect to Severity 37

LIST OF FIGURES

1. OSD service replaces user priority service in WAVE. 5
2. OSD scheme with sequence of interactions. 14
3. Membership functions for the input and output variables. 17
4. FIS rules to deduce context severity metric. 18
5. FIS input variables deduce context severity of an OBU. 18
6. Systematic load distribution (promotion and demotion) links to offer next best QoS. 25
7. Pseudo code for OSD traffic distribution heuristic. 29
8. Delay comparison of OSD enhanced WAVE with classical WAVE. 32
9. Load redistribution amongst ACs. 32
10. Load decomposition in AC, w.r.t severity, in OSD traffic distribution heuristic. 32
11. QoS w.r.t delay bounds on AC for 1000 vehicles. 33
12. Traffic demotion to achieve better QoS w.r.t delay for 240 vehicles. 33
13. EstiNet simulation scenario consisting of 1 consumer (vehicle ID 2) and 30 provider OBU. 37
14. Simulation performance comparison of classical WAVE with OSD enhanced WAVE with 95% confidence intervals. 41
15. Performance of vehicles with respect to severity in OSD enhanced WAVE with 95% confidence intervals. 44
16. RSU Micro-datacenter architecture. 47
17. RSU Cloud architecture. 48
18. Reconfiguration overhead analysis in Mininet. 50
19. FDOT RSU deployment [56]. 54
20. MDP for minimizing VM migrations. 63
21. Cost optimization optimally hosts services. 65

List of Figures - continued

22. Joint optimization consistently incurs lower VM migrations. 65

23. Cumulatively, Joint optimization incurs lower control plane modification with average demand dti over time ti. 66

24. Joint Optimization has magnitudes lower infrastructure delay, over changing average demand dti. 66

25. Every $\psi jti \in \Psi ti$ is a Pareto Optimal configuration with respect to number of service hosts and infrastructure delay. 67

26. It is evident that higher number of replications K, yields better results. 67

27. It is evident that higher number of replications K, yields better results w.r.t. cumulative average number of control plane modifications. 68

28. An increase in K the number of replications reduce margin of error, with respect to number of service hosts. 68

29. An increase in the number of replications reduce margin of error in the results, with respect to infrastructure delay. 69

30. Heuristic with K=100 and Optimization outperform purist Cost Optimization. 70

31. Heuristic incurs highest control plane modifications due to fine grain load balancing. 70

32. Purist Cost and Joint Optimization outperform Heuristic with K=100. 70

33. Heuristic with K=100 yields suboptimal infrastructure delay. Heuristic with K=100 outperforms Joint Optimization with an increase in the number of service hosts. 71

34. VM Migrations comparison for pure heuristic and heuristic followed by MDP, when starting with a 2 installation configuration. 71

35. VM Migrations comparison for pure heuristic and heuristic followed by MDP, when starting with a 3 installation configuration. 72

36. VM Migrations comparison for pure heuristic and heuristic followed by MDP, when starting with a 5 installation configuration. 72

CHAPTER 1

INTRODUCTION

Intelligent Transportation System (ITS) applications are envisioned to increase safety of drivers and passengers, reduce traffic congestion and pollution of the environment, and increase in-vehicle productivity. They are generally classified into safety, efficiency, convenience and infotainment applications. Safety applications, such as, collision avoidance, require local, neighborhood and contextual information of vehicles and surroundings. Whereas, non-safety applications such as, on-the-go internet, rely on Internet services. These applications pose unique requirements on the communication medium and require different degrees of quality of service (QoS) with respect to delay.

An integral part of ITS are Vehicular Ad hoc Networks (VANETs), which consist of vehicles with on-board unit (OBUs) and fixed roadside units (RSUs). The set of standards and protocols that govern vehicle-to-vehicle (V2V) in VANETs are defined in Wireless Access in Vehicular Environments (WAVE). WAVE offers QoS via service differentiation by using application defined priorities. However, WAVE has unbounded delay and is oblivious to network load and severity of vehicles with respect to their environment. Our context severity metric innovatively enhances WAVE to be sensitive to vehicle and environment interactions. Our novel Opportunistic Service Differentiation (OSD) technique, dynamically readjusts the WAVE packet priorities to improve utilization of lower latency queues, prioritizing packets in order of context severity. This also overcomes the unbounded delay in WAVE, which is crucial for safety applications.

On the other end, non-safety services are increasingly being hosted in the cloud. The RSU Cloud is a unique approach to hosting non-safety services on specialized RSU micro-datacenters. Optimal provisioning of constrained resources is critical in RSU Clouds. Furthermore, inherently dynamic demands from the vehicles require replication, migration and, or

instantiation of new or existing services, on virtual machines (VMs) in the RSU Cloud. A configuration is a snapshot of the network that records the service hosts and the data forwarding rules in the network. We leverage the deep programmability of SDN to dynamically reconfigure the RSU Cloud. The reconfiguration may prompt VM migrations, and modifications to the data forwarding rules. Frequent changes to service hosts and data flow not only results in degradation of service, but are also costly for service providers. We use Mininet to analyze this reconfiguration overhead.

The reconfiguration overhead is used to design and model optimal RSU Cloud resource management (CRM). In the face of dynamic demands, CRM will optimally select a service host and data forwarding configuration, such that, the reconfigurations in the network are minimized. We begin by designing the Pareto Optimal Frontier of non-dominated solutions (POF) such that each solution is a configuration that minimizes either the number of service instances or the RSU Cloud infrastructure delay, for a given demand. The network is a priori configured for some different demand and, now, the optimal CRM selects a configuration from the POF that minimizes the reconfiguration costs for the new demand.

Both CRM and OSD improve VANETs. First, OSD dramatically improves QoS, with respect to context severity. Second, CRM is a novel resource sharing technique that operates on the Pareto Frontier. Together, the CRM and OSD can improve QoS for ITS safety and non-safety applications.

I. Opportunistic Service Differentiation

Intelligent Transportation Systems (ITS) and services were initially coined to increase safety and reduce congestion and pollution [1]. Over the last decade, this framework has evolved and various fundamentals have been identified, including wireless communications for implementing ITS services. Consequentially, a 75 MHz bandwidth in the 5.9 GHz band was reserved for dedicated short range communications (DSRC) in ITS services. IEEE undertook the

task for specifying the protocols and standards for wireless communications enabling ITS services, while ITS users envisioned a range of diverse applications. These applications can be divided into different classes, such as safety, efficiency, convenience and infotainment.

IEEE 802.11p was developed as an amendment to IEEE 802.11, which defines standards for wireless local area network (WLAN) communication. The IEEE 802.11p standard facilitates DSRC and governs the medium access control (MAC) and physical (PHY) layers in the Open Systems Interconnection (OSI) model. The IEEE 1609.x protocols were developed to provide specifications that span over other OSI layers. The IEEE 1609.x protocols and IEEE 802.11p form the standards for wireless access in vehicular environments (WAVE), like Vehicular Ad hoc Networks (VANETs).

WAVE nodes consist of road-side units (RSUs) and on-board units (OBUs). Road-side units are ITS infrastructure nodes installed alongside the road, for example on traffic lights and road signs. On-board unit comprises of localization systems (for example, Global Positioning System, Inertial Measurement Unit, etc.), processing units and radio transceivers, mounted on the VANET vehicles. The IEEE 802.11p component in WAVE presides over the vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communications.

WAVE addresses the unique challenges in VANET and ITS applications, such as, fast moving nodes, multipath environment and applications with quality of service (QoS) requirements [1]. The communication channel in DSRC is divided into 7 sub channels, of 10 MHz each, consisting of one control (CCH) and six service channels (SCH). This multi-channel approach increases the efficiency of the channel and addresses high data transfer rate in fast and mobile environments. A channel is further divided into 4 access categories (ACs), each with a different priority for accessing the medium. This offers low latency communication, with service differentiation and QoS for the different classes of ITS applications.

The multi-channel coordination mechanism is addressed in IEEE 1609.4 and is incorporated into the OSI model, as a sub layer between LLC and MAC. It sits atop IEEE

802.11p MAC, which is an enhancement to 802.11a MAC. The multi-channel coordination mechanism employs four critical services to manage channel coordination and MAC service data unit (MSDU) data transfer [1]. The channel routing service (channel router) routes data packets to a designated channel. The user priority service selects an access category (AC) within the channel, based on the application defined priority. The channel coordination and MSDU data transfer service is responsible for channel selection and MSDU data delivery, respectively [1], as illustrated in Figure 1. Further details of DSRC and multi-channel operation in WAVE can be found in ([1], [2], [3], [4]).

ITS applications prioritize their application packets for one of the four ACs, based on their QoS requirement. Each AC prescribes a different set of parameter values that control its channel access and contention mechanism. This is adapted from the IEEE 802.11e enhanced distributed channel access (EDCA) mechanism to provide QoS, with respect to delay, and handle time-critical messages [2]. However, this priority to AC mapping does not account for network load, link layer bounds, the context of a vehicle, with respect to its severity and that of its neighbors or the road infrastructure information.

These short comings deteriorate the performance of WAVE in high traffic/dense vehicle scenarios [5] and causes time-critical ITS applications to exceed maximum acceptable delays [6]. This undermines the effectiveness of time-critical lifesaving safety applications for ITS. We propose a novel opportunistic service differentiation (OSD) scheme, which complements WAVE and overcomes these limitations. The OSD scheme extends WAVE nodes and primitives by incorporating a Fuzzy Inference System (FIS) in the OBU, an OSD traffic distribution heuristic on the RSU and replacing the user priority service with OSD service in the IEEE 1609.4 multi-channel operation sub layer, as illustrated in Figure 1.

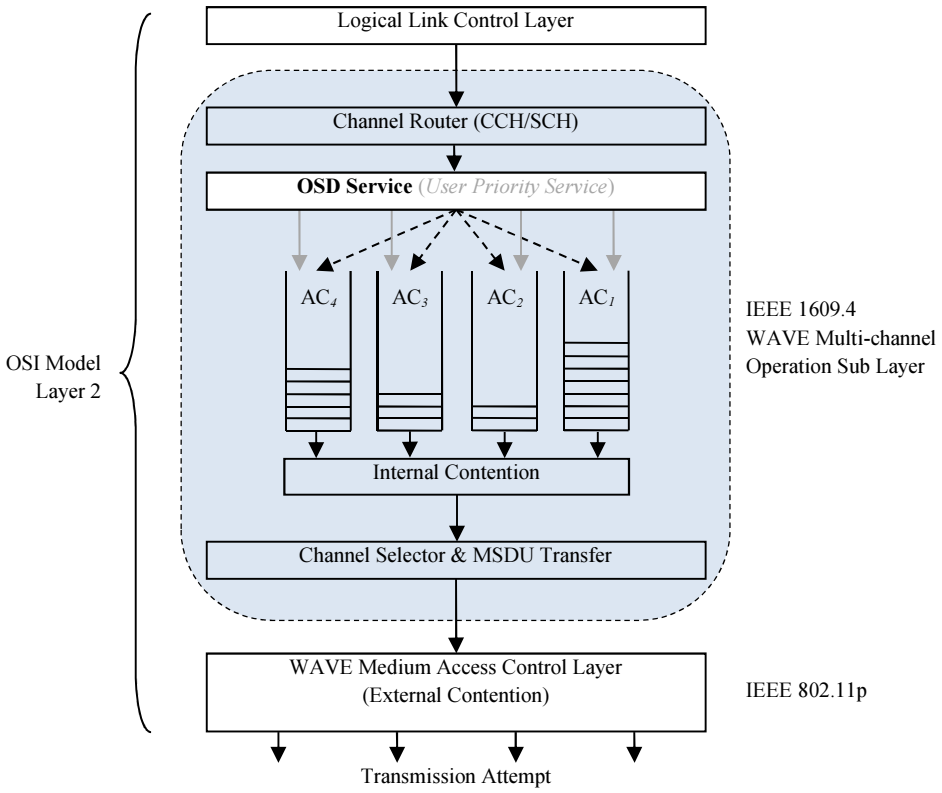


Figure 1. OSD service replaces user priority service in WAVE.

The OBU based fuzzy system infers the context severity metric for a vehicle. The context severity metric is based on the integration of the host vehicle and neighboring vehicles driving parameters with the ideal parameters from the road infrastructure. The driving parameters can be captured by a vehicle's speed, acceleration and directional stability. The ideal road infrastructure parameters can include speed limits, weather related hazards and other environmental advisories. These are typical parameters learnt over time for a given geographic region. The OSD traffic distribution heuristic on the RSU uses all the vehicles context severity metrics, network load, link layer and delay bounds on ACs to deduce a context severity based priority for each vehicle. This careful priority assignment for vehicle load ensures that ACs do not exceed delay bounds. The OSD service for a vehicle uses this context severity based priority to select an AC for all its traffic. This offers context severity aware service differentiation.

Our novel opportunistic service differentiation scheme has multiple improvements over

classical WAVE. First, we use a context severity metric that would lend WAVE critical information about its vehicles. The metric integrates critical vehicle driving parameters, such as speed, acceleration and directional stability with those of its neighboring vehicles. It is also augmented with road infrastructure information to build a context for the severity of a vehicle. This metric is used to prioritize traffic based on context severity of vehicles in the network, such that, vehicles with higher context severity are given better QoS with respect to delay.

Second, the OSD traffic distribution heuristic leverages network load and link layer bounds to opportunistically prioritize and distribute traffic amongst ACs to maximize the utilization of higher priority ACs and their inherent lower delays. The traffic is systematically redistributed amongst access categories such that the *next* best QoS is provided to the vehicles. Third, the OSD traffic distribution heuristic overcomes a major limitation of unbounded delays of WAVE [4]. We will show that the OSD traffic distribution mechanism not only improves the performance and increases the integrity of time-critical safety applications but also guarantees delay bounds on all ACs.

II. RSU Cloud Resource Management

Day in and day out, we suffer fatalities, deterioration of our environment and distress due to an antiquated labyrinth of roads. Recently, Intelligent Transportation Systems have received renewed attention from researchers and government agencies. ITS aims to integrate On-board Units in mobile vehicles with fixed roadside infrastructure into a Vehicular Ad hoc Network. Typically, OBUs consist of computational, communicational, storage, sensory and positioning systems, while RSUs can vary in size and form from small, resource constrained roadside mounted traffic monitoring cameras to high power communication towers. Numerous applications supporting ITS have been envisioned to promote safety, efficiency, convenience and infotainment.

The support of government agencies and researchers, alone, is not enough to push our roads to the smart network of interconnected vehicles and infrastructure as envisioned in ITS. We need the investment from large-scale commercial service providers. Recall, the Internet that we have come to associate as necessity, was a privileged network, only decades ago. The commercialization of the Internet was instrumental in the success it enjoys today. Similarly, if VANETs and ITS are marketed as business opportunities for commercial service providers, we will see ITS flourish. Undoubtedly, safety applications are the fundamental driving force of ITS but support for infotainment and convenience applications will increase its chances of success [7]. User interest in infotainment and convenience applications and services such as video on-demand, online multi-player gaming, on-the-go-Internet, voice over IP, remote vehicle diagnostic, road traffic management notifications, etc. will be the driving force for ITS market penetration and mass deployment of infrastructure.

ITS safety and non-safety applications and services have *very* different communication requirements. Safety applications are time sensitive with stringent limits on QoS with respect to delay and reliability ([6], [8]). Non-safety applications have very different QoS needs, ranging from no specific real-time QoS requirement to guaranteed QoS with respect to delay [7]. These disparate spectrums of QoS requirements make it impossible to design a one-fits-all solution. Researchers ([8], [9], [10]) have studied QoS for safety applications, while the focus of this research is QoS of non-safety applications. Lee *et al*. [11] call for a revolution in VANETs, integrating it with elements of cloud computing and information centric networking for safety and non-safety applications. The merging of cloud computing with VANETs opens a realm of possibilities for ITS applications and services.

Pending standardization, vehicular clouds instigate very different contributions to ITS. Some aim to interconnect vehicle resources into a cloud for cooperative sensory, storage and computing tasks [11], while others ([12], [13]) propose that Road-side Units act as gateways to traditional clouds or impose a cloud of On-board Units. In contrast to ([11], [12], [13]), we use

the reliability of fixed infrastructure and proximity of RSUs to end-users [14] to improve the quality of service (QoS). However, it is important to realize that the aforementioned vehicular clouds can be subsets and, or generalizations of each other. Moreover, as online infotainment services are moving to the cloud, vehicular clouds will not only prove to be a natural transition but a thriving commercial incentive.

We propose a novel architecture of a RSU Cloud. It consists of traditional RSUs and specialized micro-scale datacenters, which can host services. An inherent challenge in VANETs is maintaining QoS with dynamic demands. Commercial service providers will have to solve this challenge *cost effectively*. Our contribution is leveraging the flexibility and deep programmability of virtualization and Software Defined Networks (SDN), to dynamically migrate, replicate or instantiate services and reconfigure the data forwarding rules in the network to meet changes in service demands. For simplicity, we will refer to a snapshot of the network as a configuration. It captures the services instances and their locations and records the data forwarding rules in place to meet the service demands.

Despite the underlying benefits of the programmability of RSU clouds, service providers will incur costs pertaining to service instantiation and reconfiguration in light of changes in demands. Reconfiguration overhead is network traffic induced due to service migration and changes in data forwarding rules. This begs the question of optimal service hosting and configuration to reduce the overhead. Our contribution is the design of a novel RSU Cloud resource manager with multiple objectives. First, we minimize RSU Cloud infrastructure delay with QoS. Second, we minimize the number of RSUs hosting services, to minimize operational costs, e.g. renting resources, for service providers. And most importantly, we minimize the overhead of service migrations and data flow rule reconfigurations, which consume limited bandwidth resources and deteriorate network performance and QoS.

Therefore, the scope of this work and its contributions are stated below.

- Architecture for RSU Clouds and its micro-datacenters.

- Reconfiguration overhead analysis by emulating an OpenFlow [15] enabled SDN in Mininet [16].
- Define the RSU Cloud resource management model and solve it as a multi-objective Integer Linear Programming (ILP) problem for selecting a Pareto Optimal solution.
- Designing an efficient heuristic for RSU Cloud resource management.
- Use Markov Decision Process (MDP) and reinforcement learning to select a Pareto Optimal solution which minimizes the costly service migrations, over the long term.

CHAPTER 2

RELATED WORK

I. Limitations of WAVE

In this section, we will briefly discuss some related research work that studied the performance of WAVE and instigated the need to improve MAC protocols in WAVE, either by devising new MAC protocols or by improving the current MAC protocol in WAVE. We will compare these studies to our OSD scheme and highlight our major contributions.

Many researchers ([17], [9], [18]) have proposed new and ingenious MAC layer protocols that are more suitable, efficient, or secure for WAVE, while Mittag *et al*. [2] discuss different MAC approaches for WAVE. Based on these studies, we utilize the widely accepted and standard 802.11a MAC, which is the basis of IEEE 802.11p MAC. Therefore, we propose an enhancement in the multi-channel operation sub layer atop MAC in WAVE, which provides service differentiation with respect to context severity of vehicles.

Others ([19], [20]) have modeled IEEE 802.11a and IEEE 802.11b MAC distributed coordination function (DCF) and scrutinized their performance with respect to the throughput and delay. IEEE 802.11p MAC uses the carrier sense multiple access with collision avoidance (CSMA/CA) mechanism of IEEE 802.11a and service differentiation EDCA from IEEE 802.11e. Wu *et al*. [21] and Huang *et al*. [22] scrutinize QoS parameters of throughput and delay in IEEE 802.11e EDCA. Authors in [22] propose the use of an internal contention parameter that accounts for collision amongst different ACs, which is often overlooked.

Malik *et al*. [23] analyze WAVE MAC by developing a VANET model in MATLAB and evaluating channel access delay and probability of channel access. In our research, we use tractable equations for estimating delay, which reflects the intrinsic behavior of delays incurred by packets in the different ACs. S. Eichler [24] and N. Ferreira *et al.* [25] study WAVE to gain insight into collision probability, throughput and delay. There is an increase in delay in dense and

high load scenarios as studied by [24]. Wang *et al*. [26] also noticed similar deterioration in performance of WAVE since backoff window size does not adapt to increase in number of vehicles. They propose both centralized and distributed solutions to overcome these limitations by centrally calculating an optimal backoff window size or adapting the window size locally. We propose a global approach, such that the OSD traffic distribution mechanism distributes vehicle traffic based on the context severity metric. This promotes better QoS for vehicles with higher context severity.

Various authors have devised mechanisms for improving QoS for safety and non-safety applications. Amadeo *et al.* [9] and Zhou *et al.* [27] aim to provide QoS for non-safety applications. The former use an innovative MAC layer protocol that enhances the ability of WAVE to improve QoS, while the latter studies a content dissemination technique that employs timestamp based reward/user satisfaction optimization. Authors in [28] develop enhancements to WAVE to reduce the delay for safety applications. Our OSD traffic distribution mechanism strives to improve the QoS for both safety and non-safety applications. Our contribution is to extend the underlying communication protocols in WAVE by reprioritizing traffic to different ACs, to provide context severity aware service differentiation.

Furthermore, [29] uses the position of vehicles to ensure effective utilization of the channel resource, while we use the contextual information inferred from the position of a vehicle. However, both approaches bound the delay from above, unlike the unbounded delay of classical WAVE, which compromises time sensitive applications. Chrysostomou *et al*. [10] designed a dynamic tuning of contention window parameters to offer service differentiation in WAVE and improve the QoS with respect to throughput. The authors in [11] also design adaptations of the WAVE MAC protocol by using dynamic contention windows and vehicle mobility parameter of average speed to provide service differentiation. Others have analyzed and evaluated the MAC layer parameters for channel access delay and probability of channel access [23].

However, OSD uses the standard WAVE contention window and MAC layer parameters.

It uses more complex vehicle mobility parameters, such as acceleration and directional stability, to conjure a context severity metric for vehicles in the network. Our major contribution is the OSD scheme that provides service differentiation based on the context severity metric, network load and delay bounds on the ACs. We show an improvement over classical WAVE by using performance metrics, like throughput, delay, number of collisions, packet loss ratio and retransmission ratio. OSD achieves this by using a novel technique to distribute load opportunistically amongst different ACs, optimally utilizing the higher priority ACs and guarantees minimum delay experience.

II. Clouds in support of VANETs

Taxonomy and classification of ITS services and their requirements have been presented in [12] and [11], and it is evident from these classifications that ITS safety applications are dependent on direct vehicle-to-vehicle (V2V) communication, whereas, ITS non-safety applications reply on resource constrained RSUs. Therefore, strengthening the RSUs with micro-datacenters enables them to provide non-safety services. We propose a RSU cloud to capitalize the proximity of RSUs to the end-users and the reliability of fixed infrastructure, to deploy a deeply programmable network that can cater to dynamic service demands. Proponents of Fog Computing [14] instigate the benefits of moving services to the edges of the network to increase user exper-ience. In comparison to traditional clouds, RSU Clouds are assumed to be resource constrained.

Strengthening the vehicles in the VANET with a cloud not only enables myriad computational capabilities that are underutilized by safety applications alone ([12], [30]), but also overcomes the unreliable V2V communication [31]. Some aim to interconnect OBU and RSU resources into a cloud for cooperative sensory, storage and computing tasks [11], while others ([12], [13]) propose that Road-Side Units (RSUs) act as gateways to traditional clouds or design a cloud of On-board Units (OBUs). Vehicular Cloud Networking (VCN) [11] is being

proposed as a revolution to modernize the traditional VANET, which integrates information centric networking and cloud computing with VANETs. In VCN vehicles and resource constrained RSUs share their resources in one virtual platform. This is in contrast to our proposed RSU cloud, which only includes RSUs. Amongst the RSUs in the RSU cloud, some are specialized RSUs that contain micro-datacenters and can form a SDN of other RSU micro-datacenters to dynamically host services and reconfigure data flows. Our RSU cloud can easily co-exist with earlier VANET clouds or within the revolutionizing VCN.

Current techniques to ensure efficient and effective realization of RSU cloud services include rich connectivity at the edge of the network and dynamic routing protocols to balance traffic load [32] and reduce routing delay. Presently, capacity planning tools such as VMWare Capacity Planer, IBM Websphere Cloudburst, Novell PlateSpin Recon, etc. decide the VM placement locations [32]. However, they lack in load balancing at VM and result in highly imbalanced traffic distribution [32]. Various researchers ([33], [34], [35], [36], [37]) have proposed solutions for low latency cloud service deployment, either independently of cost of service location or jointly. However, like Wu *et al*. [33], we will also jointly minimize cloud service deployment cost and routing delay for the Pareto frontier. In contrast to Wu *et al.* [33] we leverage the list of Pareto optimal solutions for selecting the one deployment and network reconfiguration, with the least effect on existing service and routing configurations, while [33] uses a Nash bargaining technique to balance optimality with fairness. Our significant contribution lays in the wake of the fact that VM migrations are costly [38].

The classic cloud resource management techniques are not applicable to RSU clouds, due to the dynamic service demands and the resource constrained micro-datacenters. These pose great challenges for RSU clouds. First, VM migration is a costly process [38], which deteriorates network performance. Second, data flow reconfigurations in OpenFlow enabled networks incur cost in the control plane when flows are modified [39].

CHAPTER 3

OPPORTUNISTIC SERVICE DIFFERENTIATION FOR WAVE

In this chapter, we describe our Opportunistic Service Differentiation (OSD) scheme, as illustrated in Figure 2. OSD scheme works in both unicast and broadcast scenarios. The OBU and RSU nodes are enhanced with additional functionalities and the user priority service is replaced with the OSD service in the multi-channel operations sub layer in WAVE. The RSU and OBU are augmented with an OSD traffic distribution heuristic and a fuzzy inference system (FIS), respectively.

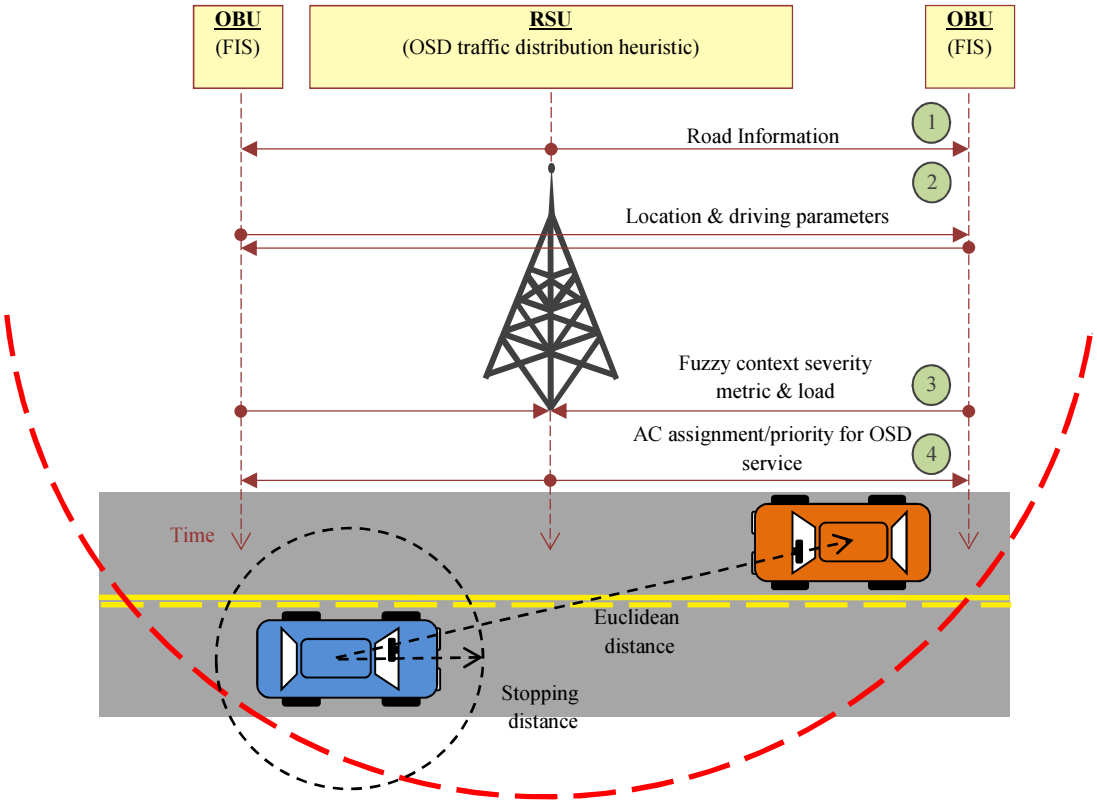


Figure 2. OSD scheme with sequence of interactions.

In step one, the RSU broadcasts road infrastructure information, specifically the ideal speed, ideal acceleration and ideal directional stability. The directional stability measures road lane departures. These are the RSU suggested ideal parameters since they are based on the

integration of speed limit with real-time weather and road condition advisories (for example, construction zones, congestion, etc.). In step two, every OBU broadcasts its location and driving parameters. The rule based FIS uses this to deduce the vehicle's context severity metric, which is relayed to the RSU in step three, along with vehicle's traffic load. The OSD traffic distribution heuristic in the RSU computes the priority for each vehicle, and communicates the new priorities to them in step four.

The OSD traffic distribution heuristic provides context severity aware service differentiation, while accounting for network load, link layer bounds and the delay thresholds on ACs. The output from the OSD traffic distribution heuristic is the priority used by the OSD service for AC selection, in the multi-channel operation shown in Figure 1. In this manner, OSD enhanced WAVE overrides application priorities, such that, the context severity of a vehicle determines the priority for all its traffic through an AC. Thus, OSD traffic distribution heuristic assigns vehicle traffic to an AC, which is a tradeoff, since it may not fully utilize AC capacity.

In the next subsections, we will delineate the details of the FIS in the OBU and give an overview of the OSD traffic distribution scheme on the RSU.

I. Fuzzy Inference System

A fuzzy inference system is installed in the vehicle's OBU to deduce the context severity metric for a vehicle, inspired from Elbes *et al.* [40]. Our novel context severity metric gauges the severity of a vehicle relative to its neighborhood and environment. The FIS employs rule-based logic to map neighborhood and environment input data to a context severity metric output. In this manner, a vehicle is associated with urgency of the vehicles in its neighborhood.

Environment parameters constitute factors such as traffic congestion, construction, weather elements, etc. The RSU integrates these factors with the road infrastructure information and broadcasts the ideal speed, ideal acceleration and ideal directional stability, y_s, y_a and y_d, respectively. Similarly, all vehicles v_j in a neighborhood broadcast their location and driving

parameters of speed, acceleration and directional stability, $x_s^{v_j}$, $x_a^{v_j}$ and $x_d^{v_j}$, respectively. Vehicle v_i uses a look up table (LUT) to find its stopping distance based on its driving parameters. This stopping distance is used to demarcate an area of critical interest, depicted in Figure 2, where v_i gives more weight w_{ij} to the driving parameters of all vehicles v_j that fall in this area. Each vehicle v_i uses Equation (1) to compute weights w_{ij}, for all neighbors v_j.

$$w_{ij} = \begin{cases} 1, & EuclideanDistance(v_i, v_j) \leq StoppingDistance^{v_i} \\ \dfrac{StoppingDistance^{v_i}}{EuclideanDistance(v_i, v_j)}, & otherwise \end{cases} \quad (1)$$

The relative computation of speed, acceleration and directional stability with respect to its neighboring vehicles and the RSU suggested ideal parameters are termed context speed, context acceleration and context directional stability. Each vehicle uses Equations (2), (3) and (4) to deduce its context speed, acceleration and directional stability. Evidently, vehicles within the stopping distance have a higher weight and hence a greater influence on vehicle's contextual information. Consequentially, the context driving parameter of a vehicle is the weighted average of the deviation of all neighborhood vehicles from the RSU suggested ideal parameters.

$$ContextSpeed^{v_i} = \frac{\sum_{j=1}^{V}\left(w_{ij} \times \frac{x_s^{v_j} - y_s}{10}\right)}{V} \quad (2)$$

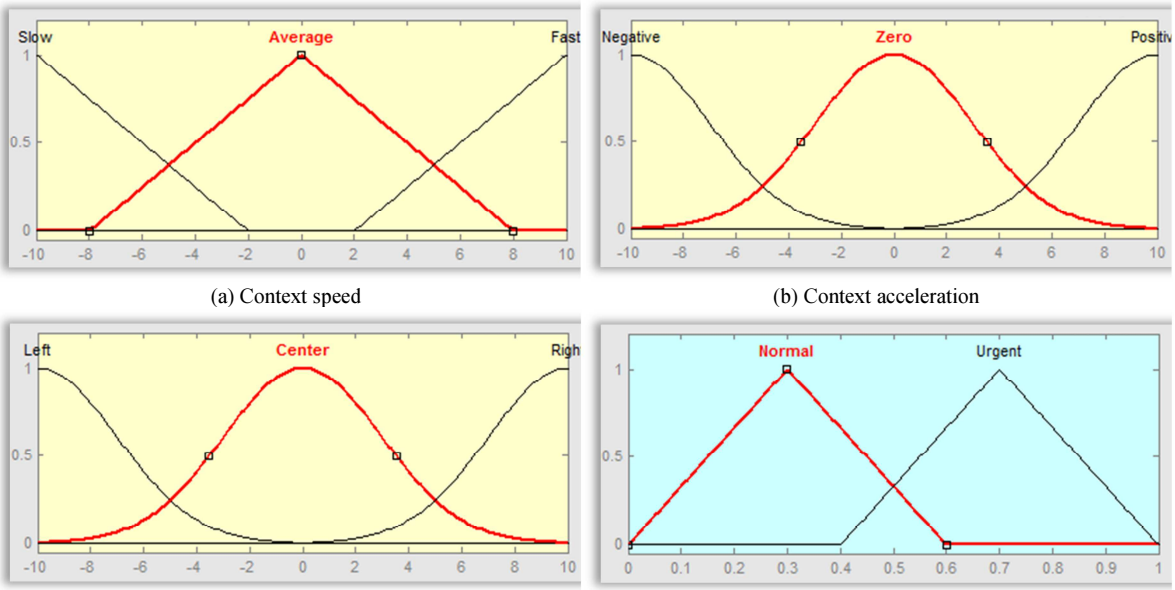
$$ContextAcceleration^{v_i} = \frac{\sum_{j=1}^{V}\left(w_{ij} \times \frac{x_a^{v_j} - y_a}{10}\right)}{V} \quad (3)$$

$$ContextDirStability^{v_i} = \frac{\sum_{j=1}^{V}\left(w_{ij} \times \frac{x_d^{v_j} - y_d}{10}\right)}{V} \quad (4)$$

The context speed, context acceleration and context directional stability are system inputs to the FIS, which is used to deduce the output, the context severity metric of a vehicle. The FIS uses membership functions to transform the crisp input values into a value between 0 and 1,

which denotes the degree of membership to a fuzzy set. The if-then rules defined in the FIS are used to infer a fuzzy set. This fuzzy set is retransformed using membership functions into a crisp output value.

Figure 3 illustrates the membership functions (MF) we have used for our system input and output. The membership functions in a FIS can be decomposed into simpler triangular and trapezoidal MF or more complex Gaussian and bell-shaped functions. The MF chosen represent the degrees of membership into the fuzzy set. We use the simpler triangular MF for context speed, since it can be defined by the three points, slow, average and fast. However, the MF for context acceleration and context directional stability are bell-shaped, to utilize their smoothness. Similar to context speed, context severity is defined by MF using two discrete points, normal and urgent.



(a) Context speed

(b) Context acceleration

(c) Context directional stability

(d) Context severity

Figure 3. Membership functions for the input and output variables.

A set of rules are used to implement the FIS. These rules are delineated in Figure 4 and conjure that a vehicle's context severity is influenced by the deviation of the driving parameters of the vehicles in its neighborhood, including itself. Note, a higher magnitude of the context severity metric implies a higher severity of the vehicle within its context.

```
Rule 1: IF (ContextSpeed IS NOT Average) THEN
                    (ContextSeverity IS Urgent)
Rule 2: IF (ContextAcceleration IS NOT Zero) THEN
                    (ContextSeverity IS Urgent)
Rule 3: IF (ContextDirStability IS NOT Center) THEN
                    (ContextSeverity IS Urgent)
Rule 4: IF (ContextSpeed IS Average)
            AND (ContextAcceleration IS Zero)
            AND (ContextDirStability IS Center) THEN
                    (ContextSeverity IS Normal)
```

Figure 4. FIS rules to deduce context severity metric.

Figure 5 illustrates an example that depicts the relation of the input variables to the output variable in our FIS. It shows the crisp values for context speed, context acceleration and context directional stability and the membership functions. Each row, numbered 1 through 4, in Figure 5 illustrates the effect of each rule, from Figure 4, on the respective MF.

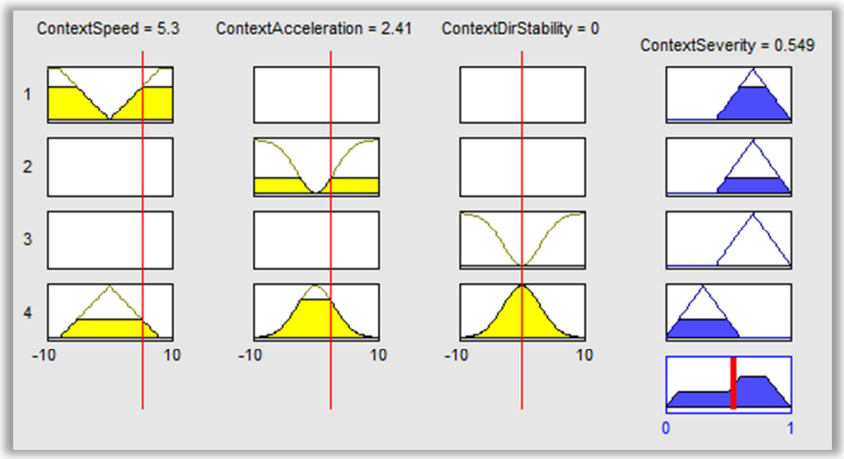


Figure 5. FIS input variables deduce context severity of an OBU.

II. OSD Traffic Distribution Scheme

All the vehicles will periodically transmit their context severity and load to the RSU performing the traffic distribution using the OSD traffic distribution heuristic. The RSU will use this information to form a global snapshot of vehicles context severities and loads. It will assign the vehicle to an AC based on its context severity, such that, all traffic from the vehicle is

transmitted through the assigned AC. The vehicles are assigned to ACs, such that, delay bounds on the ACs are not exceeded and vehicles with higher context severity metric are given better QoS with respect to delay. Hence, OSD provides service differentiation based on context severity of vehicles. It also overcomes unbounded delay, which is a major limitation in WAVE [4].

The OSD traffic distribution heuristic is our major contribution and the highlight of this research. We validate our claims of performance improvement with OSD enhanced WAVE, by formulating an unambiguous optimization problem. We define lemmas and a theorem and prove them to instigate the accuracy of our OSD traffic distribution heuristic. Furthermore, we use simulation tools to study the effect of context severity aware service differentiation in WAVE.

III. OSD Traffic Distribution Model

In this section, we will present the problem statement, delay model and discuss the Linear Programming (LP) formulation. The LP formulation defines the mathematical model for OSD enhanced WAVE, which is used for verification and validation. A mathematical model is deterministic, unambiguous with reproducible results, and serves as a baseline for the design of OSD traffic distribution heuristic. We develop a delay model that approximates the delay incurred by traffic in each AC. This enables us to use delay in a tractable and linear constraint in the LP formulation.

i. Problem Statement

Given a set of vehicles $\{v_1, v_2, \dots, v_V\}$ with context severities $\{S^{v_1}, S^{v_2}, \dots, S^{v_V}\}$, traffic loads $\{\lambda^{v_1}, \lambda^{v_2}, \dots, \lambda^{v_V}\}$ and N access categories each with a delay d_k bounded by T_k^{MAX}. Find an optimal assignment of vehicle loads to the ACs, such that, it provides QoS with respect to context severity, while not exceeding the delay bounds T_k^{MAX} for AC_k, $\forall\ 1 < k \leq N$.

ii. Delay Model

We develop a simplified model to approximate the delay, similar to [41], on an AC. It appears as a linear constraint in our LP formulation. This model allows us to compare the performance of OSD enhanced WAVE with the classical WAVE for analytical and simulation results. It is not in the scope of this research to design an intricate delay analysis model as presented in ([42], [43]), instead we approximate the process to illustrate the benefits of OSD enhanced WAVE.

We model MAC access delay, which is the time from when a packet becomes the head of the MAC queue to the time it is successfully transmitted. Recall, WAVE MAC access is based on CSMA/CA mechanism, where packets are transmitted when the channel is sensed to be idle. If the channel is busy, a retransmission is attempted after a specified backoff period. Similar to [42], we also assume collision resolution time is included in the backoff period and without loss in accuracy we use exponential distribution for backoff periods, with a mean duration of $1/\beta$.

Each vehicle generates packets according to a Poisson distribution with probability density function in Equation (5). We assume all vehicles are generating the same number of packets, each of the same size, then the probability of finding the channel idle α, is a constant for a given number of vehicles. This means that the number of backoffs k before a successful transmission, is geometrically distributed [42] with a probability mass function $\alpha(1-\alpha)^{k-1}$. Thus, the total backoff period is a convolution of possibly infinite number of backoffs [42]. Using Laplace transform, backoff periods can be modeled as in (6) and for k successive backoffs it becomes $\left(\frac{\beta}{\beta+s}\right)^k$. Then, the Laplace transform for the total backoff period is (7). Consequentially, the backoff period, which was exponentially distributed, has a mean rate of $\alpha\beta$ [42].

$$f(t) = \beta e^{-\beta t}, \forall t \geq 0 \tag{5}$$

$$\mathcal{L}[f(t)] = \int_0^\infty f(t)\, e^{-st} dt \quad = \int_0^\infty \beta e^{-\beta t}\, e^{-st} dt = \left[\frac{\beta}{-(s+\beta)} e^{-(\beta+s)t}\right]_0^\infty = \frac{\beta}{\beta+s} \tag{6}$$

$$F(s) = \sum_{k=1}^{\infty} \alpha\,(1-\alpha)^{k-1} \left(\frac{\beta}{\beta+s}\right)^k = \sum_{k=1}^{\infty} \frac{\alpha\beta}{\beta+s} \left(\frac{(1-\alpha)\beta}{\beta+s}\right)^{k-1} \tag{7}$$

$$= \frac{\alpha\beta}{\beta+s} \sum_{l=0}^{\infty} \left(\frac{(1-\alpha)\beta}{\beta+s}\right)^l$$

$$where \sum_{l=0}^{\infty} \left(\frac{(1-\alpha)\beta}{\beta+s}\right)^l \; is\; a\; geometric\;\; series\,, then$$

$$F(s) = \frac{\alpha\beta}{\beta+s} \left(\frac{1}{1-\frac{(1-\alpha)\beta}{\beta+s}}\right) = \left(\frac{\alpha\beta}{\alpha\beta+s}\right)$$

Our delay model approximates the delay for a packet to be the time spent in the total backoff period. Therefore, the delay is given as $1/\alpha\beta$. Intuitively, $1/\alpha$ is the mean number of retries and $1/\beta$ is the mean delay per retry. Recall, WAVE service differentiation is achieved by using different channel contention parameters for the ACs. Therefore, we formulate the delay incurred by traffic in each AC_k as

$$d_k = \frac{1}{\alpha_k \beta_k}. \tag{8}$$

The probability of finding the channel idle α_k is in Equation (9), where the inter-arrival time is inversely proportional to the load on the AC and transmission time $\Delta t = \frac{packet\ size}{data\ rate}$. Each vehicle v_i generates packets for each AC_k according to a Poisson distribution with mean rate $\lambda_k^{v_i}$, then the load on AC_k is $\sum_{i=1}^{V} \lambda_k^{v_i}$.

$$\alpha_k = \frac{interarrival\ time}{interarrival\ time + transmission\ time} = \frac{\frac{1}{\sum_{i=1}^{V} \lambda_k^{v_i}}}{\frac{1}{\sum_{i=1}^{V} \lambda_k^{v_i}} + \Delta t} = \frac{1}{1 + \Delta t \sum_{i=1}^{V} \lambda_k^{v_i}}. \quad (9)$$

The backoff mechanism in WAVE dictates the backoff period β_k for each AC_k. A vehicle wanting to transmit a packet from AC_k selects a random backoff time if it senses the channel is idle for arbitration inter-frame space ($AIFS_k$) in Equation (10) from [44]. The backoff time is a random number from the uniformly distributed interval [0, CW_k +1], where the initial contention window (CW_k) is equal to CW_k^{min}. However, if the subsequent transmission attempt fails, the interval is doubled by increasing the CW_k value until it reaches CW_k^{max}. The backoff value is decreased when the channel is free. Therefore, we model the backoff period as (11), where $CW_k^{min}(SlotTime)$ is the lower bound on the contention window size for AC_k. Table 1 ([45], [44], [46], [47]) delineates the WAVE parameters used to compute β_k.

$$AIFS_k = SlotTime \times AIFSN_k + SIFSTime \quad (10)$$

$$\beta_k = \frac{1}{AIFS_k + CW_k^{min}(SlotTime)} \quad (11)$$

Substituting Equation (9) into (8), we get the delay on AC_k as

$$d_k = \frac{1 + \Delta t \sum_{i=1}^{V} \lambda_k^{v_i}}{\beta_k}. \quad (12)$$

Similar to [41], we do not account for hidden node terminals or saturated/unsaturated traffic to approximate the process. We validate our delay model by illustrating that this analytical delay follows the delay in simulation, as depicted in Fig. 14(b). In this research, we study the performance of OSD in VANET, by taking a snapshot of the scenario and assuming a perfect channel [41]. Our model can be extended to account for factors such as vehicular mobility and channel fading.

IV. Linear Programming Formulation

We formulate our problem as a linear programming (LP) model with continuous variables. In the subsections below, we define the parameters, variables and present the formulation along with its discussion.

Parameters

N	Number of ACs
V	Number of vehicles
T_k^{MAX}	Upper bound on delay for AC_k
Δt	Transmission time, $\frac{packet\ size\ (bytes)}{data\ rate\ (Mbps)}$
β_k	Rate of backoff for AC_k
$\lambda_k^{v_i}$	Mean rate of packet generation, here after referred to as, load from v_i for AC_k
S^{v_i}	Context severity of v_i

Variables

$\lambda_{k\ demo\ j}^{v_i}$	$\lambda_{k,k-1,k-2,\ldots,j}^{v_i} \equiv \lambda_{k\ demo\ j}^{v_i}$ load demoted from AC_k to AC_j, $1 \leq k, j, k \neq j \leq N$
$\lambda_{k\ promo\ j}^{v_i}$	$\lambda_{k,k+1,k+2,\ldots,j}^{v_i} \equiv \lambda_{k\ promo\ j}^{v_i}$ load promoted from AC_k to AC_j, $1 \leq k, j, k \neq j \leq N$
d_k	Delay for AC_k

Formulation

Objective

$$min\ max\ \{S^{v_i} d_k\ \forall v_i, 1 \leq k \leq N\} \tag{13}$$

Subject to

$$d_k = \frac{1+\Delta t \sum_{i=1}^{V} \begin{pmatrix} \lambda_k^{v_i} + \sum_{j=k+1}^{N} \lambda_{j\ demo\ k}^{v_i} - \sum_{j=k}^{N} \lambda_{j\ demo\ k-1}^{v_i} \\ + \sum_{j=1}^{k-1} \lambda_{j\ promo\ k}^{v_i} - \sum_{j=1}^{k} \lambda_{j\ promo\ k+1}^{v_i} \end{pmatrix}}{\beta_k} \forall 1 \leq k \leq N \quad (14)$$

$$d_k \leq T_k^{MAX} \ \forall 1 < k \leq N \quad (15)$$

$$\lambda_{k\ promo\ k+1}^{v_i} + \lambda_{k\ demo\ k-1}^{v_i} \leq \lambda_k^{v_i} \ \forall v_i, 1 \leq k \leq N \quad (16)$$

$$\lambda_{k\ demo\ j-1}^{v_i} \leq \lambda_{k\ demo\ j}^{v_i} \ \forall v_i, 1 < k \leq N, 1 < j < k \quad (17)$$

$$\lambda_{k\ promo\ j}^{v_i} \geq \lambda_{k\ promo\ j+1}^{v_i} \ \forall v_i, 1 \leq k < N, k < j < N \quad (18)$$

Discussion

We formulate context severity aware service differentiation by employing severity delay products and a systematic load redistribution mechanism in our model. We use severity delay products as an innovative approach to achieve a minimization of the delay for vehicles with higher context severity metric. Recall, vehicles with higher context severity metric are in an urgent state, requiring higher QoS with respect to delay for time-critical safety applications. Therefore, our objective is formulated as the minimization of the maximum severity delay product.

Since the context severity of vehicles is a fixed value, the goal becomes to minimize the delay across the ACs, by systematically redistributing loads amongst ACs. Our minimax objective (13) entails a constant effort to improve the worst severity delay product across all ACs, to reach the optimal severity delay products. This optimality occurs when delay is balanced across all ACs or at predetermined threshold T_k^{MAX}. The delay on an AC is directly proportional to the load on that AC, and a higher priority AC can be assigned more load than a lower priority AC to reach the same delay. Inherently, the higher priority ACs are utilized first, in order of context severity.

Constraint (14) captures the delay for AC_k based on Equation (12) with the newly distributed load on it. The new load on AC_k is the sum of the loads from v_i for AC_k, the load promoted and demoted to AC_k, less the load promoted and demoted to AC_{k+1} and AC_{k-1}, respectively. It is evident from Table 1, that $\beta_1 < \beta_2 < \beta_3 < \beta_4$ for $N = 4$. This implies that the same load incurs a lower delay if assigned to AC_4 when compared to AC_3, and so on. Thus, a higher index AC has higher priority than a lower index AC. WAVE service differentiation is accomplished due to the difference in backoff for the different ACs.

We deploy a series of virtual links between each AC, which abstracts the promotion or demotion of load between ACs, as illustrated in Figure 6, where $\lambda^{v_i}_{k\ promo\ j}$ and $\lambda^{v_i}_{k\ demo\ j}$ is the load promoted or demoted from AC_k to AC_j, respectively. These links have been designed to systematically redistribute the load amongst the ACs, such that AC_k *first* promotes or demotes its neighbor in the chain as depicted in Figure 6. For example, AC_4 load is not directly demoted to AC_1, instead it is gradually demoted to AC_3, in an effort to offer the *next best* QoS.

Constraint (15) bounds the delay d_k for AC_k by T_k^{MAX}. It is important to carefully choose T_k^{MAX} to ensure QoS with respect to delay by not exceeding the maximum allowable delay for an AC. Note, that AC_1 is the lowest priority AC and is unbounded to accommodate load overflows from higher priority AC. Constraints (16), (17) and (18) are used for bookkeeping and ensuring proper distribution of loads amongst the ACs.

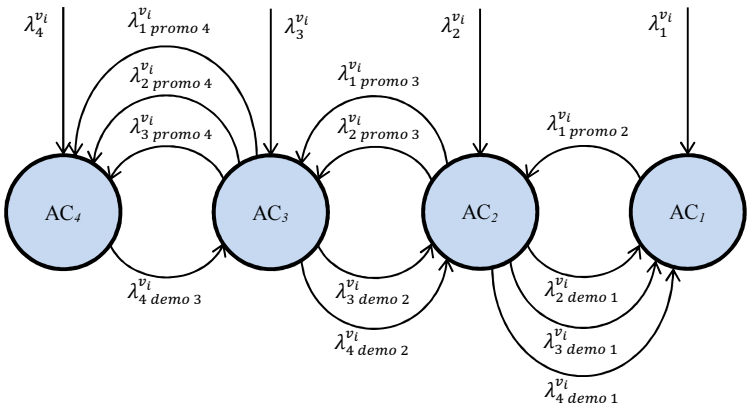


Figure 6. Systematic load distribution (promotion and demotion) links to offer next best QoS.

V. Lemmas and Theorem

The fundamental of OSD traffic distribution is the minimization of severity delay products. Since the severity metric is constant for each vehicle v_i, to minimize a high severity delay product, the delay has to be minimized. This implies that load from low priority access categories with the high delay, should be pushed to higher priority access categories, with the lower delay, to minimize the severity delay product.

Given a set $\boldsymbol{V}$ of M vehicles and N access categories, such that, $\boldsymbol{V} = \{v_1, v_2, \dots, v_M\}$ with context severities $\boldsymbol{S} = \{s^{v_1}, s^{v_2}, \dots, s^{v_M}\}$, traffic loads $\boldsymbol{L} = \{\lambda^{v_1}, \lambda^{v_2}, \dots, \lambda^{v_M}\}$ and AC_i MAC delay $d_i = \frac{1+\Delta t \sum_{k=1}^{M} \lambda_i^{v_k}}{\beta_i}$ $\forall 1 \leq i \leq N$, where $\lambda_i^{v_k} = \begin{cases} 0, \\ \lambda^{v_k} \end{cases}$ is the load of vehicle v_k on AC_i and β_i is the rate of back off for AC_i, where $\beta_i < \beta_{i+1}$ and $\Delta t = \frac{packet\ size}{data\ rate}$.

For $\boldsymbol{V} = \emptyset, \sum_{k=1}^{M} \lambda_i^{v_k} = 0,\ d_i = \frac{1}{\beta_i}$ $\forall 1 \leq i \leq N$ and $\beta_i < \beta_{i+1} \Rightarrow d_{i+1} < d_i$ $\forall 1 \leq i \leq N-1$ and AC_{i+1} has higher priority than AC_i. For $\boldsymbol{V} \neq \emptyset$, and no upper bound on d_i, all delay severity products $\{s^{v_k} d_i$ $\forall 1 \leq k \leq M, 1 \leq i \leq N\}$ can be minimized.

***Lemma 1 (delay equalization)*:** Minimum severity delay products $\{s^{v_k} d_i$ $\forall 1 \leq k \leq M, 1 \leq i \leq N\}$ occur when delay is balanced, $d_i = d_{i+1}$ $\forall 1 \leq i \leq N-1$.

***Proof by contradiction*:** Assume we have minimized each of the severity delay products in $\{s^{v_k} d_i$ $\forall 1 \leq k \leq M, 1 \leq i \leq N\}$ and $d_i \neq d_{i+1}$ $\forall 1 \leq i \leq N-1$. Then $d_i < d_{i+1}$ or $d_{i+1} < d_i$. If $d_i < d_{i+1}$, then $s^{v_k} d_{i+1}$ is not minimum, since $\exists\ d_i \mid s^{v_k} d_i < s^{v_k} d_{i+1}$. Similarly, if $d_{i+1} < d_i$, then $s^{v_k} d_i$ is not minimum, since $\exists\ d_{i+1} \mid s^{v_k} d_{i+1} < s^{v_k} d_i$. Contradicting the hypothesis that each $s^{v_k} d_i$ $\forall 1 \leq k \leq M, 1 \leq i \leq N$ is minimum. Thus the contrary must be true, that is, minimum severity delay products $\{s^{v_k} d_i$ $\forall 1 \leq k \leq M, 1 \leq i \leq N\}$ occur when delay is balanced, $d_i = d_{i+1}$ $\forall 1 \leq i \leq N-1$.

***Corollary 1 (higher AC first)*:** Minimum severity delay products $\{s^{v_k} d_i \; \forall 1 \le k \le M, 1 \le i \le N\}$ occur when higher priority AC has more load than lower priority AC, that is, $\sum_{k=1}^{M} \lambda_i^{v_k} < \sum_{k=1}^{M} \lambda_{i+1}^{v_k} \; \forall 1 \le i \le N-1$.

***Lemma 2 (higher severity first)*:** Minimum severity delay products $\{s^{v_k} d_i \; \forall 1 \le k \le M, 1 \le i \le N\}$ occur when higher severity vehicles have more load on higher priority ACs, that is, $\sum_{k=1}^{M} s^{v_k} \lambda_i^{v_k} < \sum_{k=1}^{M} s^{v_k} \lambda_{i+1}^{v_k} \; \forall 1 \le i \le N-1$.

***Proof by contradiction*:** Assume we have minimized each of the severity delay products in $\{s^{v_k} d_i \; \forall 1 \le k \le M, 1 \le i \le N\}$ and $\sum_{k=1}^{M} s^{v_k} \lambda_{i+1}^{v_k} < \sum_{k=1}^{M} s^{v_k} \lambda_i^{v_k} \; \forall 1 \le i \le N-1$. From Lemma 1, minimum severity delay products $\{s^{v_k} d_i \; \forall 1 \le k \le M, 1 \le i \le N\}$ occur when higher priority AC has more load than lower priority AC, that is, $\sum_{k=1}^{M} \lambda_i^{v_k} < \sum_{k=1}^{M} \lambda_{i+1}^{v_k} \; \forall 1 \le i \le N-1$, which contradicts $\sum_{k=1}^{M} s^{v_k} \lambda_{i+1}^{v_k} < \sum_{k=1}^{M} s^{v_k} \lambda_i^{v_k} \; \forall 1 \le i \le N-1$. Thus the contrary must be true, that is, minimum severity delay products $\{s^{v_k} d_i \; \forall 1 \le k \le M, 1 \le i \le N\}$ occur when higher severity vehicles have more load on higher priority AC, that is $\sum_{k=1}^{M} s^{v_k} \lambda_i^{v_k} < \sum_{k=1}^{M} s^{v_k} \lambda_{i+1}^{v_k} \; \forall 1 \le i \le N-1$.

***Corollary 2 (delay)*:** For $\boldsymbol{V} \neq \emptyset$, the delay on AC_i is as follows.

$$h(d,i) = \begin{cases} \dfrac{d\beta_i - 1}{\Delta t}, & \forall d > \dfrac{1}{\beta_i} \\ 0, & otherwise \end{cases} \tag{19}$$

$$g(\lambda,i) = \begin{cases} \dfrac{\Delta t\lambda + i - 1 - \dfrac{\sum_{j=1}^{i-1} \beta_j}{\beta_i}}{\left(\dfrac{\sum_{j=1}^{i-1} \beta_j}{\beta_i} + 1\right)\Delta t}, & \forall \lambda > 0 \\ 0, & otherwise \end{cases} \tag{20}$$

$$f(\lambda,i) = \frac{1 + \Delta t\lambda}{\beta_i} \tag{21}$$

Where, $f(\lambda, i)$ is the delay on AC_i for any given load λ, $g(\lambda, i)$ is the load on AC_i for delay equalization across AC_i to AC_l for any load λ and $h(d, i)$ is the load on AC_i for a given delay d. Then, d_i^{EQ} is the equalization delay for AC_i with load $\lambda - \sum_{j=i+1}^{N} h(d_j, j)$, such that, load $\sum_{j=i+1}^{N} h(d_j, j)$ has already been assigned to previous ACs.

$$d_i^{EQ} = f\left(g\left(\lambda - \sum_{j=i+1}^{N} h(d_j, j), i\right), i\right) \tag{22}$$

Therefore, the delay on an AC_i is:

$$d_i = \begin{Bmatrix} d_i^{EQ}, & d_i^{EQ} < T_i^{MAX} \\ T_i^{MAX}, & otherwise \end{Bmatrix} \forall 1 < i \leq N \tag{23}$$

$$d_1 = d_1^{EQ} \tag{24}$$

***Theorem (delay bounds)*:** For $\boldsymbol{V} \neq \emptyset$, bounded delay $d_i \leq T_i^{MAX}\ \forall 1 < i \leq N, \lambda^{v_j} = \lambda\ \forall 1 \leq j \leq M$, $\lambda_i^{v_k} = \begin{Bmatrix} 0, \\ \lambda^{v_k} \end{Bmatrix}$ is the load of vehicle v_k on AC_i, $d_i - f\left(\sum_{k=1}^{M} \lambda_i^{v_k}, i\right) \leq f(\lambda, i)\ \forall 1 < i \leq N$ and $f\left(\sum_{k=1}^{M} \lambda_1^{v_k}, 1\right) - d_1 \leq f((N-1)\lambda, 1)$.

***Proof by contradiction*:** Assume $d_i - f\left(\sum_{k=1}^{M} \lambda_i^{v_k}, i\right) > f(\lambda, i)\ \forall 1 < i \leq N$ is true. Then the load on AC_i is $\sum_{k=1}^{M} \lambda_i^{v_k} = \left\lfloor \frac{h(d_i, i)}{\lambda} \right\rfloor \lambda$, hence $h(d_i, i) - \left\lfloor \frac{h(d_i, i)}{\lambda} \right\rfloor \lambda \not> \lambda$. Therefore, the contrary must be true. Similarly, assume $f\left(\sum_{k=1}^{M} \lambda_1^{v_k}, 1\right) - d_1 > f((N-1)\lambda, 1)$ is true. Then the load on AC_l is $\sum_{k=1}^{M} \lambda_1^{v_k} = \left\lfloor \frac{h(d_1, 1)}{\lambda} \right\rfloor \lambda + (N-1)\lambda$, hence $\left\lfloor \frac{h(d_1, 1)}{\lambda} \right\rfloor \lambda + (N-1)\lambda - h(d_1, 1) \not> (N-1)\lambda$. Thus, the contrary must be true.

VI. OSD Traffic Distribution–Heuristic

We use lemmas and theorem from Section V to develop an efficient opportunistic service differentiation traffic distribution heuristic. The pseudo code for our OSD traffic distribution heuristic is presented in Figure 7. First, the OSD traffic distribution heuristic, aims to prioritize

vehicles based on their context severity metric, as in Lemma 2. Then, it assigns vehicles to higher priority AC first, from Corollary 1, to leverage the inherent lower delays, thus offering higher QoS, with respect to delay, to higher context severity vehicles.

This heuristic has a global snapshot of vehicles, their context severities, and a load. It begins by sorting vehicles with respect to severity, so that they can be assigned AC as in Lemma 2. The heuristic deduces the delay on ACs based on Corollary 2, Equations (23) and (24) and computes the maximum load (load cap), Equation (19), based on this delay. It assigns the vehicles to ACs, as in Corollary 1, until it reaches the load cap, before moving to the next best AC, with respect to priority for channel access. This ensures that the next best QoS is offered to vehicle traffic.

```
FUNCTION OSDHeuristic
  // sort vehicles based on context severity metric
  initialize vehLoadQueue
  FOR i = N to 1
     compute d_i // equation (23), (24)
     compute AC_i loadCap // equation (19)
     REPEAT
       // get vehicle load in queue without removing
       IF AC_i loadCap > vehLoadQueue.peek() THEN
         assign vehicle load to AC_i
         AC_i loadCap -= vehLoadQueue.remove() // dequeue
       ELSE
          AC_i loadCap = 0 // AC_i is full
       ENDIF
     UNTIL AC_i loadCap > 0 && !vehLoadQueue.isEmpty()
  ENDFOR
ENDFUNCTION
```

Figure 7. Pseudo code for OSD traffic distribution heuristic.

Generally, the optimal solution occurs when there is equalization in the delay across all ACs, with higher context severity load assignment to higher priority AC. This equalization delay, d_i^{EQ} for AC_i can be computed from Equation (22), where $h(d_j, j)$ is the load on AC_j with delay d_j $\forall i+1 \leq j \leq N$ and $g\left(\lambda - \sum_{j=i+1}^{N} h(d_j, j), i\right)$ is the portion of load on AC_i when the

load $\lambda - \sum_{j=i+1}^{N} h(d_j, j)$ has to be distributed amongst AC_i to AC_1 to achieve equalization in delay and $f\left(g\left(\lambda - \sum_{j=i+1}^{N} h(d_j, j), i\right), i\right)$ is the delay on AC_i with load $g\left(\lambda - \sum_{j=i+1}^{N} h(d_j, j), i\right)$. Therefore, delay for AC_i $\forall 1 < i \leq N$, either achieves equalization across AC_{i-1} to AC_1 or is bounded by T_i^{MAX} from constraint (15) given in Equation (23). However, the delay on the lowest priority AC is unbounded to accommodate the traffic overflow and is given in Equation (24). The OSD traffic distribution heuristic assigns entire vehicle load to an AC in order of context severity metric. This may not fully utilize an AC. However, the Theorem quantifies this loss in utilization and shows that, given equal load per vehicle and entire vehicle load assignment to an AC, each AC will be underutilized by atmost one vehicle load. Therefore, we have shown that our OSD traffic distribution heuristic is scalable and efficiently achieves suboptimal results *c.f.* Theorem (delay bounds), with respect to vehicle assignment to AC.

In the next chapter, we will show analytical and simulation results to demonstrate that when service differentiation in WAVE is enhanced by our OSD traffic distribution mechanism, we improve the QoS with respect to delay for ACs, and for vehicles with respect to context severity.

CHAPTER 4

RESULTS–OPPORTUNISTIC SERVICE DIFFERENTIATION

I. Analytical Results

In this section, we will compare the OSD enhanced WAVE with the classical WAVE and show that it not only outperforms classical WAVE but also accounts for network load, maximum allowable delay bounds on ACs and the critical context severity metric for VANET vehicles. We use the parameters presented in Table 1 ([45], [44], [46], [47]) for analytical comparison of classical WAVE with our optimal OSD traffic distribution formulation and heuristic. WAVE has been designed to operate at different data rates, 3-27 Mbps. In our analytical study, we use 3 Mbps, similar to studies in ([48], [49]). In these scenarios, each vehicle generated the same load comprising of randomly distributed payload across ACs. lp_solve [50] was used to solve the OSD LP formulation.

Figure 2 illustrates that the OSD scheme entails communication and computation complexities. However, as shown in Figure 8, OSD enhanced WAVE results in improvements in delay across ACs, when compared to classical WAVE. This is achieved by increasing the utilization of higher priority AC and leveraging their inherent lower delay. It is also evident that our heuristic achieves suboptimal results. Furthermore, as the number of vehicles increases, the load increases, yielding higher delay. However, OSD enhanced WAVE has a smaller rate of delay increase compared to WAVE.

Table 1
OSD and WAVE QoS Parameters

Parameter	Value
$packet\ size, data\ rate$	400 bytes, 3 Mbps
$\lambda^{v_i}\ (load\ per\ vehicle)$	10 packets/sec
$T_4^{MAX}, T_3^{MAX}, T_2^{MAX}$	70 ms, 120 ms, 180 ms
$AIFSN_4, AIFSN_3, AIFSN_2, AIFSN_1$	2, 3, 6, 9
$SlotTime, SIFSTime$	13 µs, 32 µs
$CW_4^{min}, CW_3^{min}, CW_2^{min}, CW_1^{min}$	3, 7, 15, 15
$CW_4^{max}, CW_3^{max}, CW_2^{max}, CW_1^{max}$	7, 15, 1023, 1023

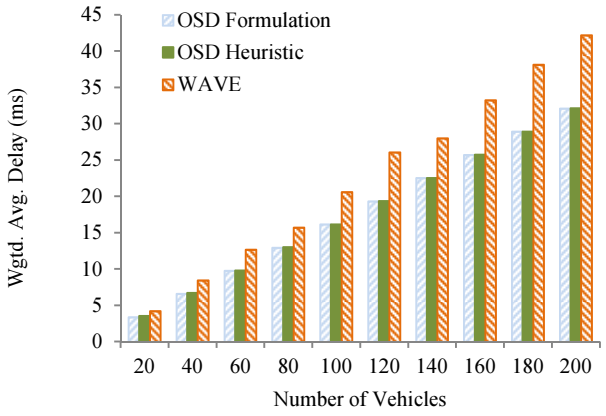


Figure 8. Delay comparison of OSD enhanced WAVE with classical WAVE.

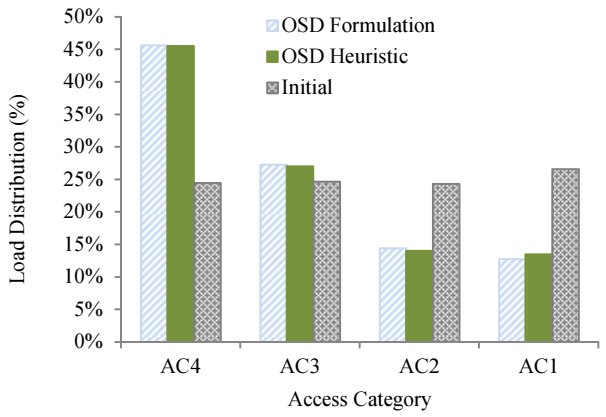


Figure 9. Load redistribution amongst ACs.

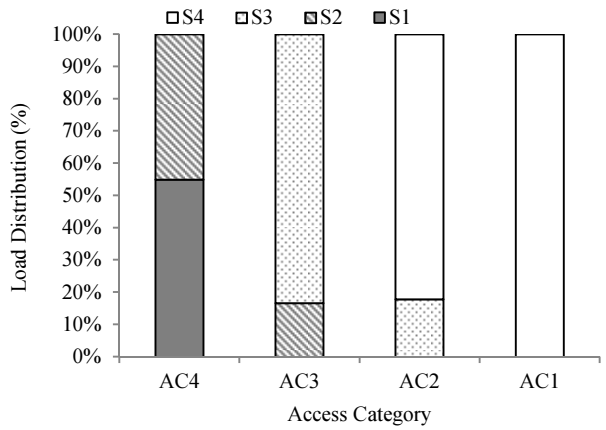


Figure 10. Load decomposition in AC, w.r.t severity, in OSD traffic distribution heuristic.

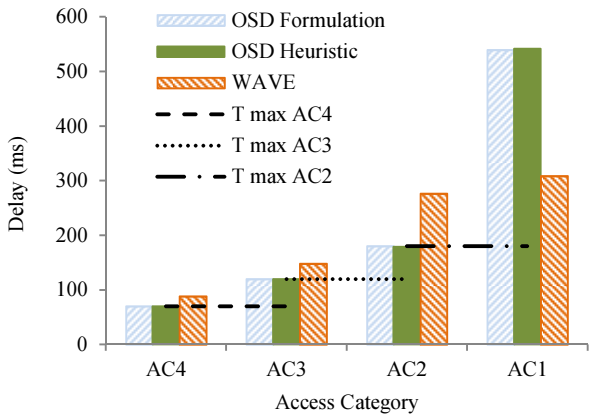


Figure 11. QoS w.r.t delay bounds on AC for 1000 vehicles.

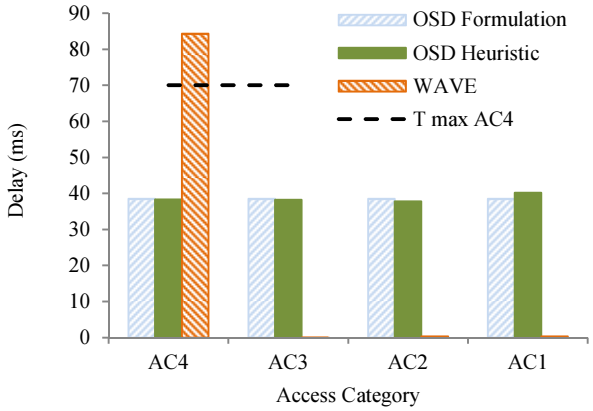


Figure 12. Traffic demotion to achieve better QoS w.r.t delay for 240 vehicles.

Figure 9 and Figure 10 illustrate opportunistic load redistribution for 200 vehicles and four context severities, S1, S2, S3 and S4, in decreasing order of severity. Note that higher priority ACs were assigned more traffic than lower priority AC. Figure 10 depicts that vehicles with higher context severity metrics were assigned to higher priority ACs. This ensures minimum severity delay products and suboptimal utilization of higher priority ACs.

Figure 11 shows how the OSD traffic distribution guarantees QoS with respect to delay for AC, bounded by T_i^{MAX} $\forall 1 < i \leq N$, while WAVE exceeds the maximum acceptable delay for AC_4, AC_3 and AC_2. Figure 12 depicts the benefit of using demotion of load to achieve better

overall QoS for all vehicles. Assume, only AC_4 has high load, OSD traffic distribution will opportunistically redistribute the load, systematically demoting it to lower priority ACs and improving QoS with respect to delay.

II. Simulation and Results

We use EstiNet 7.0 Network Simulator and Emulator [51], formerly NCTUns (National Chiao Tung University Network Simulator) [52]. It is a prevalent VANET simulator and emulator [53], which provides an integrated framework for traffic flow and network simulation. Typical simulators combine various functions, such as, links with varying delays and bandwidths, IP packet routers, TCP/IP hosts and mobility and network traffic generators into a single monolithic program. EstiNet consists of independent modules for these functions and executes them concurrently in a BSD UNIX environment [54]. In this section, we will delineate the scenario, simulation topology and analysis setup and present the performance comparison of classical WAVE with OSD enhanced WAVE in the VANET environment.

i. Scenario

Let us consider the performance of a safety application, such as, cooperative collision avoidance system, where OBU equipped vehicles communicate with each other and with the RSUs to collectively exchange data with neighboring nodes and infrastructure to circumvent accidents. In WAVE, VANET nodes interact with each other via services that can be produced or consumed. We scrutinize the V2V communication aspect of the safety application by deploying services on OBU equipped vehicles.

In WAVE, all vehicles prioritize their safety application packets, so that they are sent via AC_4. Whereas, OSD enhanced WAVE reprioritizes the application requested priorities based on the context severity of the vehicle that hosts the application. Table 3 lists the context severity of the vehicles. For simplicity, we map the continuous context severities into four discrete values, S_1, S_2, S_3 and S_4, arranged in order of decreasing severity.

Simulation logs and statistics are used to evaluate and, or infer performance and QoS metrics of throughput, packet collisions, packet loss and retransmission ratio and delay, to compare performance of classical WAVE with OSD enhanced WAVE.

ii. Topology

This V2V scenario is simulated using EstiNet, by randomly deploying agent-controlled ITS OBUs with 802.11p interface as vehicles. Each vehicle is deployed with a default application, known as, *CarAgent*, which manages the driving behavior of the vehicle, based on its profile, on the road infrastructure. A *car profile* is assigned to a vehicle and denotes its driving parameters, such as speed and acceleration. Each vehicle is equipped with an omnidirectional antenna and its physical layer parameters are configured to ensure that vehicles are within communication range of each other. Our vehicle parameters are listed in Table 2.

The V2V communication in the network is established by creating a subnet and assigning all vehicles to the same subnet. EstiNet uses this subnet assignment to generate IP and MAC addresses of the vehicles. Vehicles exchange data in the WAVE mode by joining a WAVE-mode Basic Service Set (WBSS). The service provider broadcasts its services in the control channel (CCH) via Wave Service Advertisements (WSA) and a consumer joins the WBSS of the provider. Since simulation time is directly proportional to the number of vehicles deployed, we simulated our scenario with 30 vehicles providing services to our consumer in vehicle ID 2, as illustrated in Figure 13. The simulation parameters are listed in Table 2. Similar to the work done in ([48], [49]) we use 3 Mbps data rate in our simulation.

Table 2
OSD Simulation Parameters

Parameters	Value
Vehicle	
$max\ speed$	18 meters/sec
$max\ acceleration$	1 meters/sec^2
$max\ deceleration$	4 meters/sec^2
$transmission\ power$	28.80814 dBm
$receiver\ sensitivity$	-82 dBm
Network	
$data\ rate$	3 Mbps
$load\ per\ provider$	15 packets/sec
$packet\ size$	400 bytes
WAVE	
$bandwidth$	10 MHz
$service\ channel\ ID$	172
$CCH\ interval$	50 ms
$SCH\ interval$	50 ms
$SlotTime$	13 μs
$SIFSTime$	32 μs
$AIFSN_4, CW_4^{min}, CW_4^{max}$	2, 7, 7 time slots
$AIFSN_3, CW_3^{min}, CW_3^{max}$	3, 15, 31 time slots
$AIFSN_2, CW_2^{min}, CW_2^{max}$	6, 31, 1023 time slots
$AIFSN_1, CW_1^{min}, CW_1^{max}$	9, 31, 1023 time slots
OSD	
$T_4^{MAX}, T_3^{MAX}, T_2^{MAX}$	70, 120, 180 ms
Simulation	
$simulation\ time$	10 sec
$simulation\ replications$	10
$service\ consumer$	1
$service\ providers$	30

The WAVE Short Message (WSM) application of EstiNet is added to vehicles to generate prioritized data for a consumer and transmit it over a service channel (SCH). Each service provider uses WSM to send 15 data packets/sec, of 400 bytes each, at a transmission data rate of 3 Mbps on SCH 172, via unicast communication. These network and WAVE parameters are listed in Table 2. Note we use EstiNet default WAVE parameters. Since we are simulating vehicles for a safety application, like a cooperative collision avoidance system, all service providers are generating data for the highest priority access category, AC_4.

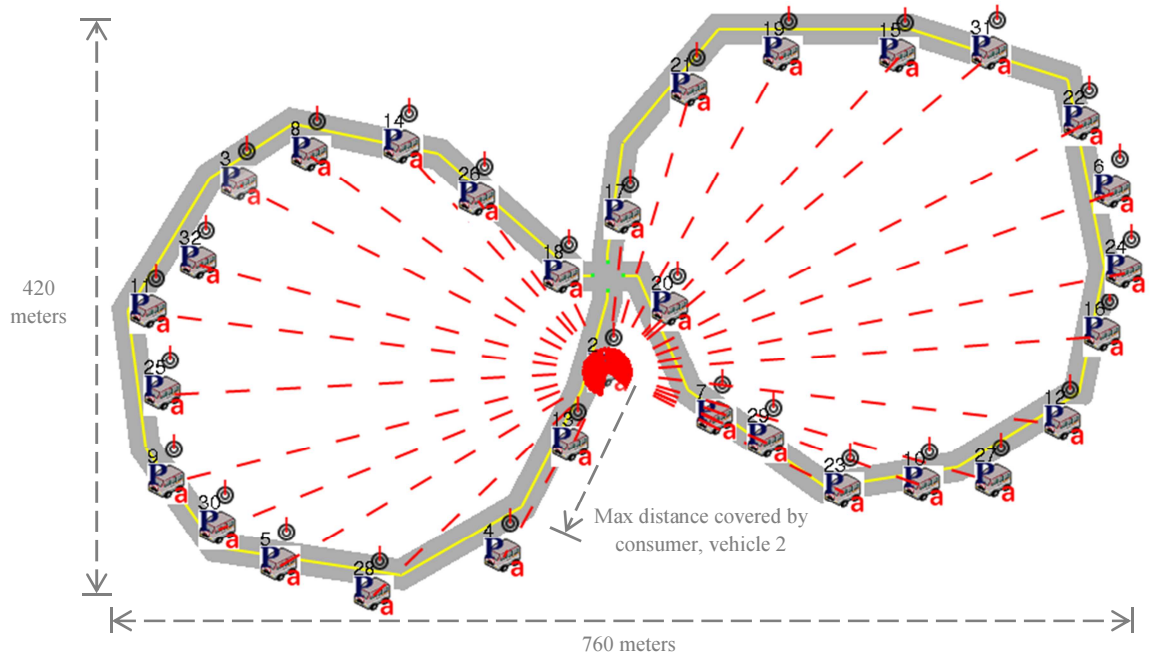


Figure 13. EstiNet simulation scenario consisting of 1 consumer (vehicle ID 2) and 30 provider OBU.

In EstiNet, priority 7 is used for AC_4. However, the heuristic for OSD enhanced WAVE, assigns vehicles to different access categories with respect to their severity, to ensure that access category delays do not exceed thresholds ($T_4^{MAX}, T_3^{MAX}, T_2^{MAX}$), delineated as OSD parameters in Table 2. This reassignment of vehicles to access categories based on severity is given in Table 3.

Table 3
OSD Vehicle Assignment to AC with Respect to Severity

		Access Category			
		AC_4 (priority 7)	**AC_3 (priority 5)**	**AC_2 (priority 3)**	**AC_1 (priority 2)**
Severity	**S_1**	7, 12, 14, 15, 28, 29			
	S_2	30	9, 10, 11, 24	4	
	S_3			5, 8, 16, 17, 18, 32	3, 6, 13
	S_4				19, 20, 21, 22, 23, 25, 26, 27, 31

iii. Analysis Setup

We monitor the network traffic using the packet trace file (*ptr*) logged by EstiNet and the packet statistics logging offered in the MAC module of a vehicle. The packet statistics logging records the out throughput (*KB/sec*) and the number of collisions at the providers. The *ptr* file records the network traffic and the packet statistics are sampled at 1 second intervals. The *ptr* file is used to deduce packet loss and packet retransmission ratio and log files for evaluating throughput and collisions. The QoS metric of delay perceived by a provider is inferred from the throughput.

There are multiple replications of a simulation run to increase the accuracy of our results. The packet loss (25) and retransmission ratio (26) are calculated using the average number of packet transmissions (*TX*), packet retransmissions (*RTX*) and packets received (*RX*). The average throughput and number of collisions for a provider are computed by excluding data from the warm-up and shut-down times, assumed to be 1 second each. The delay perceived by the providers is computed based on (27).

$$packet\ loss\ ratio = \frac{TX + RTX - RX}{TX + RTX} \tag{25}$$

$$packet\ retransmission\ ratio = \frac{RTX}{TX} \tag{26}$$

$$delay\ (ms) = \frac{1}{\frac{throughput\ (KB/sec) \times 1024}{packet\ size\ (bytes)}} \times 1000 \tag{27}$$

iv. Results and Discussion

Recall, that in our safety application simulation scenario, for classical WAVE, all 30 service providers are vehicles generating data for the highest priority access category, AC_4, and transmitting it in the same service channel, SCH 172. This simulates a high contention scenario, where vehicles are forced to compete for the physical channel. It ensures that vehicles use the

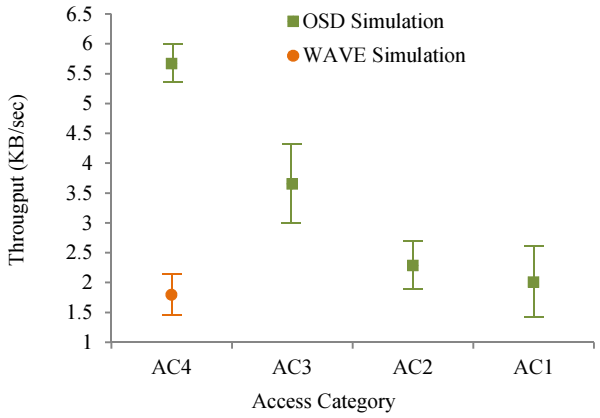
EDCA service differentiation settings in WAVE to backoff and retry for the channel. EDCA parameters for AC_4 have been selected so that vehicles in AC_4 have highest priority to access the channel and thus, incur the lowest latency. However, this service differentiation in WAVE does not account for network traffic and load on access categories, and hence cannot guarantee delay bounds for time critical safety applications. We will show that OSD enhanced WAVE accounts for network traffic and load on access categories to fully utilize all access categories and can also guarantee meeting delay requirements for high priority access categories.

In our high contention scenario for WAVE, all vehicles, providing services in AC_4, are constantly contending for the channel. This is evident in the high 383 packets/sec average collisions in AC_4, in Fig. 14(c). When the vehicles incur a collision, they backoff and retry for the channel. In the case of a collision, AC_4 is given priority, to reduce latency for AC_4 traffic. However, in this scenario, all vehicles contending for the channel belong to AC_4 and hence the priority of the access category does not benefit the vehicles. This is illustrated in the high retransmission ratio of 2.6 in Fig. 14(e), which captures the multiple retransmissions that must occur for one successful transmission. The packet loss ratio in Fig. 14(d) for vehicles providing services in AC_4, in classical WAVE, also concurs with the high collision rate and retransmission ratio.

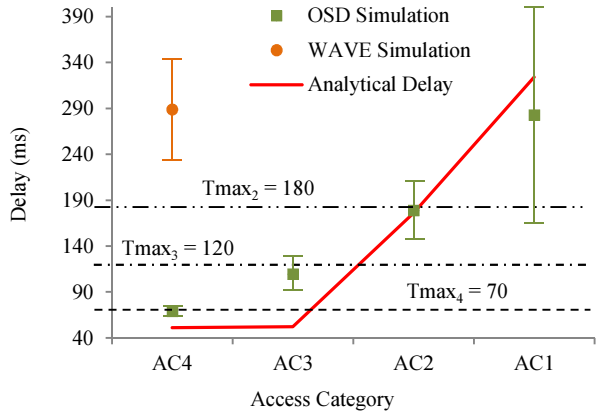
These collisions and retransmissions reduce the effective throughput of the vehicles providing services in AC_4, as depicted in Fig. 14(a), where the average classical WAVE throughput is low at 1.8 KB/sec. A low throughput, translates to a high delay for vehicles operating in the highest priority access category, AC_4. This undermines service differentiation in classical WAVE and diminishes the effectiveness of time-critical safety and lifesaving applications, like cooperative collision avoidance application. Fig. 14(b) shows that vehicles, in classical WAVE, providing services in AC_4 incur an average delay of 288 ms, which exceeds T_4^{MAX}= 70 ms. Typically, the maximum allowable latency requirement for safety applications, like cooperative collision avoidance is approximately 100 ms ([47], [46]). As illustrated, we

overcome this limitation of WAVE, using opportunistic service differentiation. We also show that our analytical delay closely follows the simulation delay, thereby validating our delay model.

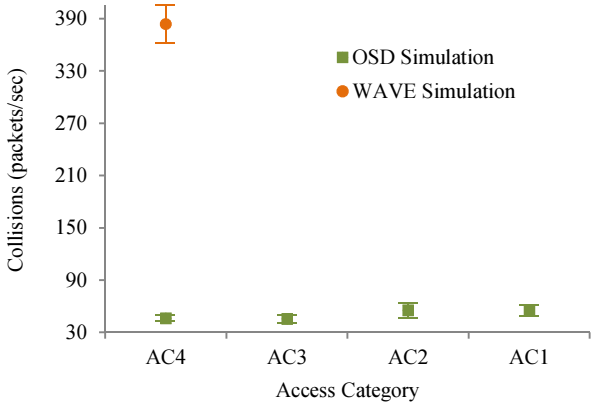
In our novel approach, we not only consider the load in the network and in the ACs but also account for the context severity of the vehicle and the maximum allowable delay requirements. Recall, our heuristic assigns a vehicle and its entire load to an access category, with respect to its context severity and the delay bounds on the access categories. This assignment of vehicles to access category with respect to severity is delineated in Table III. The heuristic assigns the vehicles to the access categories, based on the network load and delay bounds of 70, 120 and 180 ms, on AC_4, AC_3 and AC_2, respectively.



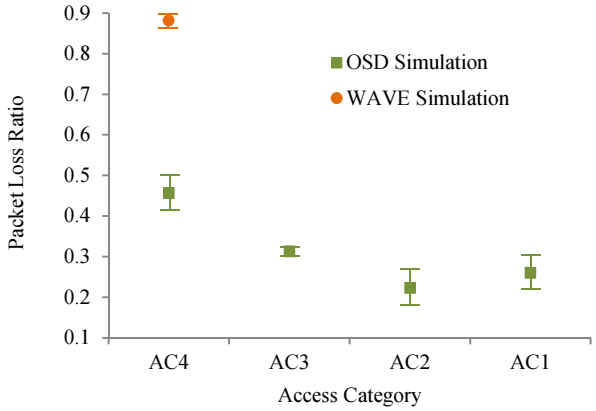
(a) OSD redistributes load and improves throughput when compared to WAVE.



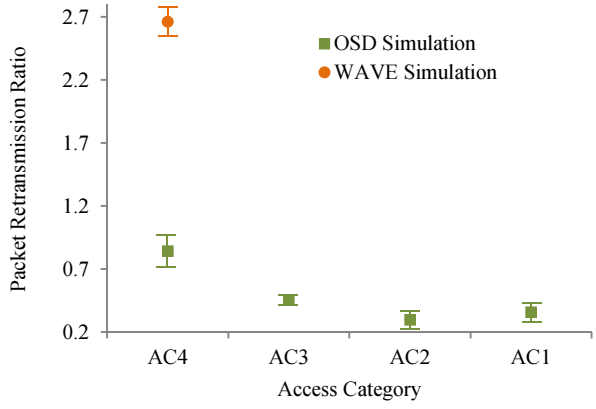
(b) Analytical delay model follows simulation results and OSD overcomes the unbounded delay in WAVE with Tmax for AC_4, AC_3 and AC_2.



(c) Load distribution across ACs utilizes inherent service differentiation in WAVE to reduce contention and collision.



(d) High throughput AC_4 shares higher packet loss ratio.



(e) Packet retransmission ratio consequentially follows packet loss ratio.

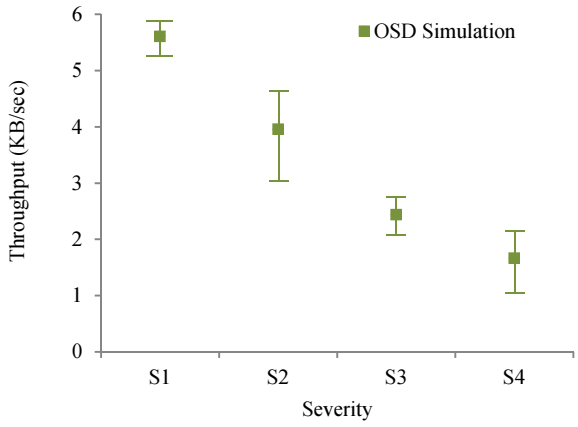
Figure 14. Simulation performance comparison of classical WAVE with OSD enhanced WAVE with 95% confidence intervals.

OSD enhanced WAVE distributed AC_4 load amongst the other access categories to utilize the inherent service differentiation mechanism of WAVE, so that there is differentiation in the backoff and retry mechanism of all the service providers. This reduces the average collision, packet loss and retransmission ratios, as illustrated in Fig. 14(c, d and e), respectively. Consequently, as vehicles incur less contention on the channel, their throughput increases, as depicted in Fig. 14(a). It is important to note that in OSD enhanced WAVE some vehicles were assigned to lower priority access category to achieve the delay guarantees in AC_4, AC_3 and AC_2.

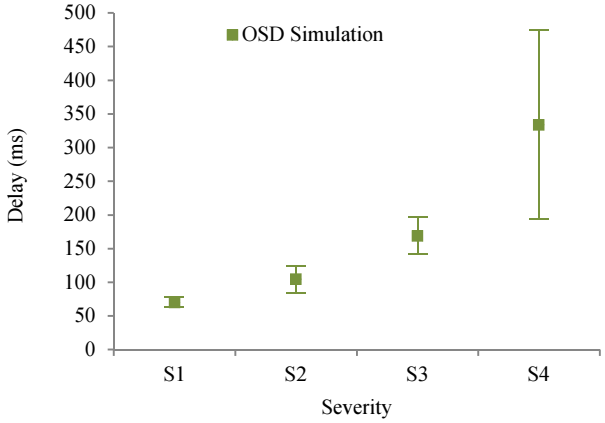
Fig. 14(b) captures the innovation and essence of our OSD technique. It not only guarantees delay bounds on the three higher priority access categories but also improves the QoS metric of delay perceived by all service providers in the network. It decreases the average delay for service providers in AC_4 from 288 ms to 70 ms from classical WAVE to OSD enhanced WAVE, respectively. Now, the safety application packets from service providers in AC_4 can ensure that latency does not exceed the maximum allowable delay of 70 ms. Fig. 14(b) explicitly illustrates that even service providers in AC_3 and AC_2 in OSD enhanced WAVE incur lower latency than service providers in AC_4 in classical WAVE. The worst case delay incurred by service providers in AC_1 is approximately the same as the average delay incurred by service providers in AC_4 in classical WAVE.

Therefore, in our simulation scenario, classical WAVE performs worse than the lowest priority access category in OSD enhanced WAVE. However, it is important to note that this is not our claim. In fact, the performance of OSD enhanced WAVE primarily depends on the delay bounds set on the access categories and the network load. If there is a low network load or no delay bounds on the ACs, OSD enhanced WAVE distributes the load such that the delay incurred by all vehicles across all access categories is equalized. In Fig. 14, we show that OSD enhanced WAVE can provide delay guarantees for ACs and improve QoS metric of delay for all vehicles with a 95% confidence interval range.

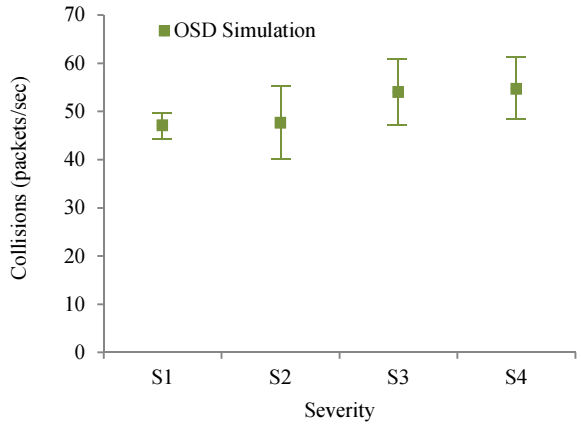
Table III lists the vehicle IDs of the service providers in the four different severities. The severity of a vehicle is different from the priority of the application or service the vehicle is providing.



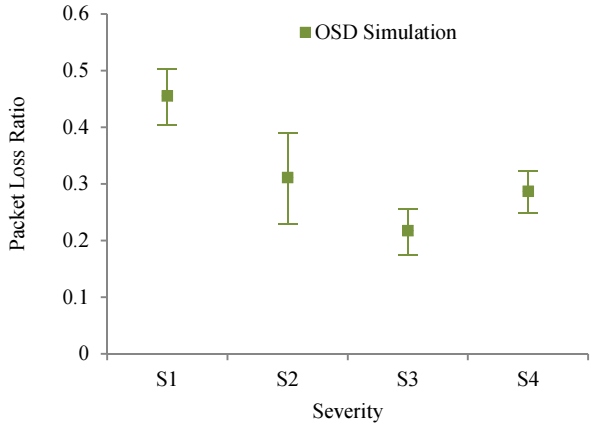
(a) ACs are utilized in order of context severity metric.



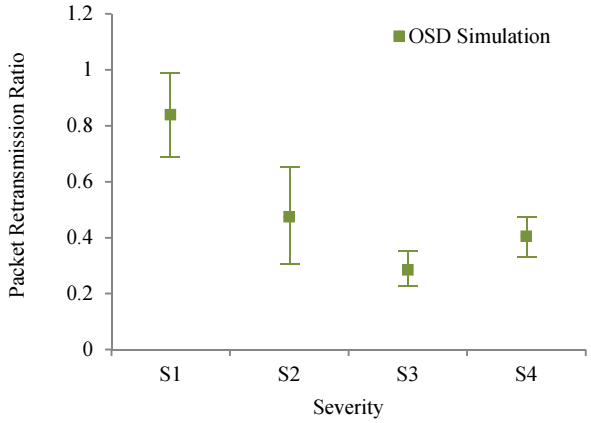
(b) Higher context severity metric vehicles have smaller delay.



(c) All ACs are optimally utilized to distribute load and share channel.



(d) Higher context severity metric vehicles are assigned to higher throughput ACs, sharing a higher percentage of packet loss.



(e) Packet retransmission ratio is consequential to the packet loss ratio.

Figure 15. Performance of vehicles with respect to severity in OSD enhanced WAVE with 95% confidence intervals.

Therefore, a low severity vehicle can generate high priority data. OSD enhanced WAVE distributes traffic across the ACs with respect to vehicle severity, in an effort to provide the best possible QoS to the higher severity vehicles. The distribution of vehicles based on severity amongst the ACs, allows us to provide service differentiation in WAVE based on the critical ITS parameter of vehicle severity. Our OSD technique for WAVE ensures highest throughput and lowest delay for the highest severity vehicles in Figure 15(a) and (b), respectively. The 95% confidence intervals are also illustrated for average collisions, packet loss and packet retransmission ratios for the four severity levels, in Figure 15(c, d, and e), respectively.

CHAPTER 5

RSU CLOUD RESOURCE MANAGEMENT

I. RSU Cloud Architecture

In this section, we will discuss the architecture of our RSU cloud, implemented as a Software Defined Network (SDN). The standard protocol for SDN is OpenFlow [15], which dictates communication primitives. There are two communication planes, the physical data plane and an abstracted control plane. This decoupling of control and forwarding planes enable the deep programmability of SDN. A SDN consists of OpenFlow enabled switches and controllers, where a switch contains data forwarding rules and controller has dynamic global network interconnection knowledge. Each switch maintains flows that pertain to data forwarding. Switches receive flow rules, proactively or reactively, from controllers, via the control plane. This separation of data and control plane enables SDNs to be dynamically reconfigured [55].

Recall, ITS aims to increase in-vehicle productivity, where users can subscribe for convenience and infotainment services such as remote vehicle diagnostics, on-the-go-Internet, online gaming, multimedia streaming, voice over IP. As illustrated in Figure 17, RSU clouds include traditional RSUs and micro-datacenters that will host the services to meet the demand from the underlying OBUs in the mobile vehicles. Traditional RSUs are fixed road-side infrastructure that can perform vehicle-to-infrastructure (V2I) and vehicle-to-vehicle (V2V) communication using the Wireless Access in Vehicular Environments (WAVE) standard. A fundamental component to RSU clouds is the RSU micro-datacenter.

A RSU micro-datacenter, illustrated in Figure 16, is a traditional RSU with additional hardware and software components that can offer virtualization and communication capabilities in a SDN. In comparison to traditional clouds, RSU cloud datacenters are resource constrained. The micro-datacenter hardware consists of a physical small form factor computing device and an OpenFlow switch. The software components on the computing device include the host operating

system and a hypervisor. A hypervisor is a low-level middleware that enables virtualization [38] of the physical resources. This allows abstraction of various virtual machines (VMs) on a single device. It is a technique widely employed in traditional datacenters that improves resource utilization, portability and fault tolerance [38]. In this manner, VMs can host services by efficiently sharing resources. VMs also enable service migrations and replications onto other VMs on disparate physical devices.

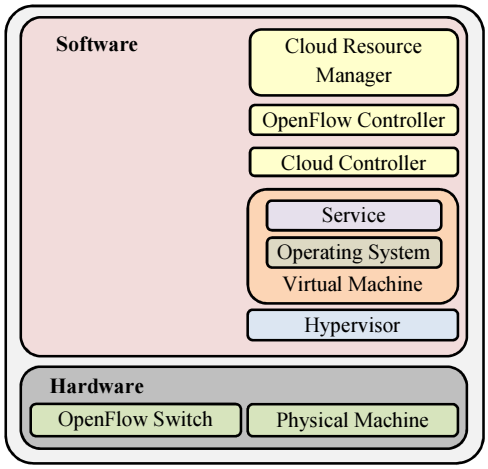


Figure 16. RSU Micro-datacenter architecture.

Optionally, one or more of the micro-datacenters will have additional software components, namely, OpenFlow Controller(s), Cloud Controller(s) and RSU Cloud Resource Manager(s). Our novel RSU Cloud Resource Manager (CRM) will communicate with OpenFlow and Cloud controllers, via the data plane, to disseminate information regarding service hosting, service migration and, or data flow changes, as illustrated in Figure 17. In the data plane, Cloud Controllers will govern service migration and hypervisors to instantiate new VMs hosting services. Consequentially, OpenFlow Controllers will update their network knowledge and simultaneously update switch flow rules via control plane.

The dynamic service demands from the vehicles may require increasing or decreasing the number of micro-datacenters hosting the services or physically migrating the VMs hosting the services from one micro-datacenter to another via the data plane. Without loss of generality, we

interchangeably use VM migration and service migration. Though, we can reprogram the RSU clouds to dynamically update service hosting and data forwarding information, it is costly and deteriorates network performance [38], [55]. VM migration in the data plane, constrains the limited bandwidth in the RSU cloud and increases network link latency, whereas, updating data forwarding information increases control plane overhead [39]. Moreover, service providers incur the cost of service migration and, or replications, and service users experience the deterioration in QoS. Naïve approaches to hosting services across all micro-datacenters are too costly, since service providers rent cloud resources from cloud infrastructure providers. Our major contribution is the novel offline Cloud Resource Manager, which is responsible for making the optimal decisions regarding the time, location and number of services hosted and data forwarding, in the RSU Cloud.

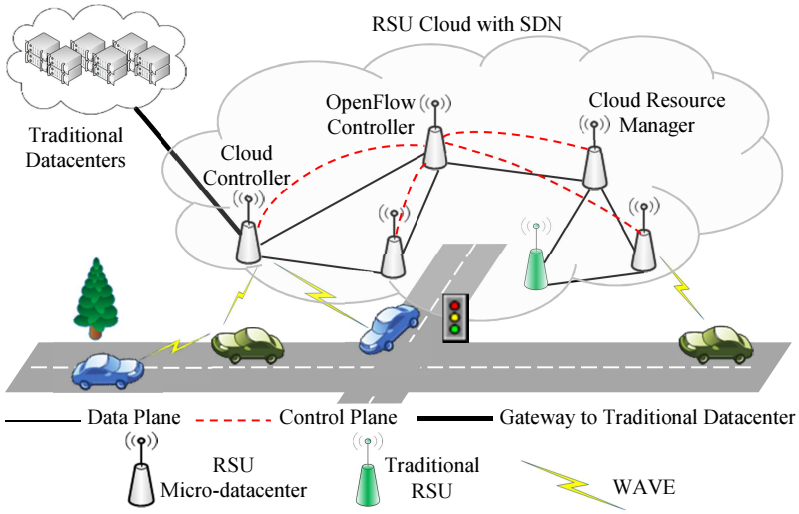


Figure 17. RSU Cloud architecture.

II. Reconfiguration Overhead Analysis in Mininet

In this section, we discuss how we emulated a RSU Cloud as a SDN in Mininet [16], to perform real-world reconfiguration overhead analysis. Recall, that SDN switches maintain data forwarding rules in flow tables. A flow table rule is a two tuple with a prefix and an action. A prefix contains, amongst other things, an ingress port, packet source and destination information.

A typical action specifies the egress port for the incoming packet at the switch. When a packet arrives at a switch it searches the prefixes in the flow tables and performs the action associated with the first matched prefix. To implement a RSU cloud SDN in Mininet, we implement a RSU micro-datacenter as a VM host connected to an OpenFlow switch, with a zero delay, as illustrated in Appendix A. Our RSU Cloud topology is inspired by Florida Department of Transportation (FDOT) deployment of RSUs [56].

It is evident from our topology that to enable real-world reconfiguration overhead analysis, we will have to support multipath. Therefore, we implemented stochastic switching for multipath in an OpenFlow enabled SDN. OpenFlow offers multipath through group tables, which enable data to be forwarded across multiple egress ports. In this case, a flow table rule *action* points to a group table identification number. A group table rule contains a list of buckets for egress ports for the same ingress port and source-destination pair. We control the data forwarding through these buckets by defining the group type. We use the *select* group type to stochastically select the buckets. The bucket weights are specified so that the traffic between the buckets is split according to the load on multiple paths between the source-destination.

We used Open vSwitch (OVS) 2.1 [57] that supports group tables and more specifically the optional *select* group type. However, OVS 2.1 implements the *select* group type by randomly selecting a group bucket, whereas, we want to select buckets stochastically. Therefore, we updated *xlate_select_group* and *group_best_live_bucket* functions on OVS 2.1 to support stochastic switching in accordance with group bucket weights. The Python scripts used for implementing the topology in Mininet and establishing the data forwarding rules, with detailed instructions are made available online for public access [58].

To perform reconfiguration overhead analysis, we implemented a SDN where each host has demands for a single service and there are a fixed number of service hosts. Figure 18 depicts two SDN over the same topology but on different configurations. Recall, a configuration is snapshot of hosts of service and the data forwarding rules in effect. In our analysis, the SDNs are

catering to an average demand of 70 Mbps and 90 Mbps, at time t and $t + 1$, respectively. We designed a joint optimization to find the optimal configuration that can meet the demand, while minimizing number of service hosts and cloud infrastructure delay. Figure 18 illustrates the configurations highlighting the switches with flow table rules, flow and group tables rules for multipath and service hosts. Our reconfiguration overhead heuristic counts the changes in the data forwarding rules and the number of service migrations. It is important to note that, deleting a host does not induce network traffic and therefore does not count as overhead.

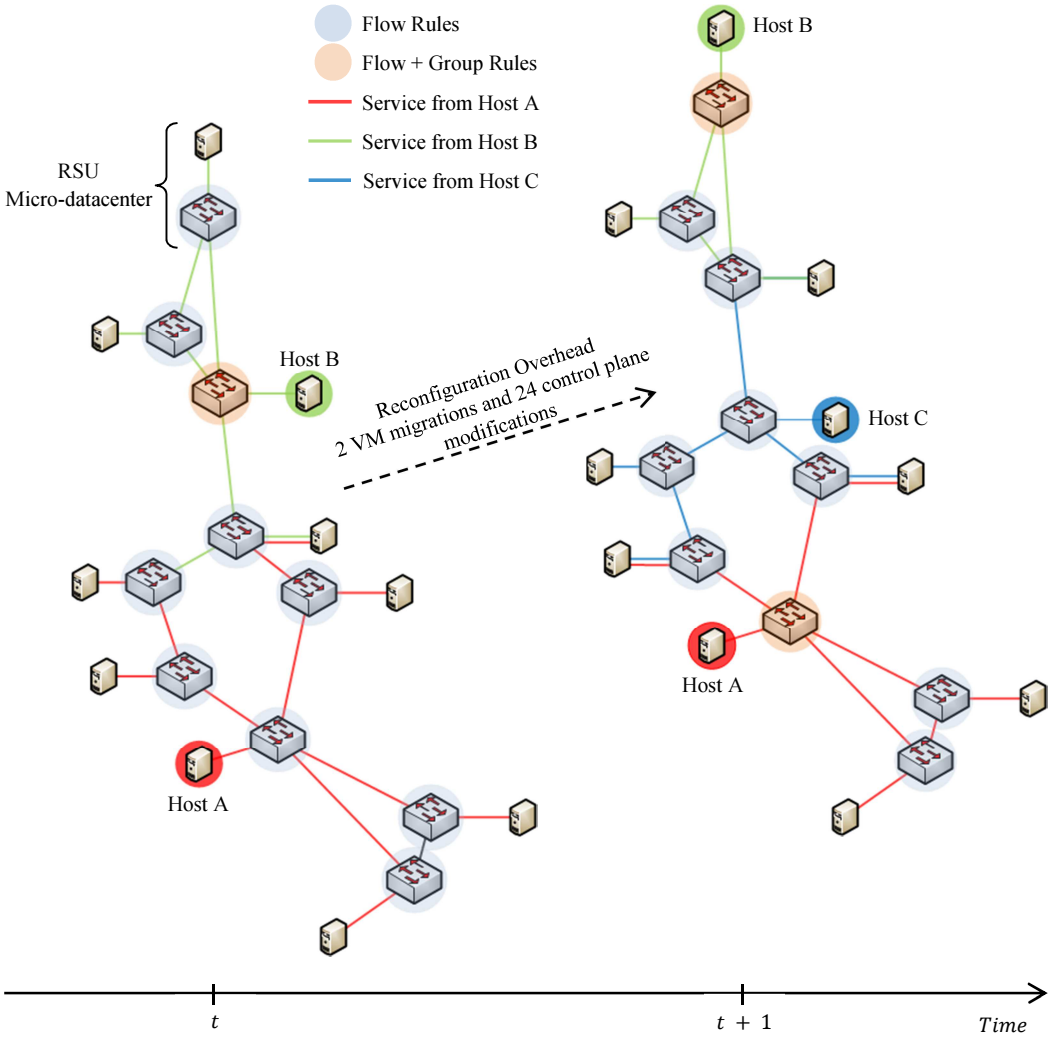


Figure 18. Reconfiguration overhead analysis in Mininet.

It is important to note that VM migrations induce network traffic in the data plane, whereas, flow and group table modifications makeup the control plane overhead. In the control

plane, there is a cost associated with adding and deleting flow or group table rules. A flow table and group table rule modification is counted doubly, a deletion of the old rule, followed by an addition of a new rule.

Based on our reconfiguration overhead analysis, we can formally define a configuration and reconfiguration overheads. A configuration is a network snapshot that records the service hosts and the data forwarding rules in the network. Data forwarding rules consist of flow rules and group rules. A configuration is defined as a three tuple $< \boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z} >$, where $\boldsymbol{X} = \{x_1, x_2, \dots, x_{|X|}\}$ is the set of service hosts, $\boldsymbol{Y} = \{y_1, y_2, \dots, y_{|Y|}\}$ is the set of flow rules and $\boldsymbol{Z} = \{z_1, z_2, \dots, z_{|z|}\}$ is the set of group rules.

Reconfiguration overhead consists of two components, number of VM migrations and control plane modifications. For simplicity, we assume that all services are hosted on VMs of the same size. Therefore, we simply count VM migrations. This can be extended to different size VM. Also, tearing down a VM does not induce network traffic and does not add to the reconfiguration overhead. Therefore, given sets of service hosts, X and X' for time t and $t + 1$, respectively, the VM migrations are equivalent to $|X - X'|$. On the other hand, all control plane modifications add to the reconfiguration overhead. Therefore, given sets of flow rules Y and Y' for time t and $t + 1$, respectively and group rules Z and Z', for time t and $t + 1$, respectively, the control plane overhead is calculated as in Equation (28). Intuitively, the control plane overhead is the sum of flow rules to be deleted and added, group rules to be deleted and added, flow rules changed to group rules and group rules changed to flow rules.

$$\begin{aligned} &control\ plane\ overhead \\ &\quad = |Y - Y'| + |Y' - Y| + |Z - Z'| + |Z' - Z| + |Y \cap Z'| + |Z \cap Y'| \end{aligned} \tag{28}$$

III. RSU Cloud Resource Management Model

Recall, our RSU cloud consists of RSU micro-datacenters that can host services to meet the demands from the vehicles. A configuration takes a snapshot of service hosts and data

forwarding rules. Also, over time, the change in demands requires costly reconfigurations to service hosting, service migration and, or replications and data forwarding rules. These reconfigurations increase service latency and deteriorate user experience [55], [38]. We can identify patterns of average demand in the network and classify them according to the time of the day.

The cloud resource management problem is the selection of a good configuration that minimizes reconfiguration overhead. In this section, we will present the problem statement and the cloud resource management model.

i. Problem Statement

Given a network graph $G = (\boldsymbol{V}, \boldsymbol{E})$, set of services $\boldsymbol{S}$, set of average demands $\boldsymbol{D} = \{d_{t_1}, d_{t_2}, \dots d_{t_{|T|}}\}$ over time period $\boldsymbol{T} = \{t_1, t_2, \dots t_{|T|}\}$ with an initial configuration $\psi^{t_1} = \langle \boldsymbol{X^{t_1}}, \boldsymbol{Y^{t_1}}, \boldsymbol{Z^{t_1}} \rangle$ for demand d_{t_1}at time t_1. The set $\boldsymbol{V}$ represents the RSU micro-datacenters interconnected by the edges in $\boldsymbol{E}$, each with bandwidth capacity $C_e \; \forall \; e \in \boldsymbol{E}$. At time $t_i \in \boldsymbol{T}$, there is a demand $b_{n,k}$ for service k, $k \in \boldsymbol{S}$ at node n, $n \in \boldsymbol{V}$ and the average demand in the network is d_{t_i}. Find a Pareto Optimal configuration from a set of Pareto Optimal Frontier of configurations $\boldsymbol{\Psi^{t_i}}$ to minimize the number of VM migrations $\langle \psi_j^{t_i} | \min\left(X_j^{t_i} - \boldsymbol{X^{t_i-1}}\right), \forall \, \psi_j^{t_i} = < X_j^{t_i}, Y_j^{t_i}, Z_j^{t_i} >, \; \psi_j^{t_i} \in \boldsymbol{\Psi^{t_i}} \rangle$. Each $\psi_j^{t_i} \in \boldsymbol{\Psi^{t_i}}$, optimally hosts services within threshold ϕ_k to meet service demands, while achieving a load-balanced network, and minimizing infrastructure delay with QoS threshold ω_k for each service k.

ii. Delay Model

In this subsection, we discuss how we compute the cloud infrastructure delay. The delay is based on a Lookup Table (LUT) with interval φ, which controls the granularity. The granularity of the LUT is a tradeoff to performance. We compute the delay on a path as a

summation of the delays on the edges in the path. The delay on an edge is the summation of processing (T_p), queuing (T_q), transmission (T_r) and propagation (T_g) delays.

Without loss of generality and similar to ([59], [60]), we currently use a G/G/1 queuing system. In practice the LUT table will be built over time, from experimental data. For modelling this, we assume a Poisson process for packet inter-arrival times λ and processing times μ, with mean and standard deviation, t_a, σ_a and t_s, σ_s, respectively. This assumption generally enables the problem formulation to be generic, suitable and adaptable to various different types of scenarios, including multimedia and network studies ([59], [60]). The coefficient of variation in inter-arrival times and packet processing times are $c_a = \sigma_a/t_a$ and $c_s = \sigma_s/t_s$, respectively. These are used in Kingsman formula [61] to approximate the queuing delay, in Equation (29). In our LUT, $c_a = 0.7$ and $c_s = 0.7$.

$$T_q = \frac{c_a^2 + c_s^2}{2} \cdot \frac{\lambda/\mu}{\mu - \lambda} \tag{29}$$

The transmission delay on the edge is based on the distance between the RSUs as in Equation (30). The length of the edge is based on the distance between the RSU nodes. Recall, our topology is inspired by Florida Department of Transportation (FDOT) RSU deployment [56], as illustrated in Figure 19. We use iTouchMaps [62] to estimate the latitude and longitude of the FDOT RSU locations and Coordinate Distance Calculator [63], to compute the distances between the RSU nodes. We use a processing delay of 10 μs and the propagation delay in Equation (31), for packet size 800 Bytes.

$$T_r = \left(\frac{length}{\frac{2}{3} \cdot speed\ of\ light} \right) \tag{30}$$

$$T_g = \left(\frac{packet\ size}{C_e} \right) \tag{31}$$

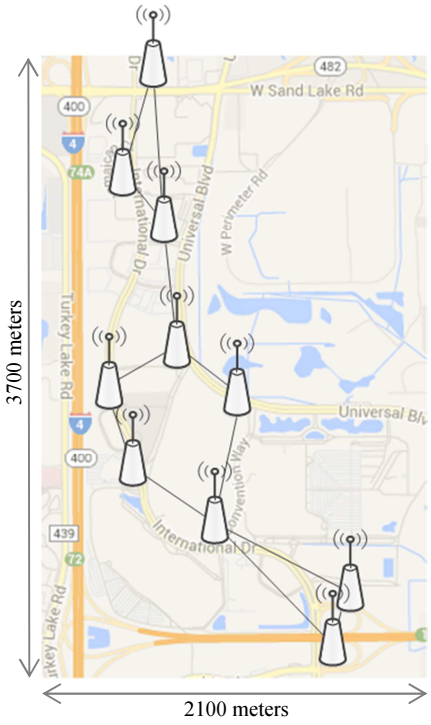


Figure 19. FDOT RSU deployment [56].

IV. Multi-Objective Integer Linear Programming Formulation

We model the RSU Cloud resource management problem as a multi-objective Integer Linear Programming (ILP) problem and solve it to obtain a Pareto Frontier. Below, we define and describe the input and output variables used in the formulation.

Input

S Number of services

N |V|, the number of RSUs

$b_{n,k}$ The demand at RSU n for service k, $\forall 1 \leq n \leq N, 1 \leq k \leq S$

$t_{n,k}$ $\begin{cases} 1, & if\ there\ is\ a\ demand\ for\ service\ k\ on\ RSU\ n, \forall 1 \leq n \leq N, 1 \leq k \leq S \\ 0, & otherwise \end{cases}$

$k_{m,n}$ The number of paths from RSU m to RSU n, $\forall 1 \leq m, n \leq N$

$p_e^{m,n,x}$ $\begin{cases} 1, & if\ RSUs\ m,n\ use\ path\ x\ with\ edge\ e, \forall 1 \leq m,n \leq N, 1 \leq x \leq k_{m,n}, \\ & 1 \leq e \leq |E| \\ 0, & otherwise \end{cases}$

C_e Bandwidth capacity of edge e

ϕ_k Threshold on number of service hosts for service k, $1 \leq k \leq S$

ω_k Threshold on infrastructure delay for service k, $1 \leq k \leq S$

φ $1 \leq \varphi < \min\{C_e, \forall 1 \leq e \leq |E|\}$, used to control the granularity of the LUT for edge e

$q_{i,e}$ The delay for load i on edge e, $\forall 0 \leq i \leq C_e, 1 \leq e \leq |E|$

$z_{m,k}$ $\begin{cases} 1, & if\ service\ k\ was\ previously\ hosted\ on\ RSU\ m, \forall 1 \leq m \leq N, 1 \leq k \leq S \\ 0, & otherwise \end{cases}$

$y_k^{m,n,x}$ $\begin{cases} 1, & if\ there\ was\ a\ control\ plane\ rule\ for\ path\ x\ between\ RSUs\ m,n\ for \\ & service\ k\ \forall 1 \leq m,n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \\ 0, & otherwise \end{cases}$

B A large constant

Output

$h_{m,k}$ $\begin{cases} 1, & if\ service\ k\ is\ hosted\ on\ RSU\ m, \forall 1 \leq m \leq N, 1 \leq k \leq S \\ 0, & otherwise \end{cases}$

$r_k^{m,n,x}$ The load carried on path x between RSUs m, n for service k, $\forall 1 \leq m,n \leq N$, $1 \leq x \leq k_{m,n}, 1 \leq k \leq S, 0 \leq r_k^{m,n,x} \leq \min\{C_e, \forall 1 \leq e \leq |E|, \exists p_e^{m,n,x} = 1,\}$

$a_k^{m,n,x}$ $\begin{cases} 1, & if\ there\ is\ a\ control\ plane\ rule\ for\ path\ x\ between\ RSUs\ m,n\ for \\ & service\ k\ \forall 1 \leq m,n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \\ 0, & otherwise \end{cases}$

l_e Load on edge e, $1 \leq e \leq |E|$

d_e Delay on edge e, $1 \leq e \leq |E|$

v_e Maps the load on edge e to a multiple of φ

$f_{e,i}$ $\begin{cases} 1, & if\ load\ on\ edge\ e\ is\ equal\ to\ i\ \times\ \varphi, \forall 0 \leq i \leq C_e, 1 \leq e \leq |E| \\ 0, & otherwise \end{cases}$

$g^{m,n,x}$ The delay on path x between RSUs m, n $\forall 1 \leq m,n \leq N, 1 \leq x \leq k_{m,n}$, $0 \leq g^{m,n,x} \leq j^{m,n,x}$

$w_k^{m,n,x}$ Ancillary variable for non-linear products of continuous variable $g^{m,n,x}$ with binary variable $h_{m,k}$, $\forall 1 \leq m,n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S$

$\alpha_{m,k}$ $\begin{cases} 1, & h_{m,k} - z_{m,k} > 0, \forall 1 \leq m \leq N, 1 \leq k \leq S \\ 0, & otherwise \end{cases}$

$\beta_k^{m,n,x}$ $\begin{cases} 1, & a_k^{m,n,x} - y_k^{m,n,x} > 0, \forall 1 \leq m,n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \\ 0, & otherwise \end{cases}$

$$\gamma_k^{m,n,x} \quad \begin{cases} 1, & y_k^{m,n,x} - a_k^{m,n,x} > 0, \forall 1 \le m,n \le N, 1 \le x \le k_{m,n}, 1 \le k \le S \\ 0, & otherwise \end{cases}$$

Our multi-objective is to minimize the reconfiguration overhead, pertaining to VM migration, control plane modifications, number of service installations and cloud infrastructure delays. The minimization in the reconfiguration overhead, is the sum of VM migrations and control plane modifications as in (32). We use weight ρ to control the priority of reconfiguration overhead, such that, minimizing VM migrations takes priority over minimizing control plane modifications.

$$min\left\{\sum_{m=1}^{N}\sum_{k=1}^{S}\rho\cdot\alpha_{m,k} + \sum_{m=1}^{N}\sum_{n=1}^{N}\sum_{x=1}^{k_{m,n}}\sum_{k=1}^{S}(1-\rho)\cdot\left(\beta_k^{m,n,x} + \gamma_k^{m,n,x}\right)\right\} \tag{32}$$

We also want to minimize the number of services hosts and achieve a load balanced network as in (33). Mathematically, to model delay on an edge with the number of service hosts, we normalize the delay, making this objective unit less [64]. We use weighted sum approach with weight P for this multi-objective optimization. To this end, we push the load across multiple paths, to minimize the infrastructure delay across all the edges d_e. Fundamentally, the load-balanced approach controls the delay on an edge from rising exponentially with load.

$$min\left\{\sum_{m=1}^{N}\sum_{k=1}^{S}P\cdot h_{m,k} + \sum_{e=0}^{|E|}(1-P)\cdot\frac{d_e}{q_{C_e,e}}\right\} \tag{33}$$

The minimization in infrastructure delay competes with the minimization of service hosts. Trivially, the services could be installed across all RSUs so that every demand is met by services hosted locally. However, this naïve approach would neither be efficient for "resource constraint" RSU clouds nor cost effective for service providers. Therefore, there is a direct tradeoff between number of service hosts and infrastructure delay. Furthermore, every reconfiguration incurs VM migrations and control plane modifications to the network. Therefore,

we have to ensure that we are not superfluous with service hosts. The VM migrations are counted as in Constraints (34) and (35).

$$B \cdot \alpha_{m,k} \geq h_{m,k} - z_{m,k} \qquad \forall\, 1 \leq m \leq N, 1 \leq k \leq S \tag{34}$$

$$h_{m,k} - z_{m,k} + \left(1 - \alpha_{m,k}\right) \cdot B \geq 0 \qquad \forall\, 1 \leq m \leq N, 1 \leq k \leq S \tag{35}$$

The control plane overhead is counted as the addition, $\beta_k^{m,n,x}$, and deletion, $\gamma_k^{m,n,x}$, of control plane rules counted as Constraints (36) through (41).

$$r_k^{m,n,x} \leq B \cdot a_k^{m,n,x} \qquad \forall\, 1 \leq m, n, m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{36}$$

$$r_k^{m,n,x} \geq a_k^{m,n,x} \qquad \forall\, 1 \leq m, n, m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{37}$$

$$B \cdot \beta_k^{m,n,x} \geq a_k^{m,n,x} - y_k^{m,n,x} \qquad \forall\, 1 \leq m, n, m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{38}$$

$$a_k^{m,n,x} - y_k^{m,n,x} + \left(1 - \beta_k^{m,n,x}\right) \cdot B \geq 0 \qquad \forall\, 1 \leq m, n, m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{39}$$

$$B \cdot \gamma_k^{m,n,x} \geq y_k^{m,n,x} - a_k^{m,n,x} \qquad \forall\, 1 \leq m, n, m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{40}$$

$$y_k^{m,n,x} - a_k^{m,n,x} + \left(1 - \gamma_k^{m,n,x}\right) \cdot B \geq 0 \qquad \forall\, 1 \leq m, n, m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{41}$$

The minimization objective in Equation (33) is subject to the following constraints. The demand should be met or exceeded by RSU service providers as in Constraints (42), (43) and (44). These constraints also ensure that only those paths carry network load that are between RSU service providers and RSU service consumers. Note, that RSU service providers can meet their demands locally.

$$\sum_{m,x} r_k^{m,n,x} \geq b_{n,k} \qquad \forall\, 1 \leq n \leq N, 1 \leq k \leq S \tag{42}$$

$$B.h_{m,k}.t_{n,k} \geq \sum_{x} r_k^{m,n,x} \qquad \forall\, 1 \leq m,n,m \neq n \leq N, 1 \leq k \leq S \tag{43}$$

$$h_{m,k}.t_{n,k} \leq \sum_{x} r_k^{m,n,x} \qquad \forall\, 1 \leq m,n,m \neq n \leq N, 1 \leq k \leq S \tag{44}$$

The load on a path is pushed to its edge and the delay on an edge is looked up in the LUT, using indexing, as in Constraints (45), (46), (47) and (48). The lookup can only happen in every load on the edge when the load matches the enumeration in the LUT.

$$l_e = \sum_{m,n,x,k} p_e^{m,n,x} \cdot r_k^{m,n,x} \qquad \forall\, 1 \leq e \leq |E| \tag{45}$$

$$l_e = \varphi.v_e \qquad \forall\, 1 \leq e \leq |E| \tag{46}$$

$$v_e = \sum_{i=0}^{C_e/\varphi} i.f_{e,i} \qquad \forall\, 1 \leq e \leq |E| \tag{47}$$

$$\sum_{i=0}^{C_e/\varphi} f_{e,i} = 1 \qquad \forall\, 1 \leq e \leq |E| \tag{48}$$

$$d_e = \sum_{i=0}^{C_e/\varphi} f_{e,i}.q_{i,e} \qquad \forall\, 1 \leq e \leq |E| \tag{49}$$

The delay on a path, $g^{m,n,x}$ is bounded by infrastructure delay threshold ω_k for service k, as in Constraints (50), (51), (52), (53) and (54).

$$g^{m,n,x} = \sum_{e=0}^{|E|} p_e^{m,n,x}.d_e \qquad \forall\, 1 \leq m,n,m \neq n \leq N, 1 \leq x \leq k_{m,n} \tag{50}$$

$$w_k^{m,n,x}.t_{n,k} \leq \omega_k \qquad \forall\, 1 \leq m,n,m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{51}$$

$$w_k^{m,n,x} \leq j^{m,n,x} \cdot h_{m,k} \qquad \forall\, 1 \leq m,n,m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{52}$$

$$w_k^{m,n,x} \leq g^{m,n,x} \qquad \forall\, 1 \leq m, n, m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{53}$$

$$w_k^{m,n,x} \geq g^{m,n,x} - j^{m,n,x}.\left(1 - h_{m,k}\right) \qquad \forall\, 1 \leq m, n, m \neq n \leq N, 1 \leq x \leq k_{m,n}, 1 \leq k \leq S \tag{54}$$

In Constraints (55) and (56), we bound the number of service hosts and ensure that every service is hosted.

$$\sum_{m=1}^{N} h_{m,k} \geq 1 \qquad \forall\, k \tag{55}$$

$$\sum_{m=1}^{N} h_{m,k} \leq \phi_k \qquad \forall\, k \tag{56}$$

We assume all RSUs have demands for all services k and are equally equipped to host any service k. Also, we assume that RSUs are fully capable to meet any demand. Therefore, we assume that the number of service hosts is directly proportional to financial cost of service hosting. This removes subjectivity from our results.

Recall, at time $t_i \; \forall \; t_i \in T$, the average network demand is $d_{t_i} \in D$, so, for every service k, we generate demand at every RSU n, that is normally distributed with mean d_{t_i} and standard deviation $\sigma = 0.05 \times d_{t_i}$ and record it in $b_{n,k}$. As previously described, we used the topology in Figure 19 and the LUT is generated for the edges in the topology.

There are various techniques for solving multi-objective linear and integer linear programming problems. One common approach is the weighted sum, where weights are used to control the priority of one objective with respect to another. Due to the intrinsically ordered nature of our problem, we are able to solve the multi-objective RSU Cloud resource management problem, by decomposing it, into a smaller dual objective ILP, Equation (33), to build a Pareto Optimal Frontier (POF) of configurations.

Next, we use the lp_solve [50] Linear Programming engine to partially solve our objective, by minimizing the number of service hosts and infrastructure delay $\sum_{m=1}^{N}\sum_{k=1}^{S} P \cdot h_{m,k} + \sum_{e=0}^{|E|}(1-P)\frac{d_e}{q_{C_{e,e}}}$, subject to constraints (42) through (56), with $P = 0$. For a given number of service host threshold, ϕ_k, the optimization will yield an optimal solution in the Pareto Optimal Frontier. We populate the Pareto Optimal Frontier Ψ^{t_i}, by adding optimal solution $\psi_j^{t_i}$ for each ϕ_k, such that, $1 \leq \phi_k \leq N$ and $\phi_k \in \mathbb{Z}^+$. We start at $\phi_k = N$ and continue until there is no infeasible solution. At the end, we will have $\psi_j^{t_i} \epsilon \Psi^{t_i}, \forall j$, ϕ_k at time t_i and average network demand is d_{t_i}.

Initially, we assume a fresh network, therefore no previous configurations exist, and thus we select Pareto Optimal with the minimum number of service hosts, as the initial configuration at t_0 for demand d_{t_0}. However, $\forall\, t_i > t_0$ there will be reconfigurations costs between Ψ^{t_i} and $\Psi^{t_{i-1}}$. We will select the Pareto Optimal $\left\{\psi_j^{t_i} | \min\left(X_j^{t_i} - X^{t_{i-1}}\right), \forall j, \psi_j^{t_i} \in \Psi^{t_i}\right\}$, such that, first it minimizes the difference in VM migrations, followed by control plane overhead, $\left\{\psi_j^{t_i} | \min\left(Y_j^{t_i} - Y^{t_{i-1}}\right), \forall j, \psi_j^{t_i} \in \Psi^{t_i}\right\}$, by controlling the weights in Equation (32).

V. RSU Cloud Resource Management–Heuristic

We design and implement a novel heuristic for the RSU cloud resource management problem. We will show that our heuristic efficiently yields suboptimal results, by *always* operating on the Pareto Optimal Frontier of non-dominated solutions. Our heuristic can be decomposed into two components, (1) generate POF, and (2) prune POF to find the configuration that minimizes VM migrations, followed by control plane overhead. It is important to note that for a given average demand d_{t_i}, we use the *same* demand ($b_{n,k}$) for each RSU as in the ILP. This ensures accurate comparison of the results from ILP optimization and our heuristic.

To generate POF, for each service $k \in S$, we begin by randomly selecting $\lceil N/2 \rceil$ nodes to be the RSUs hosting services and meet their demands locally. *For all,* the remaining RSUs, we randomly select a RSU $n \in V$ and satisfy its demand by *iteratively*, selecting a service host for a unit of its demand, such that, every unit of demand receives the current best infrastructure delay. This uniquely enables us to distribute load across the paths in the networks and achieve a load-balanced network. We repeat this, until all the RSUs demands have been satisfied. At the end, we have a configuration. We repeat this, to generate K configurations and select the configuration $\psi_j^{t_i} \in \Psi^{t_i}$, that minimizes infrastructure delay, to be included in the POF. For large values of K, we cover a large range of permutations, to increase the probability of finding a configuration that approximates the minimum infrastructure delay for service hosts.

Next, we reduce the number of service hosts by half, and repeat the process until the POF, Ψ^{t_i}, has been filled with all the valid configurations for average demand d_{t_i} at t_i. The POF generation takes $O(KlogN)$, making it scalable. Now, we have can generate Ψ^{t_i} for d_{t_i}, $\forall t_i \in T$ and select the configuration that trivially minimizes the number of VM migrations $\langle \psi_j^{t_i} | \min\left(X_j^{t_i} - X^{t_{i-1}}\right), \forall \psi_j^{t_i} = < X_j^{t_i}, Y_j^{t_i}, Z_j^{t_i} >, \psi_j^{t_i} \in \Psi^{t_i} \rangle$. We break a tie between configurations, be selecting the configuration with minimum VM migrations, followed by control plane modifications.

However, there is a short sightedness in this heuristic. Though, we minimize the reconfiguration overhead required to meet the change in demand from t_{i-1} to t_i , our heuristic applies a myopic approach to configuration selection. Consequentially, a configuration that minimizes the reconfiguration overhead from t_{i-1} to t_i, may not be the best configuration over the long term for the network. To overcome this, we employ reinforcement learning to select configurations that yield the optimal number of reconfigurations over the long term.

VI. Markov Decision Process

Thus far, the heuristic we use to select a configuration simply minimizes the number of VM migrations. This heuristic is shortsighted and seeks immediate gains, but lacks the long term knowledge to make an educated decision about the configuration to be selected such that a long term reduction in the number of VM migrations can be achieved. The configuration selection decision problem lends itself perfectly to the Markov Decision Process (MDP), where the outcome is partially random and controlled by a decision maker.

The MDP is a discrete time stochastic process, defined by a quad-tuple $< S, A, P, R >$, where S is the set of states and A is set of actions. The transition from state m to n is based on the action a, defined by the probability $P(m, n, a)$, with corresponding reward $R(m, n, a)$. The goal of the MDP is to find a "policy" that dictates the action to take in a state that maximizes the expected reward.

We design the configuration selection process as a MDP, in the following manner. First, every configuration from the POF $\psi_j^{t_i} \in \Psi^{t_i}$ from the heuristic, across all d_{t_i} for all $t_i \in T$ are enumerated as a list of states in the MDP. Next, a set of actions are defined. In our scenario, the number of RSUs $N = 10$, therefore, the heuristic finds configurations for 5, 3, and 2, service hosts. Then, the actions are defined as $A = \{a_1, a_2, a_3\}$ where the configuration with 2, 3 and 5 installations is selected, respectively. In MDP, we define state transitions using a probability matrix P, where we can only transition between a state, if it is a configuration in the next time instance, that is, $P(m, n, a), \forall m \in \Psi^{t_{i-1}}, n \in \Psi^{t_i}, a \in \text{A}$. Therefore, the configurations are chronologically ordered. The reward matrix R is populated as the complement of the VM migration costs. Therefore, the reward $R(m, n, a)$ is the difference between N, the maximum number of VM migrations, and number of VM migrations, $\left(X_j^{t_i} - X_k^{t_{i-1}}\right), \forall\, \psi_j^{t_i} = < X_j^{t_i}, Y_j^{t_i}, Z_j^{t_i} >,\ \psi_j^{t_i} \in \Psi^{t_i}, \psi_k^{t_{i-1}} = < X_k^{t_{i-1}}, Y_k^{t_{i-1}}, Z_k^{t_{i-1}} >, \psi_k^{t_{i-1}} \in \Psi^{t_{i-1}}$ and when moving from

configuration m to n on a is $R(m,n,a) = N - \left(X_j^{t_i} - X_k^{t_{i-1}}\right), \forall\, \psi_j^{t_i} = < X_j^{t_i}, Y_j^{t_i}, Z_j^{t_i} >,\ \psi_j^{t_i} \in \Psi^{t_i}, \psi_k^{t_{i-1}} = < X_k^{t_{i-1}}, Y_k^{t_{i-1}}, Z_k^{t_{i-1}} >, \psi_k^{t_{i-1}} \in \Psi^{t_{i-1}}$. This is delineated in Figure 20, below.

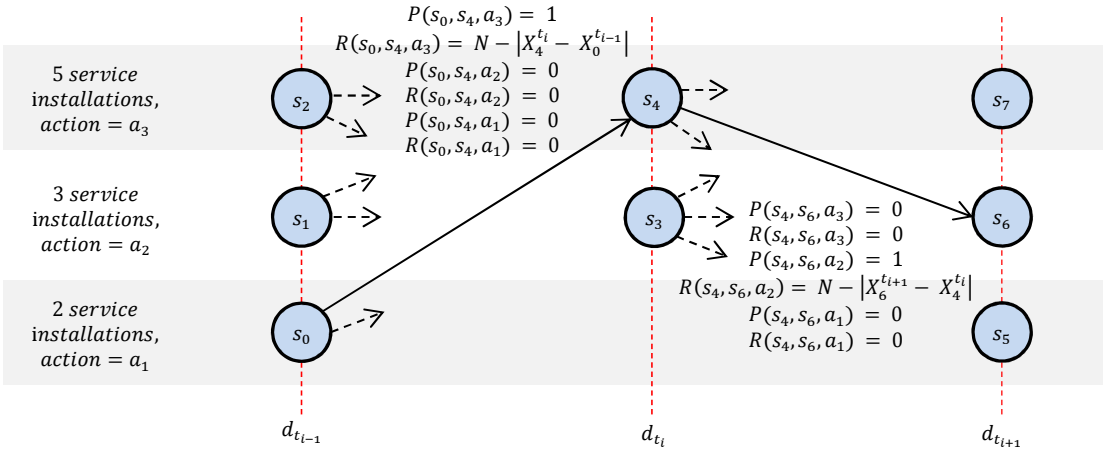


Figure 20. MDP for minimizing VM migrations.

Markov Decision Process can be solved using various techniques, such as, Q-learning, policy iteration, value iteration, linear programming, etc. We are interested in policy iteration to get the optimal policy. The optimal policy is one that has not changed in two successive iterations and maximizes the reward. The policy will dictate the action to take in a state. For this purpose, we use the MDP toolbox for MATLAB designed and developed by Chadès *et al.* [65]. The toolbox contains various techniques for solving MDP, including policy iteration. We use this toolbox with its policy iteration technique to generate a policy which will maximize reward.

Recall, our reward is the complement of the number of VM migrations therefore a policy that maximizes the reward value will minimize the number of VM migrations in the long term. This MDP derived policy for configuration selection will ensure that we select configurations that minimize the VM migrations over the long term, even though it may incur a higher VM migration cost at a time t_i, over the long run, it will incur the optimally minimal number of VM migrations.

CHAPTER 6

RESULTS–RSU CLOUD RESOURCE MANAGEMENT

I. Scenario–Topology and Analysis Setup

In this section, we will present and discuss our results. We setup the topology depicted in Figure 19 for evaluating RSU Cloud resource management, with services $S = 1$ and demands for these services at all the RSUs. We assume fast Ethernet connections between the RSUs, such that bandwidth capacity of each edge $C_e = 100Mbps, \forall 1 \leq e \leq |E|$ and the LUT interval $\varphi = 1$, so that we have a fine grain LUT. We run multiple iterations for the same network configuration. Recall, a network configuration, includes RSU host nodes, data forwarding rules and the loads carried on paths between the producer and consumer, for every service k with given demand $d_{t_i} \in D$. In our scenario, for a period T, the set of average demands $D = \{50Mbps, 60Mbps, 80Mbps, 70Mbps, 90Mbps, 50Mbps, 70Mbps\}$. In every iteration, we generate a demand at every RSU n, such that that is normally distributed with mean d_{t_i} and standard deviation $\sigma = 0.05 \times d_{t_i}$ and record it in $b_{n,k}$. We run 5 iterations to get the confidence intervals.

We compare our results with a purist approach. In the given problem, there are two possible purist approaches (1) Cost optimization, which optimally hosts services to meet network demands, as illustrated in Figure 21, irrespective to the infrastructure delay incurred by the services, and (2) Delay optimization, which optimally hosts services to minimize infrastructure delay, irrespective to the cost of hosting services. We employ a joint optimization that minimizes number of service hosts and infrastructure delay. Trivially, Delay optimization, would maximally deploy hosts, so that services are meet locally, incurring no infrastructure delay. Moreover, this would not be viable for RSU Cloud service providers. Therefore, we do not compare our Joint Optimization approach to Delay optimization.

II. Results and Discussion

The benefits of our Joint Optimization are presented in Figure 24, with improvements in orders of magnitude in infrastructure delay when compared with Cost Optimization. The reduction in cumulative VM migrations and control plane overhead is attributed to the fact that a purist approach, trying to only minimize the number of service hosts, is oblivious to the corresponding infrastructure delay. For example, in a purist approach, installing services in nodes m, n is the same as installing services in nodes i, j without regard to the infrastructure delay.

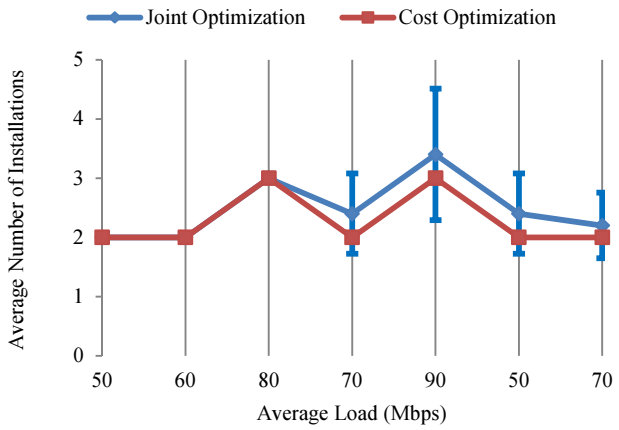


Figure 21. Cost optimization optimally hosts services.

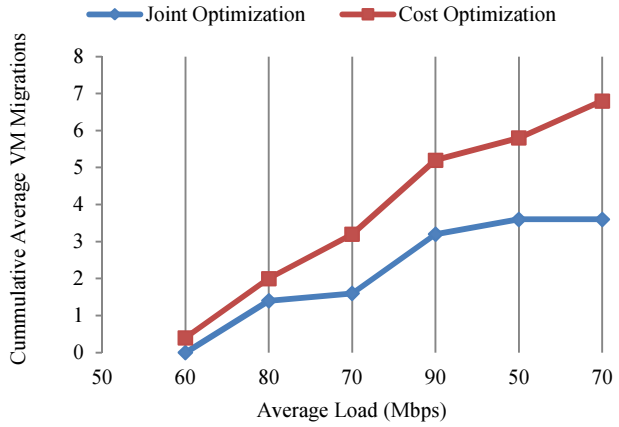


Figure 22. Joint optimization consistently incurs lower VM migrations.

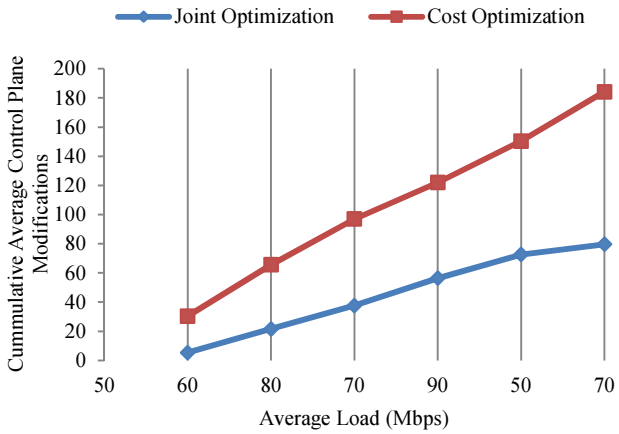


Figure 23. Cumulatively, Joint optimization incurs lower control plane modification with average demand d_{t_i} over time t_i.

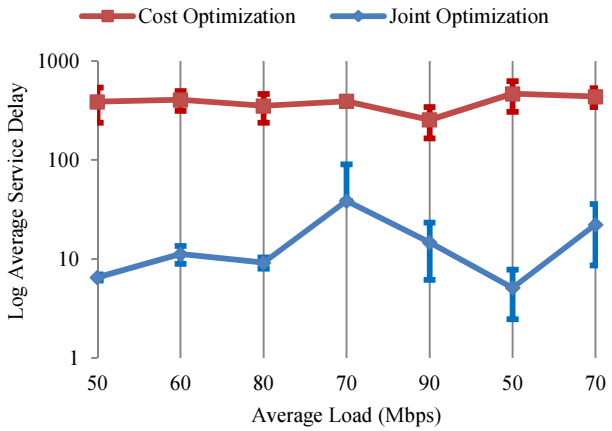


Figure 24. Joint Optimization has magnitudes lower infrastructure delay, over changing average demand d_{t_i}.

Figure 25 illustrates that each $\psi_j^{t_i} \in \Psi^{t_i}$ is a Pareto Optimal configuration that minimizes infrastructure delay and number of service hosts. Essentially, the heuristic selecting the configuration that minimizes VM migrations is always operating on the Pareto Optimal Frontier. Therefore, the final configuration selected *is* also an optimal configuration of the number of service hosts and infrastructure delay.

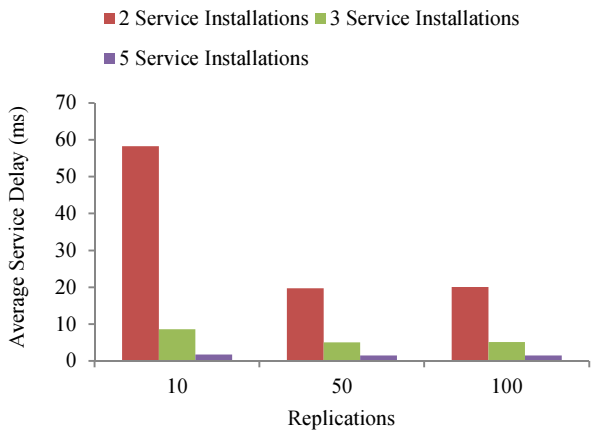


Figure 25. Every $\psi_j^{t_i} \in \Psi^{t_i}$ is a Pareto Optimal configuration with respect to number of service hosts and infrastructure delay.

In accordance with prediction intervals [66], as the number of replications, K, increases, our results show that the configurations are improving by minimizing VM Migrations Figure 26 and control plane overhead Figure 27. Due to the stochastic nature of the heuristic, Figure 28 and Figure 29 illustrate unpredictability; however, the fundamental insight is the improvement in confidence intervals as the number of replications increase to K. Therefore, we use the heuristic with K=100, as the best results to compare with Joint and Cost Optimization.

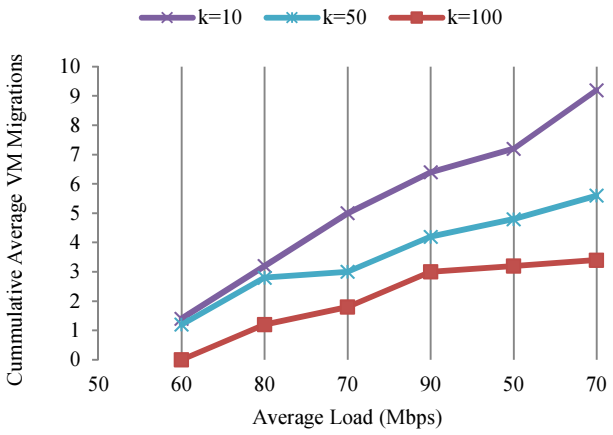


Figure 26. It is evident that higher number of replications K, yields better results.

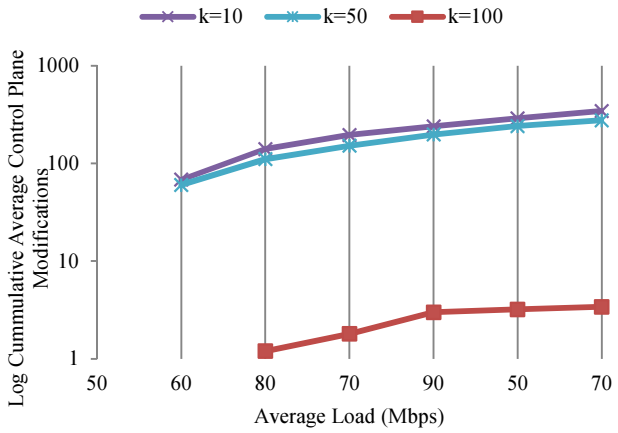


Figure 27. It is evident that higher number of replications K, yields better results w.r.t. cumulative average number of control plane modifications.

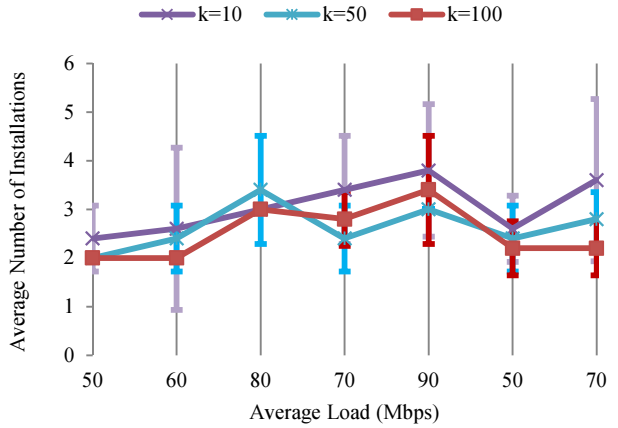


Figure 28. An increase in K the number of replications reduce margin of error, with respect to number of service hosts.

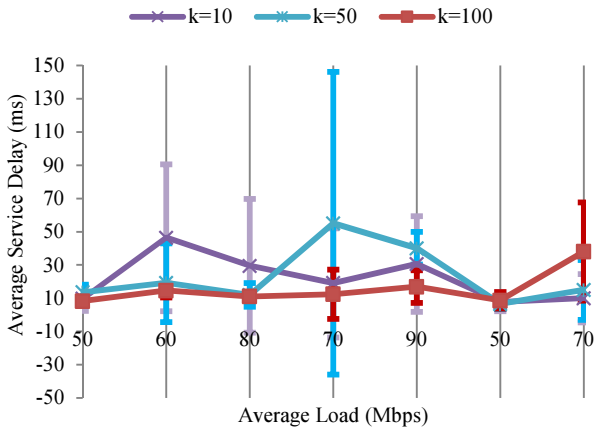


Figure 29. An increase in the number of replications reduce margin of error in the results, with respect to infrastructure delay.

Recall, we ran our Joint Optimization and Heuristic for 5 iterations. However, to ensure fair comparison of the heuristic and optimization, we consider the results from a single iteration, where each RSU n, has the same snapshot of demand $b_{n,k}$, for service k. Figure 32 illustrates how closely the Joint Optimization and Heuristic with K=100 perform when compared to Cost Optimization for the number of service hosts. However, the infrastructure delay in Cost Optimization is higher in magnitudes, when compared Heuristic with K=100 and Joint Optimization Figure 33. In Figure 33, Heuristic with K=100 outperforms Joint Optimization only with an increase in the number of services hosts, as illustrated in Figure 32.

Figure 30 depicts the significant improvement in the number of VM migrations with Heuristic K=100 and Joint Optimization. However, Figure 31, illustrates that Heuristic with K=100, performs the worst with respect to control plane modifications. This is because our load-balanced Heuristic distributes single units of load across paths until the demands are met. Each path accounts for numerous control plane modifications, as seen, in the high cost of control plane modifications. Our approach to increase the utilization of the number of paths between a provider and a consumer reduces bottlenecks and potential starvation of other RSUs, in contrast to the naïve approach of selecting the shortest path and fully utilizing the capacity of a path(s).

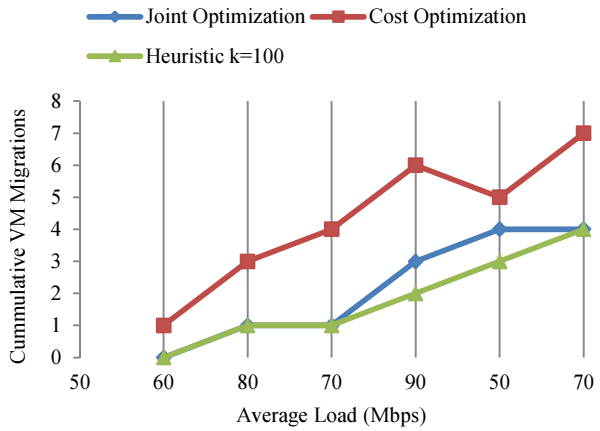


Figure 30. Heuristic with K=100 and Optimization outperform purist Cost Optimization.

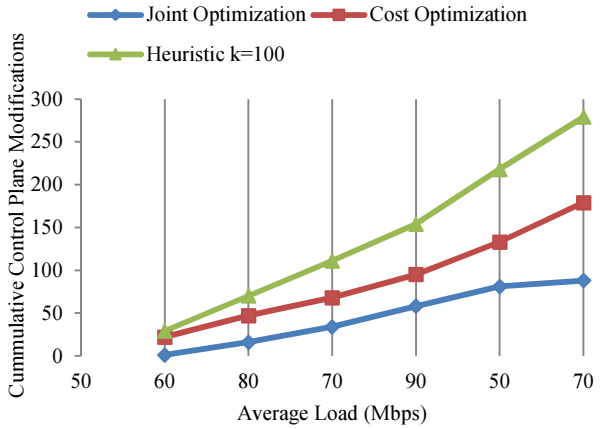


Figure 31. Heuristic incurs highest control plane modifications due to fine grain load balancing.

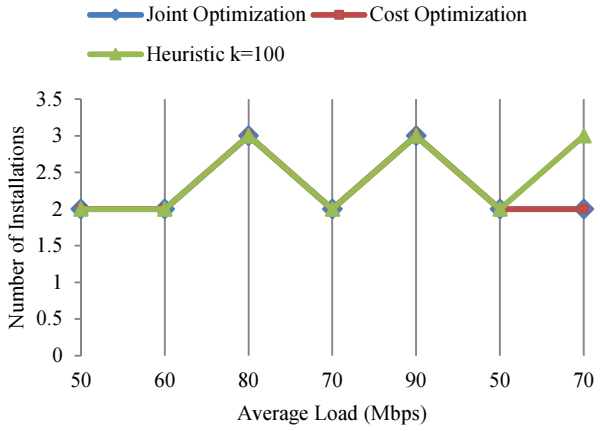


Figure 32. Purist Cost and Joint Optimization outperform Heuristic with K=100.

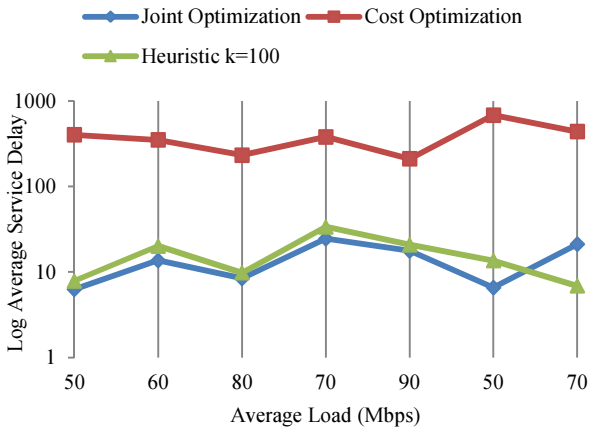


Figure 33. Heuristic with K=100 yields suboptimal infrastructure delay. Heuristic with K=100 outperforms Joint Optimization with an increase in the number of service hosts.

This is evident from Figure 34, Figure 35 and Figure 36 below. In different scenarios of the same problem, of configuration selection, we see that the configurations chosen by the heuristic alone performs as well as the reinforcement learning MDP. However, it is evident from Figure 36 that MDP can envision the long term benefit of choosing a higher VM migration cost at an average load 80 Mbps and incurs lower cumulative VM migrations over the long run. This is a novel contribution that enables configuration selections that minimize the VM migrations over the long term for service providers hosting non-safety services on the RSU Cloud.

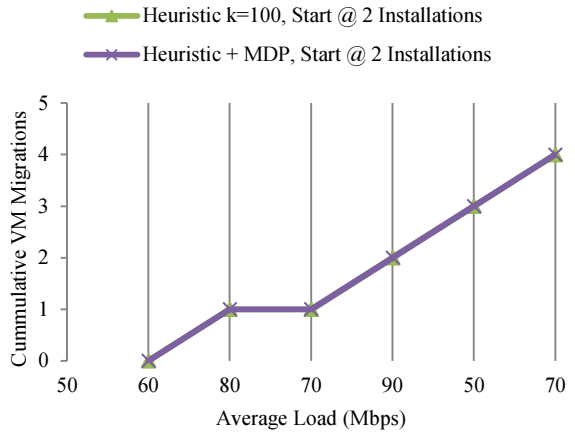


Figure 34. VM Migrations comparison for pure heuristic and heuristic followed by MDP, when starting with a 2 installation configuration.

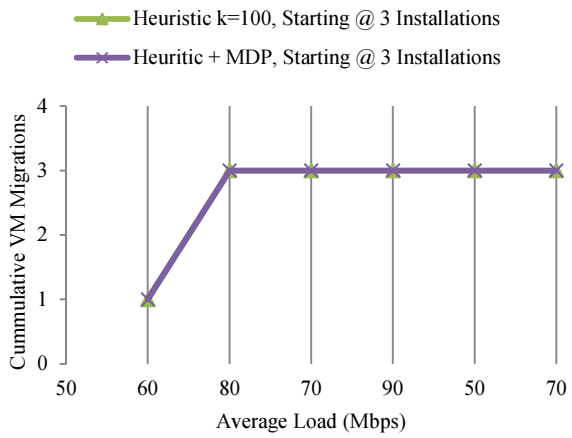


Figure 35. VM Migrations comparison for pure heuristic and heuristic followed by MDP, when starting with a 3 installation configuration.

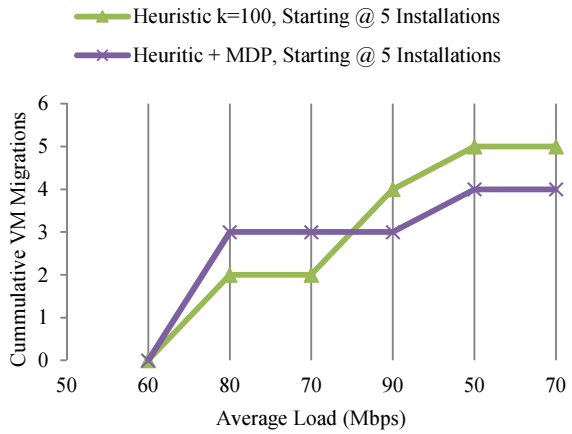


Figure 36. VM Migrations comparison for pure heuristic and heuristic followed by MDP, when starting with a 5 installation configuration.

CHAPTER 7

CONCLUSION–ENHANCED VEHICULAR APPLICATIONS

I. Opportunistic Service Differentiation for Safety Applications

We proposed a novel Opportunistic Service Differentiation (OSD) scheme to enhance WAVE. The proposed OSD is a traffic distribution mechanism that promotes service differentiation, while accounting for vehicular and communication network parameters. In this research, we emphasized the need for a context severity metric that can accurately depict the urgency of vehicle traffic. We proposed the use of a fuzzy inference system to deduce the severity of a vehicle, in the context to its surroundings, such as other vehicles and real-time road conditions. We use this context severity metric to provide service differentiation in WAVE, by using our OSD traffic distribution heuristic.

Our traffic distribution mechanism has multiple benefits over the classical WAVE. First, it assigns vehicle traffic to access categories (ACs) with respect to context severity. It provides higher QoS with respect to delay, for vehicles with higher context severity. Second, it apportions the ACs to vehicle traffic based on the network load, such that, it can guarantee delay bounds. For example, users of the OSD scheme can specify a delay bound for traffic in AC_4, the OSD traffic distribution heuristic will utilize this AC, while ensuring traffic in AC_4 meets the delay bound. This is a significant improvement over classical WAVE and its unbounded delay.

We also modeled the OSD traffic distribution mechanism as a linear programming problem to validate our claims. We derived lemmas and a theorem that form the basis for our traffic distribution heuristic. Our analytical comparisons show the improvement in QoS with the OSD enhanced WAVE and the suboptimal performance of the OSD traffic distribution heuristic. We also simulate a VANET with classical and OSD heuristic enhanced WAVE. The results show that when network traffic is opportunistically distributed amongst access categories, with respect

to context severity, it improves the QoS with respect to delay. This traffic distribution mechanism increases the utilization of ACs, while meeting upper bounds on AC delay.

II. RSU Cloud Resource Management for Non-Safety Applications

In this research, we propose a RSU Cloud to offer high QoS to non-safety applications in ITS. They will make an integral component for vehicular clouds and ITS non-safety applications. RSU Clouds consist of traditional RSUs and specialized RSUs containing micro-datacenters. The RSU Cloud is implemented as a Software Defined Network (SDN), which can host services to meet in-vehicle demands. In the event of inherent dynamic demands, RSU Clouds can be reconfigured to optimally meet the service demands. We study the effects of reconfiguration on the SDN, by implementing a SDN in Mininet. For real-world reconfiguration overhead analysis, we implemented a stochastic switching for multipath in OpenFlow-enabled SDN. We have made our contribution for implementing stochastic switching in Open vSwitch available online [58]. We formally define reconfiguration overhead, VM migrations and control plane modifications and instigate the need for efficient RSU Cloud resource management.

Our novel contribution is the architecture of RSU Cloud Resource Management (CRM) and RSU micro-datacenter. We model the CRM as a multi-objective optimization problem, for minimizing VM migrations, control plane overhead, number of service hosts and infrastructure delay. We designed a unique approach to solve the multi-objective so that we are continually operating on the Pareto Optimal Frontier. We selected an optimal configuration, such that, the VM migrations are minimized over time. We used reinforcement learning to select the configuration that minimizes VM migrations over the long run.

We illustrated how RSU Cloud resource management selects configurations that improve the infrastructure delay in orders of magnitude, with optimal number of service hosts. Over time and in face of dynamic loads, the configurations are selected as part of a Pareto Optimal Frontier, of non-dominated solutions. Any Pareto Optimal configuration is a candidate that can optimally,

with respect to infrastructure delay and number of service hosts, minimize VM migrations. To select the final configuration, we used our heuristic and reinforcement learning, to show that though some configurations may seem to immediately yield minimum VM migrations, over the long term, we may incur higher VM migrations.

III. Future Work

Our future work entails implementing the OSD traffic distribution mechanism and the OSD service for extensive simulation and real-time performance analysis with respect to QoS, heavy and light load and density of vehicles. Furthermore, a scrutiny of the scalability and latency in OSD traffic distribution mechanism would enable a detailed analysis for deployment of the OSD scheme.

In RSU Cloud resource management, our future work includes minimizing control plane modifications, an improved heuristic that incurs less control plane modifications and experimentation in Global Environment for Network Innovations (GENI) testbed.

In future, we will also explore architecture of RSU Cloud in support of OSD and evaluate its performance for vehicular applications. Traditional distributed systems problems, such as, consistent load snapshot problem, will be studies and addressed.

REFERENCES

[1] R. Uzcategui and G. Acosta-Marum, "WAVE: A Tutorial," *IEEE Communications Magazine,* vol. 47, no. 5, pp. 126-133, May 2009.

[2] J. Mittag, F. Schmidt-Eisenlohr, M. Killat, M. Torrent-Moreno and H. Hartenstein, "MAC Layer and Scalability ASpects of Vehicular Communication Networks," in *VANET: Vehicular Applications and Inter-Networking Technologies*, Hoboken, Wiley, 2009, pp. 219-273.

[3] IEEE Std 1609.4-2010(Revision of IEEE 1609.4-2006), "IEEE Standard for Wireless Access in Vehicular Environments (WAVE) -- Multi-channel Operation," 2011.

[4] Y. Morgan, "Managing DSRC and WAVE standards Operations in a V2V Scenario," *International Journal of Vehicular Technology,* vol. 2010, no. Article ID 797405, p. 18, 2010.

[5] V. Harigovindan, A. Babu and L. Jacob, "Ensuring fair access in IEEE 8002.11p-based vehicle-to-infrastructure," *EURASIP Journal on Wireless Communications and Networking,* vol. 2012, no. 168, 2012.

[6] R. Chen, D. Ma and A. Regan, "TARI: Meeting Delay Requirements in VANETs with Efficient Authentication and Revocation," in *International Conference on Wireless Access in Vehicular Environments*, 2009.

[7] M. Amadeo, C. Campolo and A. Molinaro, "Enhancing IEEE 802.11p/WAVE to provide infotainment applications in VANETs," *Ad Hoc Networks,* vol. 10, no. 2, pp. 253-269, 2012.

[8] M. A. Salahuddin, A. Al-Fuqaha and M. Guizani, "Exploiting Context Severity to Achieve Opportunistic Service Differentiation in Vehicular Ad hoc Networks," *IEEE Transactions on Vehicular Technology,* 2013.

[9] M. Amadeo, C. Campolo, A. Molinaro and G. Ruggeri, "A WAVE-compliant MAC Protocol to Support Vehicle-to-Infrastructure Non Safety Applications," in *IEEE International Conference on Communications Workshops (ICC'09)*, Dresden, Germany, 2009.

[10] C. Chrysostomou, C. Djouvas and L. Lambrinos, "Dynamically adjusting the min-max contention window for providing quality of service in vehicular networks," in *11th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net'12)*, Ayia Napa, Cyprus, 2012.

[11] E. Lee, E.-K. Lee and M. Gerla, "Vehicular Cloud Networking: Architecture and Design Principles," *IEEE Communications Magazine,* February 2014.

[12] R. Hussain, J. Son, H. Eun, S. Kim and H. Oh, "Rethinking Vehicular Communications: Merging VANET with cloud computing," in *IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, Taipei, 2012.

[13] K. Mershad and H. Artail, "Finding a STAR in a Vehicular Cloud," *IEEE Intelligent Transportation Systems Magazine,* vol. 5, no. 2, pp. 55-68, 2013.

[14] F. Bonomi, R. Milito, J. Zhu and S. Addepalli, "Fog computing and its role in the internet of things," in *In Proceedings of the first edition of the MCC workshop on Mobile cloud*

computing (MCC '12), Helsinki, 2012.

[15] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review,* vol. 38, no. 2, pp. 69-74, 2008.

[16] B. Lantz, B. Heller and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks," in *9th ACM Workshop on Hot Topics in Networks*, Monterey, 2010.

[17] Y. Zang, L. Stibor, B. Walke, H. Reumerman and A. Barroso, "A Novel MAC Protocol for Throughput Sensitive Applications in Vehicular Environments," in *IEEE Vehicular Technology Conference (VTC'07)*, Dublin, Ireland, 2007.

[18] Y. Qian, K. Lu and N. Moayeri, "A Secure VANET MAC Protocol For DSRC Applications," in *IEEE Global Telecommunications Conference (GLOBECOM'08)*, New Orleans, 2008.

[19] M. Nekovee, "Quantify Performance Requirements of Vehicle-to-Vehicle Communication Protocols for Rear-End Collision Avoidance," in *IEEE Vehicular Technology Conference (VTC'09)*, Anchorage, Alaska, 2009.

[20] J. Vardakas, I. Papapanagiotou, M. Logothetis and S. Kotsopoulos, "On the End-to-End Delay Analysis of the IEEE 802.11 Distributed Coordination Function," in *Internet Monitoring and Protection* , San Jose, 2007.

[21] H. Wu, X. Wang, Q. Zhang and X. Shen, "IEEE 802.11e Enhanced Distributed Channel Access (EDCA) Throughput Analysis," in *IEEE International Conference on Communications (ICC'06)*, Istanbul, 2006.

[22] C. Huang and W. Liao, "Throughput and Delay Performance of IEEE 802.11e Enhanced Distributed Channel Access (EDCA) Under Saturation Condition," *IEEE Transactions on Wireless Communications ,* vol. 6, no. 1, pp. 136-145, 2007.

[23] S. Malik, M. Shah, S. Khan, M. Jahanzeb, U. Farooq and M. Khan, "Performance Evaluation of IEEE 802.11p MAC Protocol for VANETs," *Australian Journal of Basic and Applied Sciences,* vol. 4, no. 8, pp. 4089-4098, 2010.

[24] S. Eichler, "Performance Evaluation of the IEEE 802.11p WAVE Communication Standard," in *IEEE Vehicular Technology Conference (VTC)*, Baltimore, 2007.

[25] N. Ferreira and J. Fonseca, "Performance Evaluation of IEEE 802.11p MAC Protocol for VANETs," in *IEEE Symposium on Communications and Vehicular TEchnology in the Benelux (SCVT'11)*, Ghent, 2011.

[26] Y. Wang, A. Ahmed, B. Krishnamachari and K. Psounis, "IEEE 802.11p Performance Evaluation and Protocol Enhancement," in *IEEE International Conference on Vehicular Electronics and Safety (ICVES'08)*, Columbus, Ohio, 2008.

[27] L. Zhou, Y. Zhang, K. Song, W. Jing and A. Vailakos, "Distributed Media-Service Scheme for P2P-based Vehicular Networks," *IEEE Transactions on Vehicular Technology,* vol. 60, no. 2, pp. 692-703, 2011.

[28] M. Di Felice, A. Ghandour, H. Artail and L. Bononi, "Enhancing the performance of safety applications in IEEE 802.11p/WAVE Vehicular Networks,," in *IEEE International*

Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM'12), San Francisco, 2012.

[29] M. Amadeo, C. Campolo and A. Molinaro, "Enhancing IEEE 802.11p/WAVE to provide infotainment applications in VANETs," *Ad Hoc Networks,* vol. 10, no. 2, pp. 253-269, 2012.

[30] M. Abuelela and S. Olariu, "Taking VANET to the clouds," in *In Proceedings of the 8th International Conference on Advances in Mobile Computing and Multimedia (MoMM '10)*, Paris, 2010.

[31] Y. Qin, D. Huang and X. Zhang, "VehiCloud: Cloud Computing Facilitating Routing in Vehicular Networks," in *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, Liverpool, 2012 .

[32] X. Meng, V. Pappas and L. Zhang, "Improving the Scalability of Data Center Networks with Traffic-aware Virtual Machine Placement," in *InfoCom*, San Diego, CA, 2010.

[33] D. Wu, J. He, Y. Zeng, X. Hei and Y. Wen, "Towards Optimal Deployment of Cloud-Assisted Video Distribution Services," *IEEE Transactions on Circuits and Systems for Video Technology,* 2013.

[34] J. Jiang, T. Lan, S. Ha, M. Chen and M. Chiang, "Joint VM placement and routing for data center traffic engineering," in *IEEE 2012 InfoCom*, Orlando, FL, 2012.

[35] F. Chang, R. Viswanathan and T. Wood, "Placement in clouds for application-level latency requirements," in *IEEE International Conference on Cloud Computing* , Honolulu4, 2012.

[36] K. Tran and N. Agoulmine, "Adaptive and Cost-effective service placement," in *IEEE GlobeCom*, Houston, TX, 2011.

[37] A. Sailer, M. Head, A. Kochut and H. Shaikh, "Graph-based Cloud Service Placement," in *2010 IEEE International Conference on Services COmputing*, Miami, FL, 2010.

[38] S. Anja and W. Dargie, "Does Live Migration of Virtual Machines Cost Energy?,," in *IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, Barcelona, 2013.

[39] A. Tootoonchian and Y. Ganjali, "HyperFlow: a distributed control plane for OpenFlow," in *In Proceedings of the 2010 internet network management conference on Research on enterprise networking (INM/WREN'10)*, San Jose, 2010.

[40] M. Elbes, A. Al-Fuqaha, M. Guizani, A. Rayes and J. Oh, "A New Hierarchical and Adaptive Protocol for Minimum-Delay V2V Communication," in *IEEE Global Telecommunications Conference (GLOBECOM'09)*, Honolulu, 2009.

[41] X. Chen, H. Refai and X. Ma, "A Quantitative Approach to Evaluate DSRC Highway Inter-Vehicle Safety Communication," in *IEEE Global Telecommunications Conference*, Washington, DC, 2007.

[42] J. Zhou and K. Mitchell, "A scalable delay based analytical framework for CSMA/CA wireless mesh networks," *Computer Networks,* vol. 54, no. 2, pp. 304-318, 2010.

[43] Y. Yao, L. Rao, X. Liu and X. Zhou, "Delay analysis and study of IEEE 802.11p based DSRC safety communication in a highway environment," in *IEEE INFOCOM*, Turin, Italy, 2013.

[44] C. Han, M. Dianati, R. Tafazolli, R. Kernchen and X. Shen, "Analytical Study of the IEEE

802.11p MAC Sublayer in Vehicular Networks," *IEEE Transactions on Intelligent Transportation Systems ,* vol. 13, no. 2, pp. 873-886, 2012.

[45] I. C. Society, "IEEE Std. for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements. Part 11: Wireless LAN MAC and PHY Specifications. Amendment 6: Wireless Access in Vehicular Enviro," IEEE Std 802.11p 2010, New York, 2010.

[46] M. Boban, G. Misek and O. K. Tonguz, "What is the Best Achievable QoS for Unicast Routing in VANETs?," in *IEEE GLOBECOM Workshops*, New Orleans, 2008.

[47] The CAMP Vehicle Safety Communications Consortium, "Vehicle Safety Communications Project - Task 3 Final Report - Identify Intelligent Vehicle Safety Applications Enabled by DSRC," National Highway Traffic Safety Administration, U.S. Department of Transportation, Washington D.C., 2005.

[48] Q. Wang, S. Leng, H. Fu and Y. Zhang, "An IEEE 802.11p-Based Multichannel MAC Scheme With Channel Coordination for Vehicular Ad Hoc Networks," *IEEE Transactions on Intelligent Transportation Systems,* vol. 13, no. 2, pp. 449-458, 2012.

[49] C. Campolo and A. Molinaro, "DREAM: IEEE 802.11p/WAVE extended access mode in drive-thru vehicular scenarios," in *IEEE International Conference on Communications (ICC)*, Ottawa, 2012.

[50] M. Berkelaar, K. Eikland and P. Notebaert, "Open source (Mixed-Integer) Linear Programming system (lp_solve)," GNU LGPL (Lesser General Public License), 2004.

[51] "EstiNet 7.0 Network Simulator and Emulator," EstiNet Technologies, 2012. [Online]. Available: http://www.estinet.com/products.php?lv1=1&sn=2. [Accessed 26 February 2013].

[52] S. Y. Wang, C. L. Chou, C. H. Huang, C. C. Hwang, Z. M. Yang, C. C. Chiou and C. C. Lin, "The Design and Implementation of the NCTUns 1.0 Network Simulator," *Computer Networks,* vol. 42, no. 2, pp. 175-197, 2003.

[53] F. J. Martinez, C. K. Toh, J. Cano, C. T. Calafate and P. Manzoni, "A survey and comparative study of simulators for vehicular ad hoc networks (VANETs)," *Wireless Communications and Mobile Computing,* vol. 11, no. 7, pp. 813-828, 2011.

[54] S. Y. Wang and H. T. Kung, "A simple methodology for constructing extensible and high-fidelity TCP/IP network simulators," in *INFOCOM*, New York, 1999.

[55] M. Bari, A. Roy, S. Chowdhury, Q. Zhang, M. Zhani, R. Ahmed and R. Boutaba, "Dynamic Controller Provisioning in Software Defined Networks," in *9th International Conference on Network and Service Management (CNSM)*, Zurich, 2013.

[56] T. E. a. O. Office, "Florida Department of Transportation," Florida Department of Transportation, [Online]. Available: http://www.dot.state.fl.us/trafficoperations/its/projects_deploy/cv/connected_vehicles-wc.shtm. [Accessed 18 April 2014].

[57] B. Pfaff, J. Pettit, T. Koponen, K. Amidon , M. Casado and S. Shenker, "Extending Networking into the Virtualization Layer," in *8th ACM Workshop on Hot Topics inNetworks*, New York, 2009.

[58] M. Salahuddin, "Stochastic Switching using Open vSwitch in Mininet," GitHub, 18 February 2014. [Online]. Available: https://github.com/saeenali/openvswitch/wiki/Stochastic-Switching-using-Open-vSwitch-in-Mininet. [Accessed 18 April 2014].

[59] A. Anttonen and A. Mammela, "Interruption Probability of Wireless Video Streaming With Limited Video Lengths," *IEEE Transactions on Multimedia,* 2013.

[60] T. Luan, L. Cai and Xuemin Shen, "Impact of Network Dynamics on User's Video Quality: Analytical Framework and QoS Provision," *IEEE Transactions on Multimedia,* vol. 12, no. 1, pp. 64-78, 2010.

[61] G. Curry and R. Feldman, Manufacturing Systems Modeling and Analysis, 2nd Edition, New York: Springer Heidelberg Dordrecht, 2011.

[62] "Latitude and Longitude of a Point," iTouchMap.com, 2014. [Online]. Available: http://itouchmap.com/latlong.html. [Accessed 19 April 2014].

[63] Boulter, "Coordinate Distance Calculator," [Online]. Available: http://boulter.com/gps/distance/. [Accessed 18 April 2014].

[64] R. Marler and J. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization,* vol. 26, no. 6, pp. 369-395, 2004.

[65] I. Chadès, M.-J. Cros, F. Garcia and R. Sabbadin, *Markov Decision Process (MDP) Toolbox,* INRA, 2009.

[66] J. Banks, J. Carson, B. Nelson and D. Nicol, Discrete-Event System Simulation (4th Edition), Prentice Hall, 2004.

[67] Y. Zhang, L. Stibor, B. Walke, H. Reumerman and A. Barroso, "A Novel MAC Protocol for Throughput Sensitive Applications in Vehicular Environments," in *IEEE Vehicular Technology Conference (VTC'07)*, Dublin, 2007.

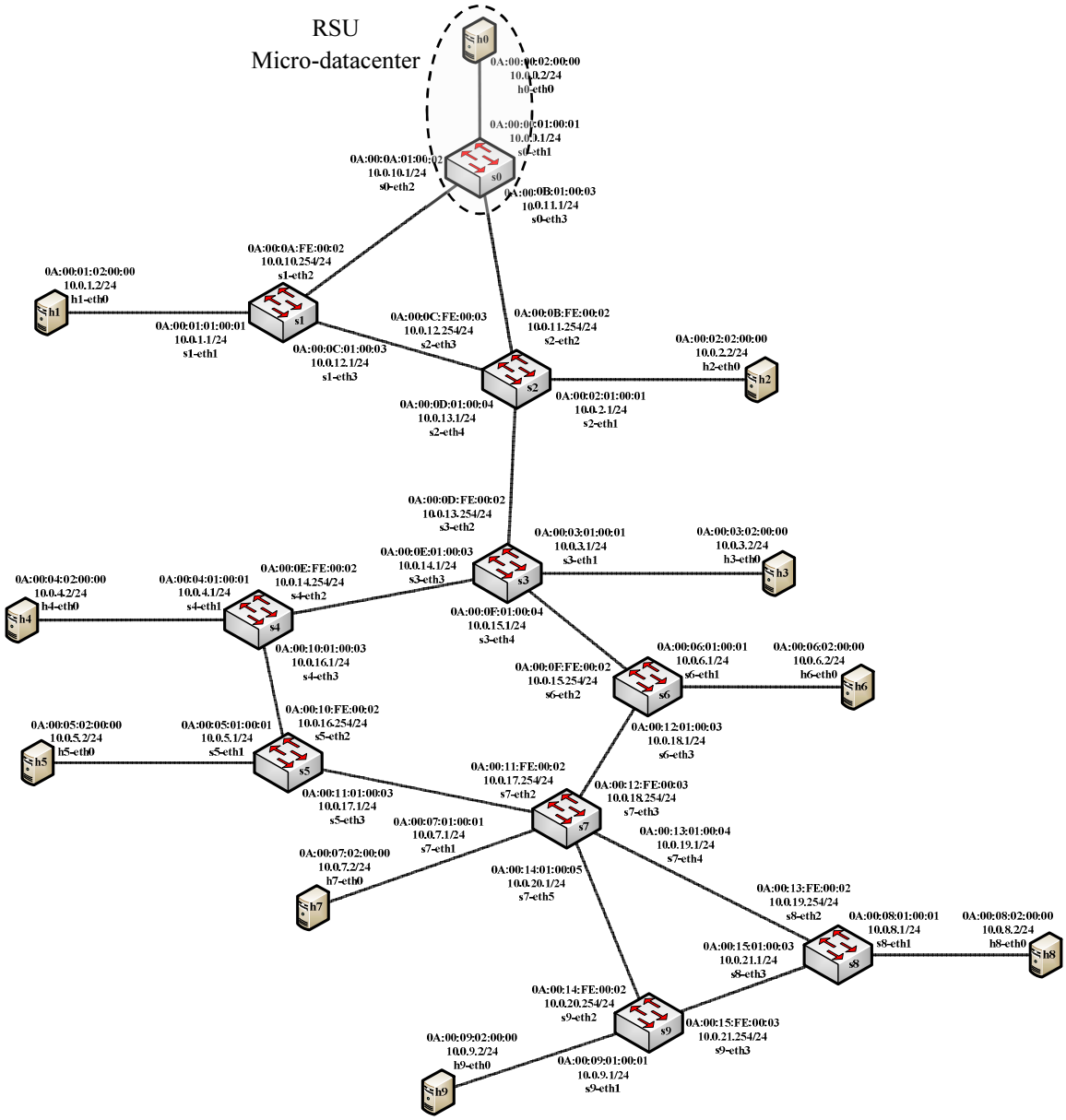
[68] N. Ferreira and J. Fonseca, "On the End-to-End Delay Analysis for an IEEE 802.11P/WAVE Protocol," in *IEEE Symposium on Communications and Vehicular Technology*, Ghent, 2011.

APPENDICES

APPENDIX A

MININET TOPOLOGY

Our Mininet Topology with IP and MAC addresses for OpenFlow enabled SDN.



APPENDIX B

ENABLING STOCHASTIC SWITCHING IN OPEN VSWITCH

Changes made in `<openvswitch-install-dir>/ofproto/ofproto-dpif-xlate.c` to support stochastic switching in Open vSwitch 2.1.

Add headers:

```
#include <stdlib.h>
#include <time.h>
```

Add global variable:

```
static bool is_srand_initialized = false;
```

Modify functions:

```
static void
xlate_select_group(struct xlate_ctx *ctx, struct group_dpif *group)
{
    struct flow_wildcards *wc = &ctx->xout->wc;
    const struct ofputil_bucket *bucket;
    uint32_t basis;

    // The following tells the caching code that every packet in
    // the flow in question must go to the userspace "slow path".
    ctx->xout->slow |= SLOW_CONTROLLER;

    basis = hash_bytes(ctx->xin->flow.dl_dst, sizeof ctx->xin->flow.dl_dst, 0);
    bucket = group_best_live_bucket(ctx, group, basis);
    if (bucket) {
        memset(&wc->masks.dl_dst, 0xff, sizeof wc->masks.dl_dst);
        xlate_group_bucket(ctx, bucket);
    }
}

static const struct ofputil_bucket *
group_best_live_bucket(const struct xlate_ctx *ctx,
                   const struct group_dpif *group,
                   uint32_t basis) // basis in not being used
{
    uint32_t rand_num = 0, sum = 0;
    const struct ofputil_bucket *bucket = NULL;
    const struct list *buckets;
```

```
    // initialize random seed once
    if (!is_srand_initialized) {
        srand(time(NULL));
        is_srand_initialized = true;
    }

    // generate a random number in [1, 10]
    rand_num = (rand() % 10) + 1;

    group_dpif_get_buckets(group, &buckets);
    LIST_FOR_EACH (bucket, list_node, buckets) {
        if (bucket_is_alive(ctx, bucket, 0)) {
            sum += bucket->weight;
            if (rand_num <= sum) {
                return bucket; // return this bucket
            }
        }
    }

    return bucket; // return NULL
}
```

APPENDIX C

MININET PYTHON TOPOLOGY SCRIPT

Python script to create topology (Appendix A) in Mininet.

```
#!/usr/bin/python

"""CRM Topology with 10 switches and 10 hosts
"""

from mininet.cli import CLI
from mininet.topo import Topo
from mininet.net import Mininet
from mininet.link import TCLink
from mininet.log import setLogLevel

class CRMTopo( Topo ):

        def __init__( self ):
                "Create CRM Topology"

                # Initialize topology
                Topo.__init__( self )

                # Add hosts
                h0 = self.addHost( 'h0' )
                h1 = self.addHost( 'h1' )
                h2 = self.addHost( 'h2' )
                h3 = self.addHost( 'h3' )
                h4 = self.addHost( 'h4' )
                h5 = self.addHost( 'h5' )
                h6 = self.addHost( 'h6' )
                h7 = self.addHost( 'h7' )
                h8 = self.addHost( 'h8' )
                h9 = self.addHost( 'h9' )

                # Add switches
                s0 = self.addSwitch( 's0', listenPort=6634 )
                s1 = self.addSwitch( 's1', listenPort=6635 )
                s2 = self.addSwitch( 's2', listenPort=6636 )
                s3 = self.addSwitch( 's3', listenPort=6637 )
                s4 = self.addSwitch( 's4', listenPort=6638 )
                s5 = self.addSwitch( 's5', listenPort=6639 )
                s6 = self.addSwitch( 's6', listenPort=6640 )
                s7 = self.addSwitch( 's7', listenPort=6641 )
                s8 = self.addSwitch( 's8', listenPort=6642 )
                s9 = self.addSwitch( 's9', listenPort=6643 )

                # Add links between hosts and switches
                self.addLink( h0, s0, delay='0ms' ) # h0-eth0 <-> s0-eth1, delay = 0ms
```

```
                self.addLink( h1, s1, delay='0ms' ) # h1-eth0 <-> s1-eth1, delay = 0ms
                self.addLink( h2, s2, delay='0ms' ) # h2-eth0 <-> s2-eth1, delay = 0ms
                self.addLink( h3, s3, delay='0ms' ) # h3-eth0 <-> s3-eth1, delay = 0ms
                self.addLink( h4, s4, delay='0ms' ) # h4-eth0 <-> s4-eth1, delay = 0ms
                self.addLink( h5, s5, delay='0ms' ) # h5-eth0 <-> s5-eth1, delay = 0ms
                self.addLink( h6, s6, delay='0ms' ) # h6-eth0 <-> s6-eth1, delay = 0ms
                self.addLink( h7, s7, delay='0ms' ) # h7-eth0 <-> s7-eth1, delay = 0ms
                self.addLink( h8, s8, delay='0ms' ) # h8-eth0 <-> s8-eth1, delay = 0ms
                self.addLink( h9, s9, delay='0ms' ) # h9-eth0 <-> s9-eth1, delay = 0ms

                # Add links between switches, with bandwidth 100Mbps
                self.addLink( s0, s1, bw=100 ) # s0-eth2 <-> s1-eth2, BW = 100Mbps
                self.addLink( s0, s2, bw=100 ) # s0-eth3 <-> s2-eth2, BW = 100Mbps
                self.addLink( s1, s2, bw=100 ) # s1-eth3 <-> s2-eth3, BW = 100Mbps
                self.addLink( s2, s3, bw=100 ) # s2-eth4 <-> s3-eth2, BW = 100Mbps
                self.addLink( s3, s4, bw=100 ) # s3-eth3 <-> s4-eth2, BW = 100Mbps
                self.addLink( s3, s6, bw=100 ) # s3-eth4 <-> s6-eth2, BW = 100Mbps
                self.addLink( s4, s5, bw=100 ) # s4-eth3 <-> s5-eth2, BW = 100Mbps
                self.addLink( s5, s7, bw=100 ) # s5-eth3 <-> s7-eth2, BW = 100Mbps
                self.addLink( s6, s7, bw=100 ) # s6-eth3 <-> s7-eth3, BW = 100Mbps
                self.addLink( s7, s8, bw=100 ) # s7-eth4 <-> s8-eth2, BW = 100Mbps
                self.addLink( s7, s9, bw=100 ) # s7-eth5 <-> s9-eth2, BW = 100Mbps
                self.addLink( s8, s9, bw=100 ) # s8-eth3 <-> s9-eth3, BW = 100Mbps

def run():
        "Create and configure CRM network"
        topo = CRMTopo()
        net = Mininet( topo=topo, link=TCLink, controller=None )

        # Set interface IP and MAC addresses for hosts
        h0 = net.get( 'h0' )
        h0.intf( 'h0-eth0' ).setIP( '10.0.0.2', 24 )
        h0.intf( 'h0-eth0' ).setMAC( '0A:00:00:02:00:00' )

        h1 = net.get( 'h1' )
        h1.intf( 'h1-eth0' ).setIP( '10.0.1.2', 24 )
        h1.intf( 'h1-eth0' ).setMAC( '0A:00:01:02:00:00' )

        h2 = net.get( 'h2' )
        h2.intf( 'h2-eth0' ).setIP( '10.0.2.2', 24 )
        h2.intf( 'h2-eth0' ).setMAC( '0A:00:02:02:00:00' )

        h3 = net.get( 'h3' )
        h3.intf( 'h3-eth0' ).setIP( '10.0.3.2', 24 )
        h3.intf( 'h3-eth0' ).setMAC( '0A:00:03:02:00:00' )

        h4 = net.get( 'h4' )
        h4.intf( 'h4-eth0' ).setIP( '10.0.4.2', 24 )
```

```
h4.intf( 'h4-eth0' ).setMAC( '0A:00:04:02:00:00' )

h5 = net.get( 'h5' )
h5.intf( 'h5-eth0' ).setIP( '10.0.5.2', 24 )
h5.intf( 'h5-eth0' ).setMAC( '0A:00:05:02:00:00' )

h6 = net.get( 'h6' )
h6.intf( 'h6-eth0' ).setIP( '10.0.6.2', 24 )
h6.intf( 'h6-eth0' ).setMAC( '0A:00:06:02:00:00' )

h7 = net.get( 'h7' )
h7.intf( 'h7-eth0' ).setIP( '10.0.7.2', 24 )
h7.intf( 'h7-eth0' ).setMAC( '0A:00:07:02:00:00' )

h8 = net.get( 'h8' )
h8.intf( 'h8-eth0' ).setIP( '10.0.8.2', 24 )
h8.intf( 'h8-eth0' ).setMAC( '0A:00:08:02:00:00' )

h9 = net.get( 'h9' )
h9.intf( 'h9-eth0' ).setIP( '10.0.9.2', 24 )
h9.intf( 'h9-eth0' ).setMAC( '0A:00:09:02:00:00' )

# Set interface MAC address for switches (NOTE: IP
# addresses are not assigned to switch interfaces)
s0 = net.get( 's0' )
s0.intf( 's0-eth1' ).setMAC( '0A:00:00:01:00:01' )
s0.intf( 's0-eth2' ).setMAC( '0A:00:0A:01:00:02' )
s0.intf( 's0-eth3' ).setMAC( '0A:00:0B:01:00:03' )

s1 = net.get( 's1' )
s1.intf( 's1-eth1' ).setMAC( '0A:00:01:01:00:01' )
s1.intf( 's1-eth2' ).setMAC( '0A:00:0A:FE:00:02' )
s1.intf( 's1-eth3' ).setMAC( '0A:00:0C:01:00:03' )

s2 = net.get( 's2' )
s2.intf( 's2-eth1' ).setMAC( '0A:00:02:01:00:01' )
s2.intf( 's2-eth2' ).setMAC( '0A:00:0B:FE:00:02' )
s2.intf( 's2-eth3' ).setMAC( '0A:00:0D:01:00:03' )
s2.intf( 's2-eth4' ).setMAC( '0A:00:0C:FE:00:04' )

s3 = net.get( 's3' )
s3.intf( 's3-eth1' ).setMAC( '0A:00:03:01:00:01' )
s3.intf( 's3-eth2' ).setMAC( '0A:00:0D:FE:00:02' )
s3.intf( 's3-eth3' ).setMAC( '0A:00:0E:01:00:03' )
s3.intf( 's3-eth4' ).setMAC( '0A:00:0F:01:00:04' )

s4 = net.get( 's4' )
s4.intf( 's4-eth1' ).setMAC( '0A:00:04:01:00:01' )
```

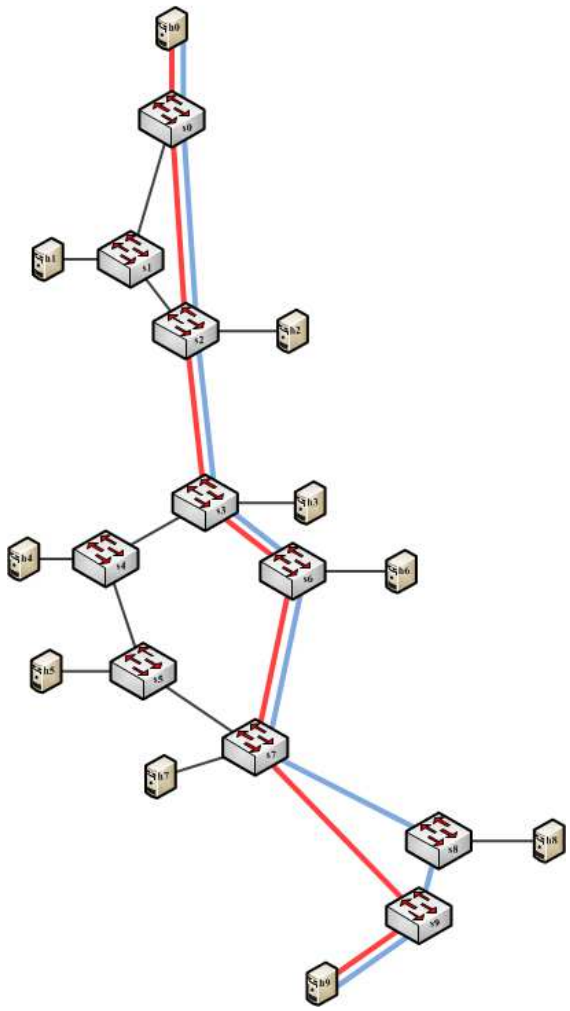
```
s4.intf( 's4-eth2' ).setMAC( '0A:00:0E:FE:00:02' )
s4.intf( 's4-eth3' ).setMAC( '0A:00:10:01:00:03' )

s5 = net.get( 's5' )
s5.intf( 's5-eth1' ).setMAC( '0A:00:05:01:00:01' )
s5.intf( 's5-eth2' ).setMAC( '0A:00:10:FE:00:02' )
s5.intf( 's5-eth3' ).setMAC( '0A:00:11:01:00:03' )

s6 = net.get( 's6' )
s6.intf( 's6-eth1' ).setMAC( '0A:00:06:01:00:01' )
s6.intf( 's6-eth2' ).setMAC( '0A:00:0F:FE:00:02' )
s6.intf( 's6-eth3' ).setMAC( '0A:00:12:01:00:03' )

s7 = net.get( 's7' )
s7.intf( 's7-eth1' ).setMAC( '0A:00:07:01:00:01' )
s7.intf( 's7-eth2' ).setMAC( '0A:00:11:FE:00:02' )
s7.intf( 's7-eth3' ).setMAC( '0A:00:12:FE:00:03' )
s7.intf( 's7-eth4' ).setMAC( '0A:00:13:01:00:04' )
s7.intf( 's7-eth5' ).setMAC( '0A:00:14:01:00:05' )

s8 = net.get( 's8' )
s8.intf( 's8-eth1' ).setMAC( '0A:00:08:01:00:01' )
s8.intf( 's8-eth2' ).setMAC( '0A:00:13:FE:00:02' )
s8.intf( 's8-eth3' ).setMAC( '0A:00:15:01:00:03' )

s9 = net.get( 's9' )
s9.intf( 's9-eth1' ).setMAC( '0A:00:09:01:00:01' )
s9.intf( 's9-eth2' ).setMAC( '0A:00:14:FE:00:02' )
s9.intf( 's9-eth3' ).setMAC( '0A:00:15:FE:00:03' )

net.start()

# Add routing table entries for hosts (NOTE: The gateway
# IPs 10.0.X.1 are not assigned to switch interfaces)
h0.cmd( 'route add default gw 10.0.0.1 dev h0-eth0' )
h1.cmd( 'route add default gw 10.0.1.1 dev h1-eth0' )
h2.cmd( 'route add default gw 10.0.2.1 dev h2-eth0' )
h3.cmd( 'route add default gw 10.0.3.1 dev h3-eth0' )
h4.cmd( 'route add default gw 10.0.4.1 dev h4-eth0' )
h5.cmd( 'route add default gw 10.0.5.1 dev h5-eth0' )
h6.cmd( 'route add default gw 10.0.6.1 dev h6-eth0' )
h7.cmd( 'route add default gw 10.0.7.1 dev h7-eth0' )
h8.cmd( 'route add default gw 10.0.8.1 dev h8-eth0' )
h9.cmd( 'route add default gw 10.0.9.1 dev h9-eth0' )

# Add arp cache entries for hosts
h0.cmd( 'arp -s 10.0.0.1 0A:00:00:01:00:01 -i h0-eth0' )
h1.cmd( 'arp -s 10.0.1.1 0A:00:01:01:00:01 -i h1-eth0' )
```

```
        h2.cmd( 'arp -s 10.0.2.1 0A:00:02:01:00:01 -i h2-eth0' )
        h3.cmd( 'arp -s 10.0.3.1 0A:00:03:01:00:01 -i h3-eth0' )
        h4.cmd( 'arp -s 10.0.4.1 0A:00:04:01:00:01 -i h4-eth0' )
        h5.cmd( 'arp -s 10.0.5.1 0A:00:05:01:00:01 -i h5-eth0' )
        h6.cmd( 'arp -s 10.0.6.1 0A:00:06:01:00:01 -i h6-eth0' )
        h7.cmd( 'arp -s 10.0.7.1 0A:00:07:01:00:01 -i h7-eth0' )
        h8.cmd( 'arp -s 10.0.8.1 0A:00:08:01:00:01 -i h8-eth0' )
        h9.cmd( 'arp -s 10.0.9.1 0A:00:09:01:00:01 -i h9-eth0' )

        # Open Mininet Command Line Interface
        CLI(net)

        # Teardown and cleanup
        net.stop()

if __name__ == '__main__':
    setLogLevel('info')
    run()
```

APPENDIX D

PERFORMING STOCHASTIC SWITCHING IN MININET

Multipath stochastic switching to route packets from host h0 to host h9 along the two highlighted paths shown below, that is, via switches [s0, s2, s3, s6, s7, s9] and [s0, s2, s3, s6, s7, s8, s9].



Add group table entry in switch s7 and flow table entries in all switches on the paths to setup the flows using the following commands.

```
# ADD-GROUP at s7 for [0, 9]

ovs-ofctl -O OpenFlow13 add-group tcp:127.0.0.1:6641
group_id=0,type=select,bucket=weight:7,mod_dl_src:0A:00:14:01:00:05,mod_dl_dst:0A:00:14
:FE:00:02,output=5,bucket=weight:3,mod_dl_src:0A:00:13:01:00:04,mod_dl_dst:0A:00:13:FE:
00:02,output=4

# ADD-FLOW(s) for [0, 9] at [0, 2, 3, 6, 7, 8, 9]
```

```
ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6634
in_port=1,ip,nw_src=10.0.0.2,nw_dst=10.0.9.2,actions=mod_dl_src:0A:00:0B:01:00:03,mod_d
l_dst:0A:00:0B:FE:00:02,output=3

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6636
in_port=2,ip,nw_src=10.0.0.2,nw_dst=10.0.9.2,actions=mod_dl_src:0A:00:0D:01:00:04,mod_d
l_dst:0A:00:0D:FE:00:02,output=4

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6637
in_port=2,ip,nw_src=10.0.0.2,nw_dst=10.0.9.2,actions=mod_dl_src:0A:00:0F:01:00:04,mod_d
l_dst:0A:00:0F:FE:00:02,output=4

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6640
in_port=2,ip,nw_src=10.0.0.2,nw_dst=10.0.9.2,actions=mod_dl_src:0A:00:12:01:00:03,mod_d
l_dst:0A:00:12:FE:00:03,output=3

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6641
in_port=3,ip,nw_src=10.0.0.2,nw_dst=10.0.9.2,actions=group=0

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6643
in_port=2,ip,nw_src=10.0.0.2,nw_dst=10.0.9.2,actions=mod_dl_src:0A:00:09:01:00:01,mod_d
l_dst:0A:00:09:02:00:00,output=1

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6642
in_port=2,ip,nw_src=10.0.0.2,nw_dst=10.0.9.2,actions=mod_dl_src:0A:00:15:01:00:03,mod_d
l_dst:0A:00:15:FE:00:03,output=3

ovs-ofctl -O OpenFlow13 add-flow tcp:127.0.0.1:6643
in_port=3,ip,nw_src=10.0.0.2,nw_dst=10.0.9.2,actions=mod_dl_src:0A:00:09:01:00:01,mod_d
l_dst:0A:00:09:02:00:00,output=1
```