8-21-2020 10:00 AM

Network Resource and Performance Optimization in Autonomous Systems: A Connected Vehicles and Autonomous Networks Perspective

Ibrahim Shaer, *The University of Western Ontario*

Supervisor: Shami, Abdallah, *The University of Western Ontario*
Co-Supervisor: Haque, Anwar, *The University of Western Ontario*
A thesis submitted in partial fulfillment of the requirements for the Master of Engineering Science degree in Electrical and Computer Engineering

Abstract

This thesis covers two topics that optimize a network-related problem subject to environment-specific constraints; placing vehicular applications and executing network traffic assignment changes. The first topic introduces an optimization model, Resource and Delay-aware V2X service Placement (RDP), and a baseline approach that only considers the resource requirements of vehicular services. Both are responsible for placing vehicular services used by vehicular applications in an edge computing environment. Under different simulation scenarios, the results obtained by RDP satisfy the delay requirements of vehicular applications as opposed to the baseline approach. The second topic examines the efficient execution of inter-domain traffic changes under bandwidth, monetary, and infrastructural constraints. An oracle algorithm and two heuristics are formulated, and evaluation criteria are devised to reflect the constraints. These algorithms are evaluated on different networks, and the results reported show that OrderSteps (OSS) heuristic satisfies the constraints and outperforms the oracle implementation in terms of run-time.

Keywords: Vehicle-to-everything applications, V2X applications, Optimization, Edge Computing, Inter-domain Traffic Engineering, TE, Partitioning, Execution Plans, Traffic Assignment Changes.

Summary for Lay Audience

This thesis examines two topics that are related to autonomous vehicles and Internet data traffic routing. Autonomous vehicles are envisioned to transfer the authority in the decision-making process from the human driving it to the computer operating the vehicle. To achieve this vision, the vehicle must be equipped with sensors that are gathering information from their surrounding environment and applications, resembling the mobile applications, responsible for extracting useful insights that can be valuable in risky and dangerous situations. Such situations require fast processing and response in the space of 10ms from these applications. Unfortunately, all these applications can not be installed on a single vehicle due to its limited capacity. Therefore, the first topic investigates this issue by first breaking down these applications into smaller independent components for capacity purposes. After that, these components are installed on infrastructure of varying capacity close to the vehicles, so that the components' requirements in terms of capacity and the vehicles' requirements in terms of response time are satisfied. Currently, people are extensively using their mobile phones and their smart TVs to surf the Internet and to watch movies. Any activity related to the Internet requires, on the network level, to route the data from source to destination as a response for a specific request. To better manage the routing process, network operators have divided the network into a set of routers whereby each set is defined as an Autonomous System (AS). Each of these Autonomous Systems (ASes) forms relations with other ASes to route the traffic exiting them to its destination. The main objective of each AS is to maximize the profit and optimize its resource usage which is achieved by mandating the routing of traffic in specific ways in the

infrastructure they manage. Each network operator is concerned in routing traffic generating mass data such as YouTube and Netflix traffic. Due to the traffic's dynamicity, the network operators' respective routing strategies change frequently, so that they keep adhering to their objective. The second topic covers transitioning between two routing strategies in a way that satisfies the network operators objectives.

Statement of Co-Authorship

This thesis includes the following manuscript that has been accepted for publication.

- I. Shaer, A. Haque, A. Shami, "Multi-Component V2X Applications Placement in Edge Computing Environment," in *International Conference on Communications (ICC)*, 2020 (accepted)

This thesis contains the following manuscript which will be submitted for review.

- I. Shaer, G. Sidebottom, A. Haque, A. Shami, "Efficient Execution of Egress Traffic Engineering Changes" (pending submission)

The following coauthors have provided their expertise to complete the abovementioned articles:

- A. Shami contributed to the work done in Chapter 3 and 4 through his theoretical and technical expertise as a professor and his opinion and perspective.
- A. Haque contributed to the work done in Chapter 3 and 4 through his theoretical and technical expertise as a professor and his opinion and perspective.
- G. Sidebottom contributed to the work done in Chapter 4 through his expertise and knowledge as a Principal Engineer at Juniper Networks.

Epigraph

“He who has a why to live for can bear almost any how.” Friedrich Nietzsche

Dedication

I would like to dedicate this thesis to my family who always wanted me to succeed and advance in my life.

Acknowledgements

Putting together all of this research work and smoothly transitioning into a new country would not have been possible without the help of many people. This excerpt is dedicated to express my gratitude to the people that were important factors in the realization of my Master's experience.

To my supervisors, Dr. Abdallah Shami and Anwar Haque. Thank you for your continuous support that helped me to grow as a researcher and most importantly as a critical thinker. Under your mentorship, I saw myself gain courage to tap into the uncertainties of research and come out with decisive victories. Throughout my research journey, I always found them willing to help and to listen to my concerns and guide me on any questions that I have. Again, thank you for all of your efforts that culminated in what I am today. I am looking forward to my future collaboration with you.

To Greg Sidebottom, my research industry partner, thank you for helping me throughout my research journey. You helped me realize how my work can be applied to industry problems. You were a true mentor for embedding "The Art of Programming".

To my family, Ahmad, Svetlana and Ikram, thank you for your encouragement and faith. I would not be more certain about my future steps academically without your support and belief. How can I forget you nodding in approval and saying that I got it while I am talking about my "techy" gibberish.

Along my research journey, I got to befriend some great people that enriched my experience throughout and made the adaptation process a lot more graceful. Thank you Abdallah, Tareq,

Yiota, Aziz, and Ayman. You were always there for me to share with me my ups and my downs. I am lucky to have gained such fulfilling friendships.

When I joined the lab, I was fortunate to come across a great person that later became a very close friend. That person is Dimitrios Manias. Dimitri was always there when I needed cheering or when I needed to rant. Either way, the conversation ended with laughter. Thank you Dimitri for being there not only as a friend but more as a brother.

Despite the famous proverb that says “out of sight, out of mind”, the Mohammads (Ibrahim, Bzeih, and Saleh), Mustafa, Soukaina, and Marya were sure to clearly defeat that saying. Thank you for always staying in contact despite the time differences and the calamities of recent events. Your support and encouragement will be forever cherished.

While the current pandemic has confined us at home, it was a great opportunity for me to get to know my roommates better. Jacob, Owen, Nicole and Jenna, thank you for being there to ease the isolation experience.

Lastly, I would like to thank my lab fellow members for offering help and support when needed which were both rewarding personally and professionally.

Table of Contents

List of Tables

List of Figures

Nomenclature

ASes Autonomous Systems

CA Cooperative Awareness Basic Service

DEN Decentralized Environmental Notification Service

E2E end-to-end

IP Internet Protocol

ITS Intelligent Transportation System

LDM Local Dynamic Map

OSS Order StepS Heuristic

QoS quality of service

RAM Random Access Memory

RDP Resource and Delay-aware V2X basic service Placement

RSU Roadside Unit

TE Traffic Engineering

V2X Vehicle-to-everything

Chapter 1

Introduction

The optimal placement of services and the management of network resources are important factors for the realization of autonomous systems such as connected vehicles and autonomous networks. In both cases, to achieve the desired performance of the envisioned systems, a clear objective and environment-specific constraints need to be considered. This thesis presents two articles representing a framework that encompasses the management and placement of network resources of specific applications while maintaining the application specific constraints. In this perspective, this thesis examines two important applications in the autonomous systems domain. The first topic covers the placement of vehicular services used by vehicular applications also referred to as vehicle-to-everything (V2X) applications in an edge computing environment where both delay and resource requirements are considered. The second topic addresses the automation of inter-domain network traffic assignment changes while considering peer links' bandwidth constraints, monetary costs, and infrastructural constraints. Each article illuminates new concepts, presents new algorithms, and advances the state-of-the-art.

1.1 Research Contributions

The following chapters introduce several research contributions:

- Chapter 3:

1. Decompose V2X applications into multi-V2X basic services;
2. Formulate the optimal V2X applications placement by considering their delay requirements and the resource requirements of their constituent components;
3. Evaluate the performance of the optimal placement in terms of average and density distribution of the delay for each V2X application under different traffic conditions.

- Chapter 4:

1. Devise evaluation criteria that verify the compliance of execution plans with network performance objectives;
2. Propose two new heuristics, one based on graph representation of traffic assignment changes and the second on sorting the traffic changes according to the cost associated upon their execution;
3. Formulate an exhaustive search algorithm, used as a benchmark to compare heuristics, that partitions the set of traffic assignment changes;
4. Evaluate and compare the proposed approaches on networks of different sizes and configurations.

Chapter 2

Background

This section presents background concepts that are needed for the understanding of the contributions in the following sections.

2.1 Edge Computing

To transition from the cloud computing to the edge computing paradigm, the research community have achieved several intermediate conceptual advancements that were necessary building blocks for the development of the edge computing scheme. The main intuition behind all of these concepts is to provide computational access close to mobile agents, which include user equipment (UE) and vehicles, in order to avoid relying on the centralized computing environment [1].

The first advancement in this transition culminated in the concept of ad-hoc clouds. Mobile agents that are spatially close combine and virtualize their computational power, so that the other agents that need these resources can use them to their benefit. However, many challenges need to be addressed for the utilization of ad-hoc clouds that are mainly related to privacy and security issues, reliability and incentives systems for agents to collaborate [1]. Ad-hoc cloud serves better as an amendment to a present architecture that provides cloud services rather than being the main

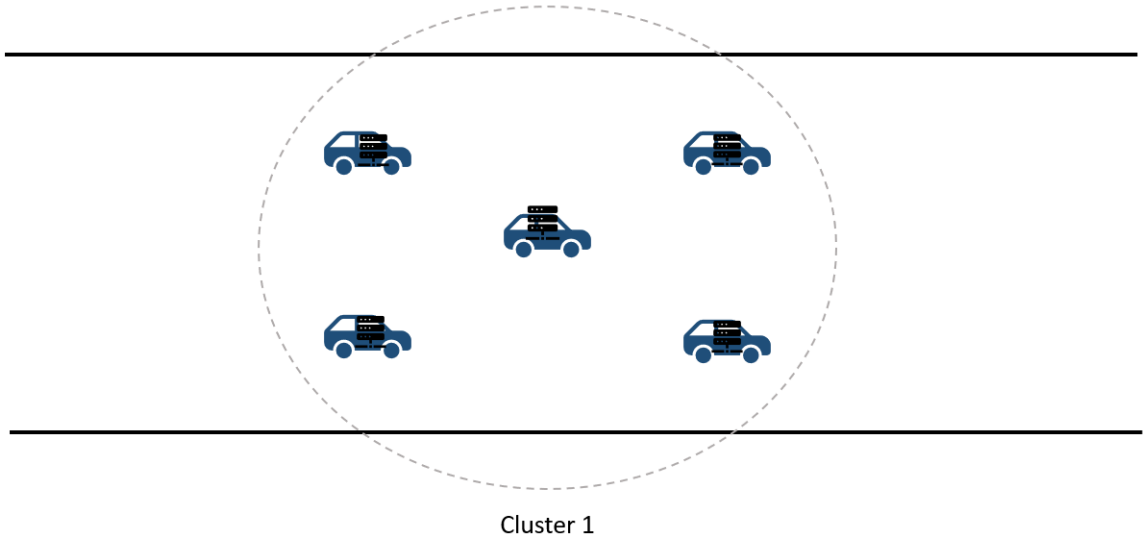


Figure 2.1: Ad-hoc Cloud

architecture that serves this purpose. Fig. 2.1 illustrates a typical example of vehicular ad-hoc network where vehicles form clusters dictated by vehicles' communication ranges. In each cluster, vehicles share their computational resources through virtualization allowing other vehicles to opportunistically use these resources.

While the Mobile Cloud Computing (MCC) deployment paradigm suffers from high latencies and exerts extreme traffic conditions on the network backhaul, the concept of cloudlets was revolutionary for mitigating the limitations of MCC. The main idea of cloudlets is deploying servers with high computational power and storage, in predefined locations, in close proximity to mobile users. The concept mirrors WiFi hotspots in providing cloud services to mobile users. However, this paradigm suffers from two significant shortcomings. (1) Connecting to a cloud service requires switching from the mobile network to the WiFi services provided by the cloudlets. This incurs delays due to the handover process which degrades the Quality of Service (QoS) of the applications in need of cloud services. (2) The cloudlets do not necessarily provide ubiquitous coverage for mobile users as they are placed following the network operators' objectives which limits the cloudlets' support for mobility. [1]

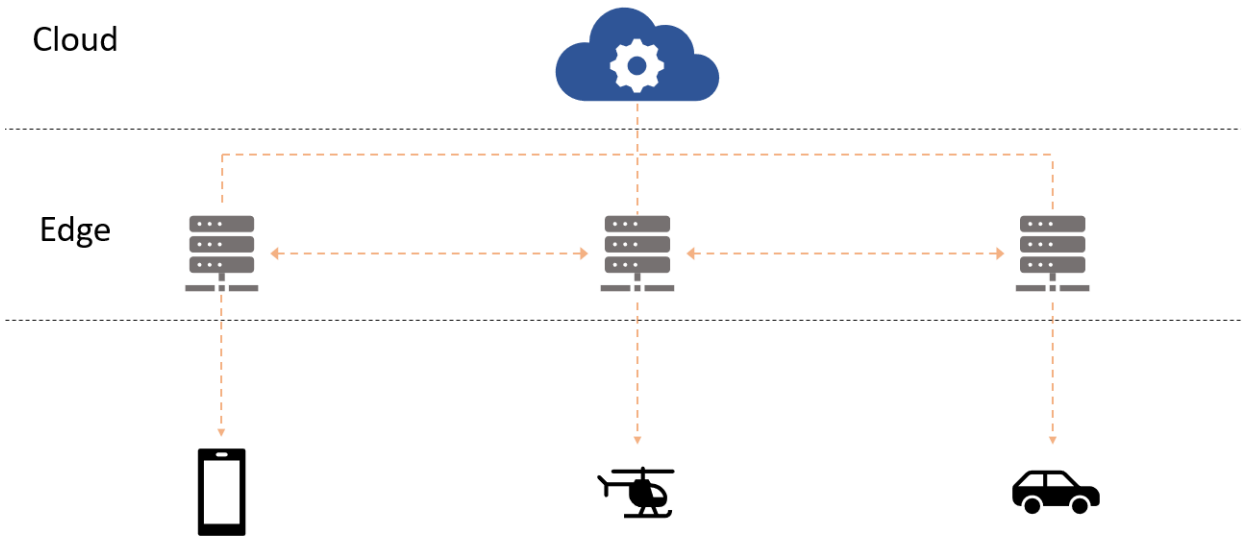


Figure 2.2: Edge Computing

The last advancement which is considered analogous to edge computing is fog computing. The concept of fog computing was introduced by Cisco [2] whereby computational resources are deployed at the edge of the network to provide low latency requirements for mobile and vehicular applications, contextual awareness for these applications, and wide-spread geographical distribution that can be leveraged to collect pervasive spatial-temporal data [1]. Fig. 2.2 demonstrates the de-facto architecture of an application deployment environment that integrates the edge computing principle. As it is depicted, different mobile agents such as mobile phones, Unmanned Aerial Vehicles (UAVs), and cars communicate with the servers deployed at the edge of the network. The edge servers share information among each other and communicate and transmit information to the centralized cloud.

The edge computing principle has spawned many research efforts that resulted in defining concepts such Mobile Edge Computing (MEC) [3], Small Cell Cloud (SCC) [4], Mobile Micro Clouds (MMC) [5], and Follow Me Cloud (FMC) [6]. The main difference between these concepts lies in the management and control of virtualized resources, deployment strategies, and the densification levels of computing resources.

2.2 Autonomous Vehicles

Any autonomous vehicle system relies on three main components [7]: RoadSide Unit (RSU), On-Board-Unit (OBU), and Application Unit (AU). The OBU is responsible for collecting and processing and sending data from sensors and surrounding environment. The RSU is the entity that hosts applications and provides Internet connectivity to the vehicles. The AU uses the connection capability provided by the OBU to use the applications hosted on the RSU. In detail, the main functionalities of these entities are as follows:

- On Board Unit (OBU): The OBU provides wireless radio access capabilities for the vehicle, and it ensures its connectivity to other vehicles and RSUs. Also, it is responsible for applying network functions such as congestion control and reliable data transfer. Additionally, it can act as a relay to forward the data from other OBUs destined to a different vehicular infrastructure. Lastly, it represents an intermediate infrastructure between the AU and the surrounding environment by providing connectivity to the AU.

- Application Unit (AU): The separation between the OBU and AU is logical. The AU is dedicated to use the applications of RSU, or it can host safety and non-safety vehicular applications. The AU uses the OBU to ensure its connectivity to the outside world.

- RSU: RSUs are usually deployed along the road side which are responsible for extending the vehicular agents' communication range, gathering and distributing the circulated information in the environment, hosting safety applications that are accessible to vehicles using communication technologies and providing Internet connectivity to OBUs. [7]

2.3 Traffic Engineering

Internet traffic engineering (TE) is a branch of network engineering that focuses on the performance optimization of IP networks by applying engineering principles that controls and manages

Internet traffic. The performance is optimized at both resource and traffic levels by considering the requirements of the network traffic representing myriad of applications and the underlying network infrastructure. On a bigger scale, the main objective of TE is to consolidate the reliability, scalability, and the survivability of the network. To that end, automated TE is performed to optimize Internet routing function by effectively steering the network traffic. The routing policies are executed within a framework that prioritizes the perceived performance by end users as it is the most significant indicator of the network performance while considering the utilization of the underlying resources. Careful methods should be applied for measuring network performance. When choosing network performance metrics, it is important that they represent the global network state. Optimizing inaccurate metrics can achieve local spatial and temporal objectives but can lead to devastating consequences on the quality of service of network applications and the state of the network in general [8].

Optimization of TE is a continual process that depends on the dynamicity of network traffic environment. This environment is always in the process of introducing new technologies, applications and user segments. Any or all of these factors impose new performance objectives that need to be integrated in the Internet TE procedure. Many aspects should be considered to create the pipeline for any Internet traffic engineering problem. This requires identifying the problem under study, solution space, desirable features of the solution, and the representative evaluation of the solution [8].

Before developing the TE principle, the networking research community adopted different routing algorithms and protocols responsible for routing traffic entering and leaving a specific domain or Autonomous Systems (ASes). For routing traffic entering a domain, each router between source and destination nodes selects the path with the least cost to forward the packets [8]. This is imposed by the Interior Gateway Protocol (ISPs) [9] that is designed to choose such a path for forwarding traffic. Such a methodology has a prominent disadvantage. Every router will prefer using the path with the least cost to forward traffic which will cause congestion on the path with least cost[10].

Regardless of the objectives that can vary depending on the network operator, the main intuition of TE is to avoid link congestion which has implications on the delays, jitter and performance predictability [8]. TE achieves this objective by routing traffic through under-utilized paths which contributes to network load balancing. To that end, TE creates tunnels that define a path from a source node to a destination node. To make a comprehensive decision, TE gathers information about the network topology, traffic distribution, and the resources available on the traversed links [10].

Two different methods of network traffic control can be leveraged in this respective: pro-active and reactive. Pro-active methods assume some unfavorable future events and create policies to prevent their occurrence. On the other hand, reactive methods are adopted for events that already took place in the network and measures are applied to roll back the network to its previous desirable state. Adopting any of the methods optimizes the utilization of network resources which significantly decreases capital expenditure and increases projected revenues by providing more resources to be leveraged for more network services [8].

Bibliography

[1] P. Mach and Z. Becvar, "Mobile Edge Computing: A Survey on Architecture and Computation Offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628-1656, thirdquarter 2017, doi: 10.1109/COMST.2017.2682318.

[2] M. K. Saroa and R. Aron, "Fog Computing and Its Role in Development of Smart Applications," in *2018 IEEE Intl Conf. on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Melbourne, Australia, 2018, pp. 1120-1127, doi: 10.1109/BDCloud.2018.00166.

[3] Y. Yu, "Mobile edge computing towards 5G: Vision, recent progress, and open challenges," *China Communications*, vol. 13, no. Supplement2, pp. 89-99, 2016, doi: 10.1109/CC.2016.7833463.

[4] "*FP7 European Project, Distributed Computing, Storage and Radio Resource Allocation Over Cooperative Femtocells (TROPIC)*,". [Online]. Available: https://cordis.europa.eu/project/id/318784

[5] S. Wang *et al.*, "Mobile Micro-Cloud: Application Classification, Mapping, and Deployment," in *Proc. Annu. Fall Meeting ITA (AMITA)*, New York, NY, USA, Oct. 2013, pp. 1-7.

[6] T. Taleb and A. Ksentini, "Follow me cloud: interworking federated clouds and distributed mobile networks," in *IEEE Network*, vol. 27, no. 5, pp. 12-19, September-October 2013, doi: 10.1109/MNET.2013.6616110.

[7] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti and H. Zedan, "A comprehensive survey on vehicular ad hoc network", *Journal of Network and Computer Application*, vol. 37, pp. 380-392, Jan. 2014.

[8] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. 2002. *Overview and Principles of Internet Traffic Engineering*. RFC 3272. RFC Editor

[9] S. Misra, S. Goswami, "Interior Gateway Protocols," in *Network Routing: Fundamentals, Applications, and Emerging Technologies* , Wiley, 2014, pp.131-157, doi: 10.1002/9781119114864.ch6.

[10] Ciena, *All About MPLS Traffic Engineering*, [Online]. Available: https://media.ciena.com/documents/All+About+MPLS +Traffic+Engineering+E+Book.pdf

Chapter 3

Multi-Component V2X Applications Placement in Edge Computing Environment

3.1 Introduction

Intelligent Transportation Systems (ITSs) are envisioned to ameliorate traffic congestion and improve road safety and traffic experience. ITSs have drawn the attention of a large number of stakeholders due to their direct effect on the manufacturing of sensor and wireless-equipped vehicles known as connected and autonomous cars. In this regard, Vehicle-to-everything (V2X) applications are considered a key enabler for the shift to ITSs in terms of traffic management. These applications allow the vehicles to communicate and exchange information with their surrounding environment that includes other vehicles, pedestrians and supporting road side units (RSUs).

To ensure road safety, these applications operate with stringent end-to-end (E2E) latency/delay. There are different paradigms that can determine the placement of V2X applications to address the E2E latency requirements. The placement of these services is disruptive to the customary cloud-based infrastructure. The projected increase in the number of connected and autonomous vehicles will result in data explosion. The data will be routed to a single centralized server creating severe

network traffic congestion [1]. Additionally, the centralized servers are usually located far from vehicles generating data; thus, incurring a huge E2E delay and high probability of packet loss. Furthermore, this architecture exposes a single point of failure, which is huge risk to take for time and mission-critical V2X applications.

Given these circumstances, distributing the cloud computing technology in proximity to vehicles is proposed as a viable solution to deal with the shortcomings of the centralized paradigm [2]. This computing architecture is referred to as Edge Computing. Edge Computing can support the latency requirements of the V2X applications which are critical for their performance [3]. In addition, the edge servers collect data from the close local nodes which allows for a more individualized experience for the V2X application users. While Edge Computing paradigm can ensure some V2X system-level performances, this come at the expense of limited computational power at the edge. This limitation hinders the processing of large amount of data. To address this challenge, Microservices architecture that decomposes a single application into decoupled modules can be used. Additionally, virtualization techniques can be adopted to make the full use of the available resources at the edge. Combining Microservices architecture and virtualization techniques has the potential to minimize the computational tasks executed at the edge; thus, making it a viable option for the placement of V2X applications. Hawilo *et. al* [4] investigated the applicability of this paradigm for Virtual Network Functions which display similar characteristics to V2X applications making it a viable option for their placement.

In the domain of V2X applications, 3rd Generation Partnership Project (3GPP) [5] envisions complex V2X applications that combine vehicle status analysis, imminent traffic events generation, and raw sensor data exchange that define the function of autonomous and connected vehicles. Each of these applications relies on the data processing and analysis of miniscule V2X basic services. Current research efforts in the domain of vehicular applications placement disregard the nature of vehicular applications and mainly focus on an abstract formulation of these applications which is detached from their defined structure mandated by standardization institutions.

This chapter starts with an overview of related work and the state-of-the-art in the field. After that, the system model and the problem formulation are presented. Next, the simulation procedure is outlined, and the results are investigated. The last section concludes the chapter and presents concluding remarks.

3.2 Related Work

A large body of work has been conducted to study the placement of vehicular applications, integration of edge computing paradigm into the vehicular environment, and computation offloading in vehicular networks. This section will discuss the related work in these fields and pinpoint their shortcomings motivating this research study.

Emera *et. al* [6] investigate the advantages of integrating Multi-Access Edge Computing environment compared to centralized cloud paradigm on the end-to-end (E2E) latency of Vulnerable Road Users (VRU) messages dissemination and processing. In their experiments, the application responsible for the processing of VRU messages was deployed on edge and cloud server. Both approaches were evaluated with varying the VRU density and the number of vehicles receiving the transmitted messages. In the same context, reference [7] addresses the placement of V2X services in a hybrid environment that includes edge and cloud services. To that end, the authors devise three approaches that consider the delay and resource requirements of these services.

To address vehicle mobility, references [8, 9, 10] discuss the migration of vehicular applications under different environment-related constraints. Yu *et. al* [8] first propose a hierarchical architecture divided into three levels: vehicular cloud, roadside cloud and central cloud. Each of these infrastructures have varying degrees of communication and computation capabilities. The authors focus on the efficient resource management of the defined entities while satisfying the service requests of moving vehicles. Towards that goal, a game theoretical approach and a VM reservation scheme are presented where the vehicle's main objective is to reserve the resources it needs while

minimizing the resource leasing costs. Similarly, [9] addresses the migration of V2X applications while satisfying their QoS requirements. The authors integrate trajectory prediction models to find the optimal migration decision that is achieved through partial updating of the trajectory and V2X applications' priority. A VM migration problem is also addressed by [10] in a roadside cloudlet. The authors focus on when and where the VM should be migrated. The decision is dictated by the overall network cost which encompasses migration costs and network traffic cost.

In the field of computation offloading, [11] proposes a resource-aware parallel offloading scheme that considers the mobility and the computational capabilities of the hosting vehicles. Markov chains were adopted to model the time-varying aspect of the aforementioned factors. To parallelize the offloading procedure, the tasks of non-safety applications are partitioned and offloaded to hosting severs. The sum of task offloading delay, tasks decomposition, and handover cost are adopted as the performance metric of offloading performance. Zhang *et. al* [12] adopt a predictive offloading strategy whereby its main objective is to minimize the latency and the transmission cost of the offloading process. The authors discuss two offloading strategies. The first utilizes the RSUs for offloading tasks and following the mobility predictive model, the results are delivered to the RSU that is closest to vehicle's position upon the completion of task processing. The second leverages the V2V multi-hop communication to deliver the task to the RSU predicted to produce task processing results when the requesting vehicle is in its proximity.

Purely focusing on the resource availability, [13] addresses the offloading of resource-intensive vehicular application tasks by considering the resources of both the vehicles and MEC servers. Their approach utilizes the unused TV spectrum. The work by Gangadharan *et. al* [14] targets the efficient delivery of services considering the bandwidth utilization and cost. Additionally, partitioning of the requested data is leveraged to accommodate for the limited resources at the edge.

All of these research efforts suffer from some limitations. While most of aforementioned work consider the resource requirements of vehicular applications, only some take into account the re-

source and the delay requirements of vehicular applications. Also, none of the works proposes decomposition of vehicular applications into smaller vehicular services responsible for processing the offloaded tasks or vehicle requests. In the same context, the environments whereby partitioning of tasks is applied, the authors assume that such tasks are uniform, and all edge nodes can process them. This shows that the applications deployed on the edges are homogeneous. Lastly, none of the related work consider realistic traffic conditions while evaluating their respective approaches.

As opposed to the above-mentioned research efforts, this study is the first to address the placement of V2X applications' components considering their resources requirements and the delay requirements of V2X applications. No comparisons are conducted between this study and the state-of-the-art in terms of results and problem formulation because it is the first research effort that focuses on solving the aforementioned problem. Given these circumstances, a self-evaluation approach is followed. Two optimization models are devised; the first that considers both the delay and the resource requirements, while the second only addresses the resource requirements. The second approach mirrors the environment constraints considered in some of the related work to measure their performance in the environment set in this experiment. Both models are evaluated under realistic simulation traffic conditions.

3.3 System Model

In the reference model, a highway scenario is considered. Each of the vehicles moving on the highway is running a set of V2X applications that are collecting data from nearby RSUs to function autonomously. RSUs and the vehicles are communicating directly using Dedicated Short-range Communication [15] and no communication takes place between any vehicles. Each RSU is equipped with a server which are both considered as an edge computing node. Vehicles are receiving data from V2X basic services placed on each RSU. European Telecommunication Standardization Institute (ETSI) defines three V2X basic services that are the foundation of any envisioned

V2X applications. The V2X basic services are as follows:

(1) Cooperative Awareness Basic Service (CA) [16] is responsible for creating, analyzing and sending Cooperative Awareness Basic Messages (CAMs) which include information about the vehicle's status and attributes, (2) Decentralized Environmental Notification Service [17] (DEN) broadcasts Decentralized Notification Messages (DENM) whenever a road hazard or abnormal traffic condition takes place, and (3) Media Downloading [18] service is requested on demand by the passengers of the vehicle.

ETSI defines Local Dynamic Maps (LDMs) [19] that are responsible for storing the sent CAMs and DENMs. Because LDMs store spatial relevant information, an LDM is deployed on each edge server. LDMs are queried by V2X basic services in order to retrieve information. Finally, in addition to basic vehicular services that are related to road safety, there is a variety of innovative applications that are referred to as value-added services that are of lower priority [20]. These services include augmented reality, parking location and others that are part of the infotainment services provided by vehicular applications. Compared to road safety applications, these services display high levels of diversity and individuation. Therefore, they need to be migrated when the vehicle moves from one edge server coverage zone to another. For this purpose, each edge server reserves part of its resources to accommodate these migrating services. In this section, the system design and the optimal optimization technique for V2X basic service placement are presented.

3.3.1 System Design

In the reference model used for the placement of V2X basic services, HWY 416 IC-721A that passes through the city of Ottawa is considered. The edge computing servers are deployed uniformly along the highway as the deployment of RSU is out of the scope of this study. No communication interference zone exists between any two successive RSUs to avoid the possibility of encountering ping-pong handover cases which will be difficult to handle in an optimization

model. Also, the handover delay when a vehicle changes its association from one RSU to another is considered negligible. Additionally, the vehicles are assumed to be always connected to RSUs throughout their journey. The E2E latency of a service is the sum of the communication, processing, transmission and propagation delay. The propagation delay is dependent on the medium of communication which is out of the scope of this study, and therefore considered negligible. In this model, DSRC, the communication technology between the moving vehicles and RSUs, affects the communication delay. In the proposed model, the processing and transmission delay between the communicated edge servers is considered. Each edge computing server has the same computing and processing power that are expressed by the number of cores and RAM available. Finally, the vehicle density is considered to model real case scenarios.

3.3.2 Optimization Problem

In the optimization function, a set of edge servers and V2X services are considered. Let N denote the set of edge servers where $n \in N$. Let U denote the set of unique V2X basic services where $u \in U$. The availability of the computational resources on the edge is denoted by matrix Cap where Cap_{kn} denotes the kth computational resources available on edge server n. Matrix R represents the resources required by the V2X basic services where R_{ku} represents the kth computational resources required by V2X basic service u. A binary row vector $\overrightarrow{q}$ denotes the edge servers a vehicle can communicate with. Let C be the matrix that represents the processing and the transmission latency between edge servers where C_{ij} represents the latency between edge server i and edge server j. Matrix M represents the V2X services needed by V2X applications where $M_{au} = 1$ denotes that application a needs V2X basic service u. Let X denote the placement matrix where $X_{un} = 1$ means that V2X basic service u is placed on edge server n. The column X_u denotes the placement of the V2X services on edge server n. D_a^v and D_a^{th} denote respectively the delay experienced by a moving vehicle v served by application a and the maximum tolerable threshold of this delay. To represent

the vehicles' density, γ is used. d^v_{com} and d^v_{DL} denote respectively the communication and download latency between a vehicle v and a serving edge server. The optimization function used to minimize the delay of V2X application is as follows:

$$min \sum_{a \in A} D^v_a \tag{3.1}$$

where:

$$D^v_a = d^v_{com} + max(M_a \odot min(X \odot (\gamma C \times \overrightarrow{q})) + d^v_{DL} \tag{3.2}$$

subject to:

$$D^v_a \leq D^{th}_a, \forall a \in A \tag{3.3}$$

$$RX \leq Cap \tag{3.4}$$

$$\sum X_n = 1, \forall n \in N \tag{3.5}$$

In what follows, the explanation of the equations (3.1)-(3.5).

- Equation (3.1) describes the overall objective which is minimizing the summation of the delay of all V2X applications experienced by a vehicle requesting their services.
- Equation (3.2) presents the components contributing to the delay of a V2X application. The delay of a V2X application is the delay of the V2X services it relies on depending on the edge servers a vehicle can communicate with. Because the functioning of a V2X basic service is independent of other V2X basic services, the delay of a V2X application is defined as the maximum of the delay of its constituent V2X basic services. This value is added to the communication and download link delay.
- Next, the equations (3.3) – (3.5) describe the constraints. Equation (3.3) defines that the delay of an application should not exceed its maximum defined tolerable delay.

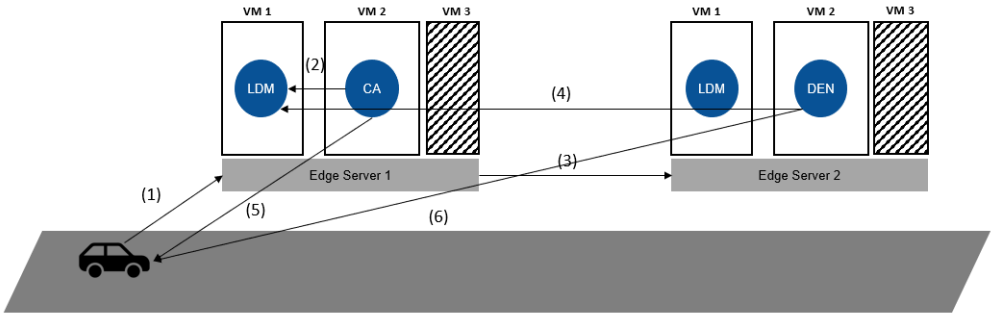


Figure 3.1: System Model

- Equation (3.4) ensures that the resources allocated to a V2X service do not exceed the available resource on the hosting edge server.

- Equation (3.5) limits placing only one V2X service on each edge server.

Fig. 3.1 illustrates an example of the communication and processing that takes place for a V2X application that requires CA and DEN services, given that each server has resources reserved for migrating applications denoted by VM 3.

The logic governing the realization of the application is as follows:

(1) The vehicle requests the services of an application. This step incurs communication delay that is denoted by d^{v}_{com}.

(2) The CA service found on Edge Server 1 requests the necessary information from the LDM. The processing of the request on the LDM is denoted by C_{11}.

(3) Edge server 1 communicates with the closest server that include DEN basic service. No delay is considered in this phase.

(4) DEN queries and receives information from the LDM that is closest to the requesting vehicle. The LDM on edge server 1 has accurate information about the requesting vehicle's surrounding environment. This delay is the sum of the processing delay of LDM on edge server 1 and the transmission delay between edge server 1 and 2. This is denoted by C_{12}.

(5, 6) These steps represent the CA and DEN response to the requesting vehicle. This delay is denoted by d^v_{DL}.

For basic service CA, the delay is as follows:

$$d^v_{CA} = C_{11} + d^v_{DL}$$

Similarly, the delay for DEN is:

$$d^v_{DEN} = C_{12} + d^v_{DL}$$

Given that the requests for each basic service are executed in parallel and that these services are independent in their execution, the delay experienced by a vehicle v requesting the services of application a is:

$$D^v_a = d^v_{com} + max\{d^v_{CA}, d^v_{DEN}\}$$

3.4 Experimental Setup and Results

3.4.1 Simulation Setup

In order to evaluate the placement of V2X basic services, a realistic simulation environment must be created. To this end, Simulation of Urban Mobility (SUMO) [21] was used to extract the movement of vehicles along a highway. A 4 km highway that resembles HWY 416 IC-712A was considered as a reference highway. Ontario traffic volume for provincial highways [22] provided the average daily traffic and the accident rates during summer, winter, weekdays and weekends. In the simulation setup, the statistics offered by this report were used to emulate moderate and heavy traffic experienced on HWY 416 IC-712A highway that is expressed through the vehicles

per hour parameter in SUMO. Regarding the movement of the vehicles, Table 3.1 summarizes the key parameters that are used in the simulation.

Table 3.1: SUMO vehicle movement parameters

Parameter	Value
Maximum Speed	$27.7 m/s$
Maximum Acceleration	$2.6 m/s^2$
Maximum Deceleration	$4.5 m/s^2$

The V2X applications considered are Platooning (PL), Sensor and Sensor State Mapping (SSM), Emergency Stop (ES), Pre-crash Sensing Warning (PSW) and Forward Collision Warning (FCW). Their corresponding performance requirements and service components are presented in Table 3.2 [23, 24]. Choosing these V2X applications stems from their importance and stringent performance requirements in the realm of the autonomous cars. In addition, in the context of the defined problem, each of the chosen V2X application offers a unique combination of V2X services.

Table 3.2: Breakdown of V2X applications' V2X basic service and performance metrics

Application	Service(s)	Latency(ms)	Reliability(%)
PL	CA	50	90
SSM	CA, DEN, Media	20	90
ES	DEN	10	95
PSW	CA, DEN	20	95
FCW	CA, DEN	10	95

In the simulation procedure, the communication delay between a vehicle and an RSU is 1 ms [25]. In this model, the processing delay is the amount of time required by a Local Dynamic Map to process the data requested by other V2X services either placed on the same or different edge server. In [26], the authors devise an LDM according to the specifications defined by ETSI. The application defines two Application Programming Interfaces (APIs) that retrieve information of the IDs of the vehicles driving on the same road and the vehicle driving immediately ahead of the requesting vehicle. For different number of queried vehicles ranging from 5 to 20 vehicles, the response time was between 3 and 5 ms with no clear correlation between the size of the data

and the response time. Consequently, in the simulation setup, the processing delay is generated uniformly between 3 and 5 ms. In the same context, the authors in [27], assumed the transmission latency between two edge servers to be between 1 and 5 ms.

Because the simulation procedure takes place under several vehicle densities, the increase of the data processing and transmission overhead with the increase of number of vehicles is inevitable. In this regard, the execution cost increases with the number of vehicles in proximity to the vehicle requesting the V2X application services. As the implementation of LDM did not consider cases beyond 20 vehicles, the added delay for these cases will be in the form of $\log(NC/20)$ where NC represents the number of cars and the expression is derived from the increase of processing delay upon the increase in size of the queried data in SQL [28].

Regarding edge servers, 10 edge servers are deployed every 400 m alongside the highway. Each of the RSUs hosts an LDM, V2X service and an optional migrating service. The computational requirements of CA, DEN and Media services are those of a small, medium and large VMs. Table 3.3 summarizes the edge server capabilities and the computational requirements of CA, DEN and Media services. In the experimental procedure, the placement of the V2X basic services is carried out using the defined optimization function. Next, vehicular traffic simulation is executed for defined densities that reflect moderate and heavy traffic. The traffic traces were generated for 1500 seconds. Every 10 seconds, a snapshot of the road condition is taken and delays for each V2X application for each vehicle is calculated. Finally, at the end of the simulation, the average delay for each V2X application is obtained.

Table 3.3: Computational Requirements

Entity	**Number of Cores**	**RAM**
Edge Server	8	8
CA	2	2
DEN	2	4
Media Service	4	6
LDM	4	2

3.4.2 Implementation

The optimization function was solved using IBM ILOG CPLEX 12.9.0 through its Python API. The solution is provided instantly for all simulation scenarios with different vehicle densities on a laptop with an Intel Core i7-8750 CPU, 2.21 GHz clock frequency and 16 GB of RAM. The final solution includes the V2X services placed on each edge server.

3.4.3 Results and Discussion

To evaluate the efficacy of the optimization function, the simulation procedure was carried out using two different traffic scenarios each representing moderate (Scenario 1) and heavy (Scenario 2) traffic models. The results are obtained as an average for five independent runs. To assess the placement function, the average delay of each V2X application under study is obtained, and it was compared to the maximum tolerable delay. Additionally, the model is evaluated using the probability density function the delay of each V2X application. The density function provides a more thorough overview about the distribution of the delays in terms of detecting extreme values that are overshadowed by the common values trend. Furthermore, the density functions reveal the shortcomings of the approaches that are concealed by the calculation of the mean.

The suggested optimization function is an integer linear program (ILP) that is considered part of the resource allocation problem proved to be NP-complete [29]. This optimization problem was infeasible, so a new heuristic algorithm that relaxes the delay threshold for each application by magnitudes of the reliability metric is formulated and successfully executed. This heuristic is referred to as: Resource and Delay-aware V2X basic services Placement (RDP) . The results of the simulation process in terms of the average delay and the probability densities of each V2X application are presented in Figures 3.2-3.5.

Figure 3.2 shows the mean delay for each of the V2X applications. The results clearly show that the average delay experienced by each V2X application is within the tolerable threshold. These

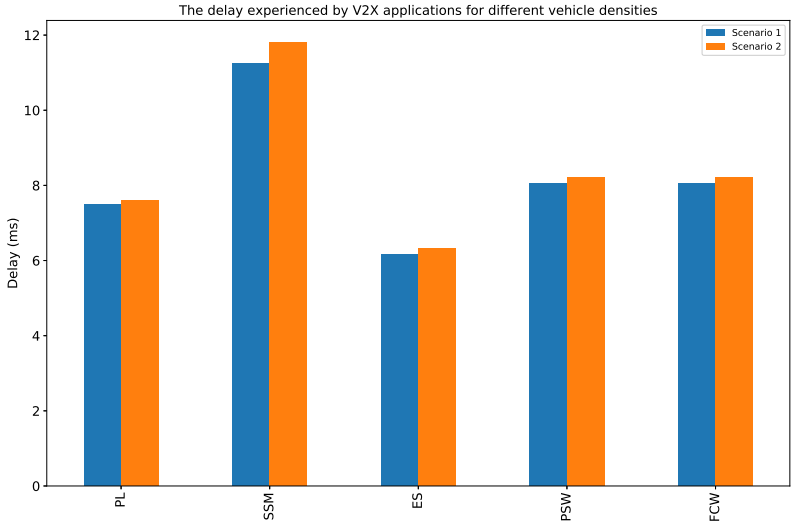


Figure 3.2: Average Delay of V2X Applications for different Vehicle Densities

results show that the heuristic algorithm met the stringent V2X application delay requirements. In terms of the vehicular traffic effect, the mean of delay of each application has slightly increased but still fulfills the overall objective of the placement function. The vehicles' density has contributed to an increase in the average delay of the V2X applications in the range of 1.3% to 4.8% whereby the SSM application has experienced the greatest variation. This fact shows that the placement of Media Service, that SSM relies on, is the most sensitive to traffic variation. On the other hand, the probability density functions tell a different story.

Figures 3.3 and 3.4 depicting the delay distribution for both cases show that the delay is highly skewed to the left which supports the viability of the approach. However, this is not the case for FCW application which shows that for each scenario, 20% and 25% of the experienced delay exceeds the tolerable threshold which is beyond the 5% permitted shown in Table 3.2. In terms of traffic effect, it is observed that there is a slight right shift of the probability distribution in scenario 2. Additionally, it is observed that some applications have similar probability distributions. This is attributed to the fact that these applications need the same V2X services, and as it shows, these services incur the most delay out of the other services that they rely on.

The dispersion of some of the probability density function is due to the limited number of edge

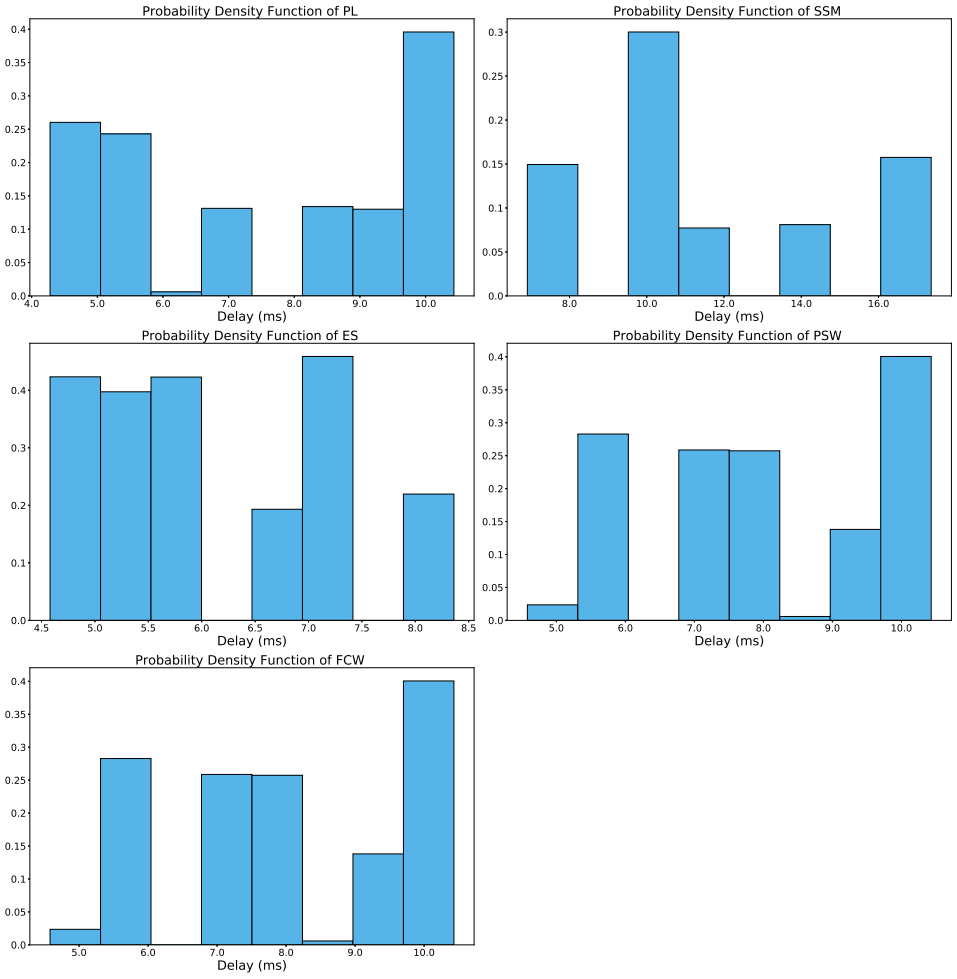


Figure 3.3: Probability Density Function for 1500 vehicles/hour

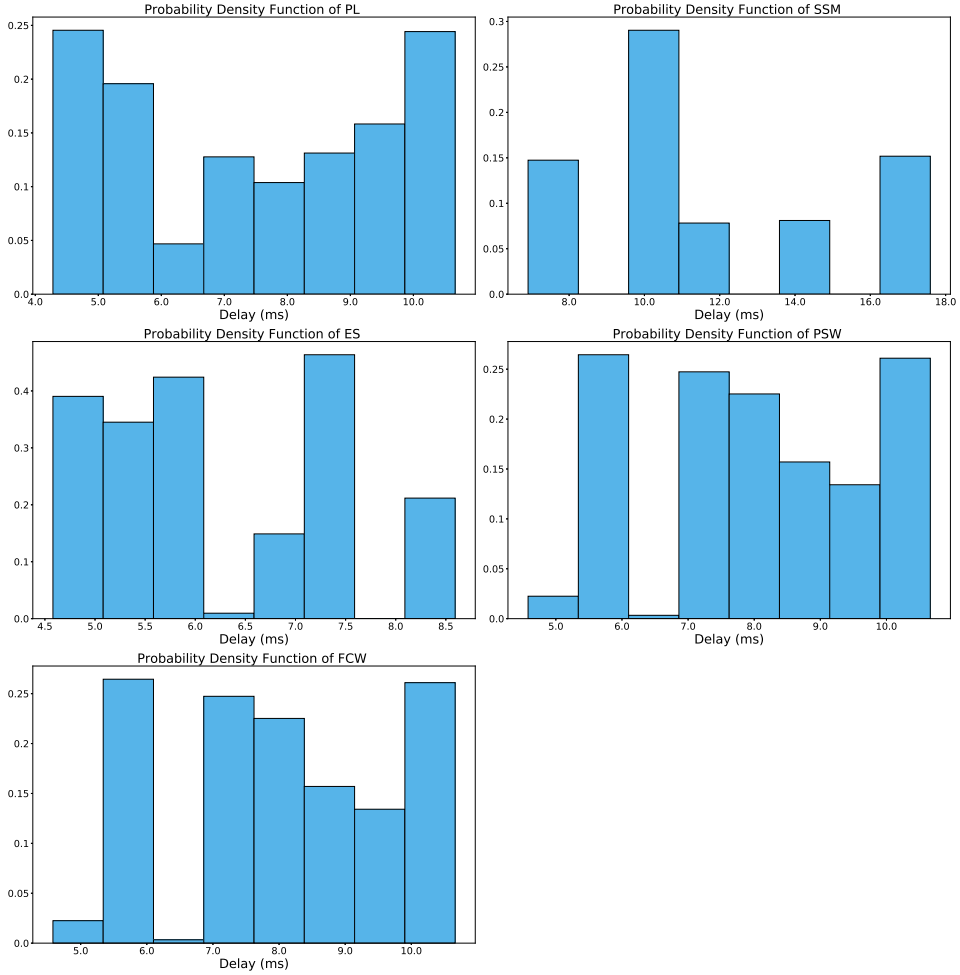


Figure 3.4: Probability Density Function for 1800 vehicles/hour

nodes hosting V2X services. The limited number of edge servers means that vehicles at the start and the end of the route will suffer from prolonged delay due to the distance separating the vehicles and the closest V2X basic services. In the cases of continued route, the suggested approach can be replicated along the highway to ensure that V2X services are delivered as expected.

For comparison purposes and to further cement this study's approach, a baseline approach that maximizes the resource utilization at each node server is compared to RDP. The baseline approach formulates a placement algorithm that takes into consideration only the available resources at each node. This baseline approach is reffered to as Resource-Aware Algorithm (RAA). The two approaches were evaluated according to the probability density functions of the delays of ES and

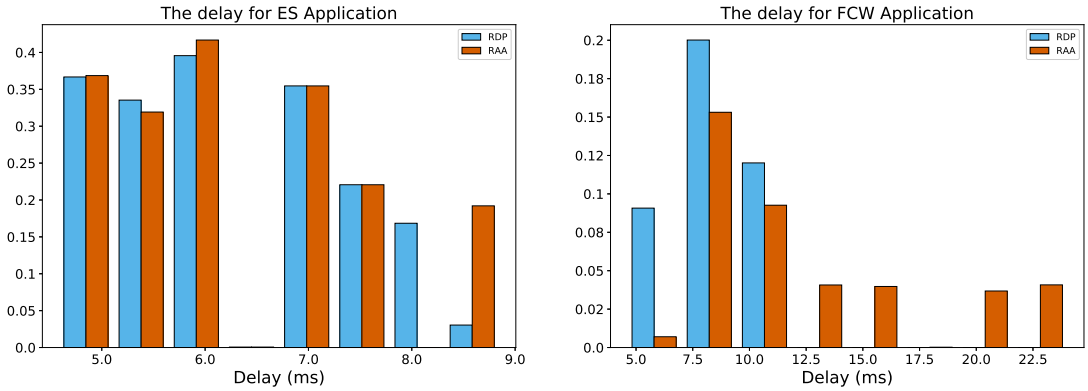


Figure 3.5: RDP vs RAA

FCW applications. These applications are chosen because of their most stringent delay requirements compared to the other applications. The probability densities are depicted in Figure 3.5.

The baseline line approach's density function shows promising results regarding the ES application as the full delay distribution is below the tolerable threshold. More values are concentrated on the extremes which makes it harder to gauge its value whenever the application is requested. However, for the case of FCW, this approach fails to be within the tolerable threshold rendering this approach ineffective for mission-critical applications. This is to be expected given that FCW application requires CA and DEN basic services. Due to the nature of RAA that overlooks the delay requirements, one of the five runs may have maximized the overall resource utilization for the edge servers; thus, deploying more of CA services results in decreasing the utilization which incurs extra delay for FCW application when requesting the services of CA.

3.5 Conclusion

This chapter addressed the efficient placement of V2X basic service comprising different V2X applications in an edge computing environment. To this end, an optimization function that minimizes the delay for multi-component V2X applications consisting of V2X services while considering the resource requirements of these services under different vehicular traffic conditions is formu-

lated. The approach was evaluated under realistic scenarios where homogeneous edge servers with limited computational power and variable traffic conditions were considered. Furthermore, the approach was compared to a baseline approach that only considers the resource requirements of V2X basic services. The results have shown that the approach guarantees an acceptable quality of service (QoS), and outperforms other approaches while emulating realistic conditions.

Bibliography

[1] J. Barrachina *et al.*, "Road Side Unit Deployment: A Density-Based Approach," *IEEE Intelligent Transportation Systems Magazine*, vol. 5, no. 3, pp. 30-39, Fall 2013, doi: 10.1109/MITS.2013.2253159.

[2] F. Faticanti, F. De Pellegrini, D. Siracusa, D. Santoro and S. Cretti, "Cutting Throughput with the Edge: App-Aware Placement in Fog Computing," in *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, Paris, France, 2019, pp. 196-203, doi: 10.1109/CSCloud/EdgeCom.2019.00026.

[3] D. Sabella *et al.*, "Toward fully connected vehicles: Edge computing for advanced automotive communications," 5GAA, December 2017. [Online]. Available: https://5gaa.org/wp-content/uploads/2017/12/5GAA T-170219- whitepaper-EdgeComputing 5GAA.pdf.

[4] H. Hawilo, M. Jammal and A. Shami, "Exploring Microservices as the Architecture of Choice for Network Function Virtualization Platforms," *IEEE Network*, vol. 33, no. 2, pp. 202-210, March/April 2019.

[5] *Study LTE Support For Vehicle To Everything (V2X) Services (Release 14)*, document 3GPP TR 22.885 V14.0.0, Dec. 2015.

[6] M. Emara, M. C. Filippou, and D. Sabella, "MEC-Assisted End-to-End Latency Evaluations

for C-V2X Communications," in *2018 European Conference on Networks and Communications (EuCNC)*, June 2018, pp. 1-9.

[7] A. Moubayed, A. Shami, P. Heidari, A. Larabi, and R. Brunner, "Edge-enabled V2X Service Placement for Intelligent Transportation Systems." *IEEE Transactions on Mobile Computing*, doi: 10.1109/TMC.2020.2965929.

[8] R. Yu, Y. Zhang, S. Gjessing, W. Xia and K. Yang, "Toward cloud-based vehicular networks with efficient resource management," in *IEEE Network*, vol. 27, no. 5, pp. 48-55, September-October 2013, doi: 10.1109/MNET.2013.6616115.

[9] X. Yu, M. Guan, M. Liao and X. Fan, "Pre-Migration of Vehicle to Network Services Based on Priority in Mobile Edge Computing," *IEEE Access*, vol. 7, pp. 3722-3730, 2019, doi: 10.1109/ACCESS.2018.2888478.

[10] H. Yao, C. Bai, D. Zeng, Q. Liang, and Y. Fan, "Migrate or not? exploring virtual machine migration in roadside cloudlet-based vehicular cloud," *Concurrency and Computation: Practice and Experience*, 27(18):5780-5792, 2015.

[11] J. Xie, Y. Jia, Z. Chen, Z. Nan and L. Liang, "Efficient task completion for parallel offloading in vehicular fog computing," *China Communications*, vol. 16, no. 11, pp. 42-55, Nov. 2019, doi: 10.23919/JCC.2019.11.004.

[12] K. Zhang, Y. Mao, S. Leng, Y. He and Y. ZHANG, "Mobile-Edge Computing for Vehicular Networks: A Promising Network Paradigm with Predictive Off-Loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36-44, June 2017, doi: 10.1109/MVT.2017.2668838.

[13] J. Du, F. R. Yu, X. Chu, J. Feng and G. Lu, "Computation Offloading and Resource Allocation

in Vehicular Networks Based on Dual-Side Cost Minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1079-1092, Feb. 2019, doi: 10.1109/TVT.2018.2883156.

[14] D. Gangadharan, O. Sokolsky, I. Lee, B. Kim, C. Lin and S. Shiraishi, "Bandwidth Optimal Data/Service Delivery for Connected Vehicles via Edges," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, 2018, pp. 106-113, doi: 10.1109/CLOUD.2018.00021.

[15] Z. Xu, X. Li, X. Zhao, M. H. Zhang, and Z. Wang, "DSRC versus 4G-LTE for Connected Vehicle Applications: A Study on Field Experiments of Vehicular Communication Performance," *Journal of Advanced Transportation*, vol. 2017, pp. 1-10, 2017.

[16] ETSI EN 302 637-2 (V1.3.2), "Intelligent Transport Systems (ITS) Vehicular Communications: Basic Set of Applications - Part 2: Specification of Cooperative Awareness Basic Service," 650 Route des Lucioles F- 06921 Sophia Antipolis Cedex - FRANCE, Technical Report, 2014.

[17] ETSI EN 302 637-3 (V1.2.1), "Intelligent Transport Systems (ITS) Vehicular Communications: Basic Set of Applications - Part 3: Specifications of Decentralized Environmental Notification Basic Service," 650 Route des Lucioles F- 06921 Sophia Antipolis Cedex - FRANCE, Technical Report, 2014.

[18] ETSI TR 102 638 (V1.1.1), "Intelligent Transport Systems (ITS) Vehicular Communications: Basic Set of Applications - Definitions," 650 Route des Lucioles F-06921 Sophia Antipolis Cedex - FRANCE, Tech. Rep., 2009.

[19] ETSI TR 102 863 (V1.1.1), "Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM); Rationale for and Guidance on Standardization," 650 Route des Lucioles F- 06921 Sophia Antipolis Cedex - FRANCE, Technical report, 2011.

[20] L. Li, Y. Li, and R. Hou, "A Novel Mobile Edge Computing-Based Architecture for Future Cellular Vehicular Networks," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, 2017.

[21] P. A. Lopez et al., "Microscopic Traffic Simulation using SUMO," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, Maui, HI, 2018, pp. 2575-2582, doi: 10.1109/ITSC.2018.8569938.

[22] Ontario Ministry of Transportation, "Provincial Highways Traffic Volumes 1988-2016," December 31, 2016. [Online]. Available: https://www.library.mto.gov.on.ca/SydneyPLUS/TechPubs/Theme.aspx?r=702797f=files

[23] 3GPP, "Study Enhancement 3GPP Support for 5G V2X Services (Release 15)," TR 22.886 V15.1.0, March 2017.

[24] 3GPP, "Study Enhancement 3GPP Support for 5G V2X Services (Release 16)," TR 22.886 V15.1.0, December 2018.

[25] D. Martin-Sacristan, S. Roger, D. Garcia-Roger, J. F. Monserrat, A. Kousaridas, P. Spapis, S. Ayaz, and C. Zhou, "Evaluation of lte- advanced connectivity options for the provisioning of v2x services," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Apr. 2018, pp. 1-6.

[26] H. Shimada, A. Yamaguchi, H. Takada, and K. Sato, "Implementation and Evaluation of Local Dynamic Map in Safety Driving Systems," *Journal of Transportation Technologies*, vol. 05, no.02, pp.102-112, 2015.

[27] S. Maheshwari, D. Raychaudhuri, I. Seskar, F. Bronzino, "Scalability and Performance Evaluation of Edge Cloud Systems for Latency Constrained Applications," in *Proceedings of the*

2018 IEEE/ACM Symposium on Edge Computing (SEC), Seattle, WA, USA, 25-27 October 2018, pp. 286-299.

[28] "MySQL 8.0 Reference Manual::13.8.2 EXPLAIN Statement," MySQL. Accessed on: Oct. 19, 2019. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/explain.html.

[29] P. M. Pardalos, D. Du, R. L. Graham, N. Katoh, and T. Ibaraki, "Resource Allocation Problems," in *Handbook of Combinatorial Optimization*, Boston, MA: Springer, 2013, pp. 905-1006.

Chapter 4

Efficient Execution of Egress Traffic Engineering Changes

4.1 Introduction

The Internet's main objective is to route network traffic, consisting of flow of packets, from a source node to a destination node. To achieve this objective, the set of packets are routed through routers and different network domains defined as Autonomous Systems (ASes). On a macroscopic level, the Internet is grouped into a set of domains, each consisting of thousands of routers, called ASes owned by network operators to facilitate the management of network traffic. Neighboring ASes are connected through Autonomous System Border Routers (ASBRs) using peer links. Based on the packets' destination addresses, some ASes are considered transit domains responsible for forwarding packets destined to other ASes. To determine the next hop peer link, border routers utilize Border Gateway Protocol (BGP) [1, 2] that includes many network-related metrics in its decision making process. These metrics include distance, transmission delay, and the number of hops that allow routers to identify the best next hop relative to any destination.

Given the enormous variability in network services, their traffic flows cannot be steered through

ASes in a uniform manner [3]. For that, different ASes form contractual relationships that route an equitable share of traffic. Accordingly, each AS determine its respective expenses while satisfying the quality requirements of network applications. This requires averting from the conventional shortest path algorithms used for network traffic routing which may cause link congestion on shortest paths [4]. To tackle this problem, network routers are augmented with Traffic Engineering (TE) [5] capability that steers network traffic on alternative routes different than the conventional ones.

Egress Peer Engineering (EPE) is the TE process that steers traffic exiting one AS network to a peer AS in the most cost effective way [5]. Generally, the objectives for different interdomain TE depend on the type of domain or AS under study. For transit domains, the main objective of interdomain TE is to maximize their profit by optimizing the utilization of their network resources while satisfying the users' QoS requirements. To satisfy their main objectives under resource and QoS constraints, network operators devise traffic assignment plans that map prefixes defining network traffic to specific internal network routes. Due to the changing network conditions driven by the dynamicity of users' activity and alteration of peering contracts, traffic assignment plans need to be constantly updated, so that they will not drift away from the TE main objectives that incorporate business and technical requirements [6]. These circumstances impose network operators to analyze current network conditions to create a new traffic assignment plan that satisfies these objectives.

Despite the desirable properties of the new plan, applying large scale traffic assignment changes is risky and can cause many unfavorable results. Some combinations of these changes can lead to overloaded peer links resulting in packet losses and network disruptions. This can result in many implications not only on the users' perceived QoS but also on the trust relationship between peering ASes. Thus, Execution Plans (EPs) are formulated to offer a smooth transition from an obsolete old plan to a favorable new plan by breaking down the set of traffic assignment changes into small subsets called steps.

In this domain, current research prioritizes creating TE policies for inter-domain traffic and the introduction of Software-defined Networking paradigm [7] to optimize a set of business-related,

network-related objectives or a combination of both that change the traffic assignments. Network performance is only evaluated at the start and end states of the network but never during the transition phase between these states which is the main goal of EPs. A separate line of research focuses on updating the network routing behavior by applying network reconfigurations such that the network performance quality is preserved. This area of research is coined as "Consistent Network Update Problem" [8]. While the implementations of these network updates can be a source of inspiration for this study, they are still an alternative to Interior Gateway Protocol [9] responsible for routing traffic within each AS which is not the scope of this study. Furthermore, the approaches apply a flawed evaluation criteria that do not consider monetary cost constraints, bandwidth constraints and the frequent network reconfiguration overhead. All of these parameters are pertinent for evaluating network performance and ensuring its stability during migration process. As a result, optimization models that generate EPs for inter-domain traffic assignment changes are inexistent in current scientific literature which leaves network operators with costly, error-prone and potentially unsafe alternatives of manual traffic changes adjustments that follow trial-and-error method. Therefore, there is a need for the automation of EPE under well-formulated constraints that capture the infrastructural and business objectives of network operators. Moreover, with regards to EPs generation process, applying partitioning solutions that are viable in other areas of research, such as computation offloading, is oblivious to the constraints of the network environment. This study utilizes a toolkit that provides two plans whereby desirable execution plans need to be implemented to change the state of the network from costly old plan to a better new plan.

This chapter starts by providing a thorough overview about the literature and state-of-the-art in the field of study. The next section illuminates the system model. The section that follows presents the evaluation of execution plans. Next, the proposed algorithms are examined. After that, the experimental procedure and setup are outlined and the results are discussed. Finally, the last section includes concluding remarks.

4.2 Related Work

This section provides a thorough explanation of the related work that covers the topics of network traffic engineering, network consistent updates, application partition and computation offloading. Based on the reviewed literature, the last section pinpoints the shortcomings of the mentioned work and accordingly outlines the novelty of this work.

4.2.1 Network Traffic Engineering and Network Consistent Updates

A great deal of research was conducted to study interdomain TE especially with the introduction of new networking paradigms such as Software-defined Networking (SDN). In [10], the authors formulate a heuristic that chooses the set of egress routers, in a hybrid network that integrates SDN nodes, so that the maximum link utilization of the egress link is minimized. In the same context, Guo *et. al* [11] propose incremental migration of SDN nodes into the traditional IP networks. To achieve that objective, the authors utilize a heuristic, a genetic algorithm, which outputs the fraction and the location of routers to migrate to maximize TE benefits. Other works, such as [12] focused on devising dynamic TE methods to adapt to the unpredictability of traffic flows. Their main objective was to ensure balanced loads among different engineered paths of predictable and unpredictable traffic flows and minimize the maximum link utilization of networks.

In the context of consistent network updates, authors have addressed the migration of traffic flows from one network configuration to another through the introduction of intermediate stages that ensure this transition. Brandt *et. al* [13] create an algorithm that decides if a congestion-free network flows migration is possible given a start and an end network configuration. Two variations of the algorithms are devised according to the capacity of the links. The first, which is based on the findings of Hong *et. al* [14], is implemented if there exist a free capacity on the links. The second considers a flow augmentation approach to deal with full capacity networks. Dionysus [15] aims at dynamically scheduling rule-based network configurations depending on a real-time analysis of the

behaviour of switches. The algorithm developed is based on the four pillars of consistent network updates [16]. The migration strategy is evaluated according to the link's bandwidth capacity, the switches' memory resources, and the network configuration's update process runtime. The authors of [17, 18] both implement a heuristic that is executed multiple times according to the pre-defined split of migration sets. While the approach in [18] is evaluated based on minimizing the execution overhead inherently defined as the number of steps for migration, Ghorbani *et. al* [17] evaluated their approach based on links' bandwidth violation.

4.2.2 Application Partitioning and Computation Offloading

Existing literature related to task allocation mechanisms resembles this study's objective in finding methods that divide some service into tasks and offloading their execution to resource constrained infrastructure. Wang *et. al* [19] propose a solution for cooperative task execution in a mobile cloud environment considering the traffic between devices and tasks' data dependency. To that end, the authors use dynamic programming to partition the application into tasks and then utilize Best Fit Decreasing algorithm to offload these tasks to mobile devices. Similarly, in [20] the authors propose an approach that partitions an application into sub-tasks and offloads them to nearby edge nodes to minimize the latency and offloading failure due to communication disruptions.

The literature addressing partial offloading of applications, which partition applications to offloadable and non offloadable disjoint set of tasks depending on some set objectives, fall in the same category. In [21], the authors propose an algorithm that aims to find the optimal partitioning plan which reduces the total application execution time and time complexity while addressing the instability of network conditions. To that end, the application tasks are mapped into a graph representation that is iteratively decomposed by combining offloadable and non-offloadable vertices until the maximum performance gain is obtained. Similarly, in [22] the approach leverages the graph representation of application tasks. This time the objective is to minimize the energy con-

sumption while satisfying the completion time constraints. Finally, the authors of [23] formulate an optimization function whose decision variable determines the tasks to be offloaded. The main objective of the optimization function and the developed heuristic is to maximize energy saving and satisfy execution time deadline. In terms of problem formulation, the work by Sidebottom *et. al* [24] formulated heuristics that generate execution plans based on a graph representation of traffic assignment changes. A two-step process is adopted for execution plans generation. In the first step each partition represents a graph connected component. In the second step, further partitioning of the connected components is implemented and evaluated.

4.2.3 Novelty of this study

Regarding the TE related literature, this study focuses on completely different set of objectives. While the above discussed literature main purpose is to find a new TE method that satisfies a set of objectives, this study finds an execution plan of traffic assignment changes that are results of applying some TE policies. The network flow migration related literature have some limitations in their methodologies, evaluation criteria or results. The work [17, 18] assumed an exact number of migrations per step which can not be applied in real world scenarios because of the variability of network sizes and traffic flows. Given these circumstances, intelligent methods need to be implemented to gauge the number of steps for migrating network flows. Additionally, some of TE related work lacked incorporating either bandwidth constraints, runtime, or the number of traffic assignment changes per step in their evaluation criteria. While [16] considers the number of steps for completing the migrations, the calculated number grows exponentially with the decrease in the free link capacity. This incurs a significantly large implementation delay when considering normal traffic conditions where the links are utilized by barely 50%. Furthermore, [15, 17] did not group the migrations in a single step which results in significantly increasing the reconfiguration processes for large number of traffic changes.

In addition, applying the task partitioning and offloading schemes to traffic assignment changes does not consider the constraints of the EPE networks. One prominent limitation of the above-mentioned work is the random method used to partition the set of tasks which cannot be applied to the execution plan steps. Another limitation pertains to the task allocation paradigm that maps tasks to mobile devices sorted by indexing order. Furthermore, the literature related to partial partitioning of applications suffer from some limitations which hinders their implementation for the partitioning of the set of traffic assignment changes. The common limitation of these research efforts is that they only address the partitioning of the application into two disjoint sets; offloadable and nonoffloadable. This is considered an oversimplification of the current study that involves not only finding traffic changes per step but also the number of steps which is only considered two sets in the aforementioned works. In their environment of study, the authors do not consider the resource limitations of either the local or remote execution medium which is equivalent to peer link capacity in this problem.

Lastly, the application partitioning literature have a well-defined evaluation criterion that gauges a particular QoS metric (latency). Comparatively, this study evaluates different execution plans according to weighted quality metrics that consider design (number of steps and their sizes) and business aspects (monetary cost). The main drawbacks of [24] are that there is no heuristic that produces good experimental results, and the evaluation criteria include incomparable metrics.

4.3 System Model

In this section, the EPE network structure under study is first presented by an illustrative example. Then, the notations of the network elements are defined. Finally, the concepts of plan, traffic assignment and network costs are discussed.

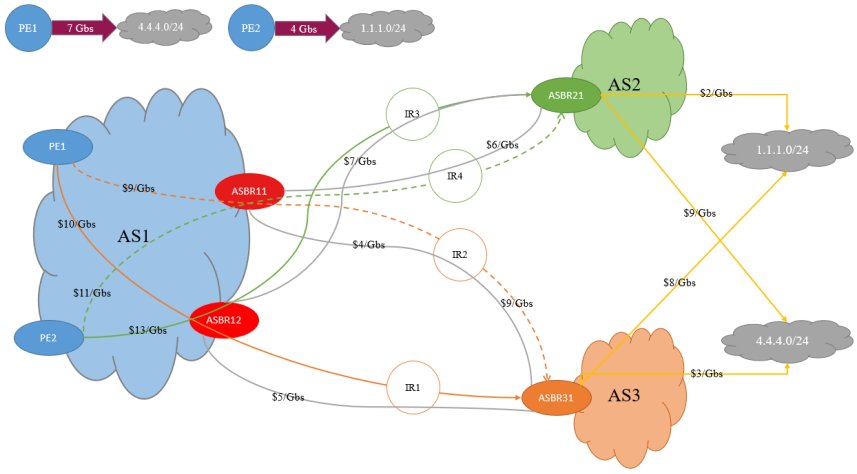


Figure 4.1: EPE Network

4.3.1 Illustrative Example

Figure 4.1 represents the reference model of an EPE network, where each of the Provider Edges (PE) in AS1 receives a traffic flow destined to AS2 and AS3. AS1 is connected to AS2 and AS3 using 2 peer links (PL) respectively depicted as grey arcs whereby each link has a limited bandwidth and cost defined by AS operators. The wavy arrows represent Internal Routes (IR) which form a many-to-many relationship between PEs and PLs. The yellow arrows denote the external routes (ER) that transports traffic from border routers connected to PLs to the destination prefixes. Each of the IRs, PLs, and ERs is associated with a cost function which is expressed as a motivating example by the charging rate per bandwidth (b) consumed. The cost function of IRs is denoted by ir, PLs by pl, and ERs by er.

To illustrate EPE network, an example of two traffic flows are presented in the upper left corner of the figure. The arrow denotes the bandwidth of the traffic to be engineered. As part of the initial traffic assignment plan, IR 1 and 3 were assigned to steer the traffic entering PE1 and PE2. Due to the dynamics of the EPE traffic, the cost of the initial traffic assignment plan does not satisfy the TE objectives anymore. As a result, a new plan that reduces the general cost of TE is devised which steers the traffic entering AS1 through IR 2 and 4.

To demonstrate the contribution of the new traffic assignment plan, the notation of the network structures involved and the cost calculations are as follows. Let $PE = \{pe_1, pe_2\}$, $PL = \{pl_1, pl_2, pl_3, pl_4\}$, $IR = \{(pe_1, pl_4), (pe_1, pl_3), (pe_2, pl_1), (pe_2, pl_2)\}$, $ER = \{(pl_1, pr_1), (pl_1, pr_4), (pl_2, pr_1), (pl_2, pr_4), (pl_3, pr_1), (pl_3, pr_4), (pl_4, pr_1), (pl_4, pr_4)\}$, $TR = \{(pe_1, pr_4), (pe_2, pr_1)\}$, $tr_{pe_1,pr_4} = 7$, $tr_{pe_2,pr_1} = 4$. Following the order of IRs and PLs, and ERs, the cost functions associated with each of their tuples is $ir(b) = \{10 \times b, 9 \times b, 13 \times b, 11 \times b\}$, $pl(b) = \{7 \times b, 6 \times b, 4 \times b, 5 \times b\}$, and $er(b) = \{2 \times b, 9 \times b, 2 \times b, 9 \times b, 8 \times b, 3 \times b, 8 \times b, 3 \times b\}$.

For the initial plan, the total cost for initial traffic assignment is calculated as follows. $cost_{pe_1,pr_4} = ir_{pe_1,pl_4}(7) + pl_4(7) + er_{pl_4,pr_4}(7) = 10 \times 7 + 5 \times 7 + 3 \times 7 = \$126$, $cost_{pe_2,pr_1} = ir_{pe_2,pl_3}(4) + pl_3(4) + er_{pl_3,pr_1}(4) = 13{\times}4 + 7{\times}4 + 2{\times}4 = \$88$. The total cost of the initial plan is \$214. Similarly, the cost of the new plan is as follows. $cost_{pe_1,pr_4} = ir_{pe_1,pl_3}(7) + pl_3(7) + er_{pl_3,pr_4}(7) = 9{\times}7 + 4{\times}7 + 3{\times}7 = \$112$, $cost_{pe_2,pr_1} = ir_{pe_2,pl_2}(4) + pl_2(4) + er_{pl_2,pr_4}(4) = 11 \times 4 + 6 \times 4 + 2 \times 4 = \$76$. The total cost of the new plan is \$188. The traffic assignment changes contributed in reducing the costs from \$214 to \$188.

4.3.2 EPE Network Elements

In the reference network, there are n PEs, m PLs, and l prefixes (PRs). PLs has an associated bandwidth. Let B denote the bandwidth where $B \in \mathbb{Z}^+$. Let I denote the set of PEs where $i \in I$. Let J represent the set of PLs where $j \in J$. K represents the set of PRs where $k \in K$. IRs are responsible for forming many-to-many relationship between PEs and PLs $IR = I \times J$. External routes (ERs) connect PLs to PRs $ER \subseteq J \times K$. Traffic flows (TR) associate PEs to PRs $TR \subseteq I \times K$ where $tr_{ik} \in B$ such that i represents a peer link and k represents a prefix. Each of these structures has a cost function (c) that maps a bandwidth (b) to a cost (C) ($c : b \longrightarrow C$). These cost functions are expressed as: c_{ij} represents the cost function of IR that connects $i \in I$ and $j \in J$. c_j represents the cost function of PL where $j \in J$. c_{jk} represents the cost function of ER that connects $j \in J$ and

$k \in K$.

4.3.3 Plans and Network Elements Costs

Each plan is a traffic assignment *ta* that maps TR expressed by the tuple (i, k) to a peer link denoted by $j \in J$.

$$ta(i,k) = j \mid (\exists (i,k) \in TR) \tag{4.1}$$

The cost of each network element ($C_{element}$) is equal to the bandwidth of the TR (tr) passing through this element as a function of the cost function associated with each element ($c_{element}$) which is expressed as $C_{element} = c_{element}(tr)$. The cost of assigning network traffic $C(ta)$ is equal to the sum of the cost of each network element; IR, PL and PR. As such, the cost of traffic assignment *ta* is the sum of the cost of the IRs $C_{int}(ta)$, the cost of PLs $C_{peer}(ta)$, and the cost of PRs $C_{ext}(ta)$.

$$C_{int}(ta) = \sum_{(i,j)\in IR} c_{ij}\Big(\sum_{((i,k),j)\in ta} tr_{ik}\Big) \tag{4.2}$$

$$C_{peer}(ta) = \sum_{j\in PL} c_{j}\Big(\sum_{((i,k),j)\in ta} tr_{ik}\Big) \tag{4.3}$$

$$C_{ext}(ta) = \sum_{(j,k)\in ER} c_{jk}\Big(\sum_{((i,k),j)\in ta} tr_{ik}\Big) \tag{4.4}$$

$$C(ta) = (4.2) + (4.3) + (4.4)$$

The main objective of the EPE planner is to change the state of the network from a traffic assignment *ta* to a better set *ta′* that reduces the overall cost. Therefore, the differences between *ta* and *ta′,* denoted by the function $\Delta(ta, ta')$*,* need to be identified. The function $\Delta(ta, ta')$ maps a traffic flow that is changing (i, k), between *ta* and *ta′*, to its destination peer link. The domain of the function $dom(\Delta(ta, ta'))$ represents all the traffic that is changing between *ta* and *ta′*. Any element

of $\Delta(ta, ta')$ assigns the traffic according to ta'.

$$dom(\Delta(ta, ta')) = \{(i, k) \mid ta(i, k) \neq ta'(i, k)\} \tag{4.5}$$

$$\Delta(ta, ta')(i, k) = ta'(i, k) \tag{4.6}$$

4.4 Evaluation of Execution Plans

4.4.1 Definition and Notation

An execution plan (ep) is a sequence of steps, each step includes a subset of the traffic assignment changes $dom(\Delta(ta, ta'))$. The defined steps are disjoint sets meaning that a changing traffic appears in exactly one step. Each ep is an ordered partition of the traffic that is changing where each partition member represents a step. The execution plan is denoted by $ep(ta, ta')$ where $\mid ep(ta, ta') \mid$ is the number of steps of this execution plan. ta_s is the traffic assignment after step s, $ta = ta_0$ is the start of traffic assignment before the first step and $ta_{|ep(ta,ta')|}$ is ta'. After defining a partition of the changes, the next stage is to order the partition elements or steps according to an optimization objective. Executing these steps in some sequence, changes the network to a desirable state. $ep(ta, ta')_s$ denotes the execution plan of step s. Also, $\Delta_{tr,s} = ep(ta, ta')_s$.

$$ep(ta, ta')_s = dom(\Delta(ta_{s-1}, ta_s)) \tag{4.7}$$

$$\bigcup_{s=1}^{|ep(ta,ta')|} ep(ta, ta')_s = dom(\Delta(ta, ta')) \tag{4.8}$$

4.4.2 Evaluation Metrics

In order to properly evaluate execution plans, a comprehensive set of metrics must be considered.

4.4.2.1 Progress Quality (q_π)

The evaluation functions should highly value the execution plans that reduce monetary costs as early as possible which implies that network state is transitioning rapidly to the desired state. This is of particular interest to network operators when execution plans take considerable time for its completion. In this case, network operators desire that the execution plans are reducing the costs early to avoid any network disruptions or violations of QoS requirements during the execution of the steps. PLs are constrained by their bandwidth. Therefore, execution plans should consider this limitation as overloaded PLs can incur huge packet loss and degradation of QoS. Evaluation functions should heavily penalize execution plans that can overload PLs while executing the steps. This condition is considered by the significant increase in the execution plan cost.

For progress component q_π, it starts by calculating the ratio π_s (Eq. 4.9) of cost reduction after step s ($C(ta_s) - C(ta')$) to the overall cost reduction $C(ta) - C(ta')$. It is notable to mention that π_s is equal to 1 if $C(ta_s) = C(ta')$,it is equal to zero if $C(ta_s) = C(ta_0)$ and $\pi_s \in [0, 1]$ if $C(ta_s) \in [C(ta_0), C(ta')]$. As such, π_s shows the contribution of step s in reducing the cost of an execution plan. To emphasize the importance of early reduction of cost, $q_\pi \in [-1, 1]$ (Eq. 4.10) is obtained as the sum of π_s of each step weighted by step order s.

$$\pi_s = 1 - \frac{C(ta_s) - C(ta')}{C(ta) - C(ta')} \tag{4.9}$$

$$q_\pi = \frac{\sum_{i=1}^{s} \pi_s}{s} \tag{4.10}$$

4.4.2.2 Balance Quality (q_r)

Execution plans that have a balanced number of traffic changes per step are desirable. The most balanced execution plan is the one that has equal number of traffic changes per step. As a result, network disruption is less likely under these conditions. $q_r \in [0, 1]$ measures how balanced are the the steps of an execution plan. For that, it first calculates (r) denoting the average difference

between the maximum and the minimum number of traffic changes among all the steps of an execution plan (Eq. 4.11) and then q_r (Eq. 4.12) is obtained by normalizing r between 0 and 1 such that bigger difference in traffic changes produces a worse result (closer to 0).

$$r = \frac{max_s|\Delta_{tr,s}| - min_s|\Delta_{tr,s}|}{2} \tag{4.11}$$

$$q_r = 1 - \frac{r}{r+1} \tag{4.12}$$

4.4.2.3 Ideal Step Size Quality ($q_{|\Delta_{tr,s}|}$)

There is a fine line that determines a favorable number of traffic assignment changes in each step. This condition is similar to the one considered by Sanvito *et. al* [25] where the authors argue about the drastic effects of frequent reconfiguration in the realm of TE. Large number of changes in a single step would accelerate the transitioning phase; however, at the risk of causing drastic network changes that the internal structure would not withstand. Conversely, small number of changes may result in a prolonged execution process which keeps the network in an unstable state for a long time, a situation deemed unfavorable for satisfying operators' quality objectives. Given the importance of this factor in measuring the quality of an execution plan, the evaluation function formulated incorporates a manually tuned ideal step size parameter. The automated tuning of this parameter is left for future research. The last metric $q_{|\Delta_{tr,s}|} \in [0, 1]$ describes the difference between the ideal step size $ideal_{|\Delta_{tr,s}|}$ and the size of each step given the total number of traffic changes $\sum_s \mid \Delta_{tr,s} \mid$ and the number of steps $\mid ep(ta, ta') \mid$ (Eq. 4.13).

$$q_{|\Delta_{tr,s}|} = 1 - \frac{\mid ideal_{|\Delta_{tr,s}|} - \frac{\sum_s|\Delta_{tr,s}|}{|ep(ta,ta')|} \mid}{ideal_{|\Delta_{tr,s}|}} \tag{4.13}$$

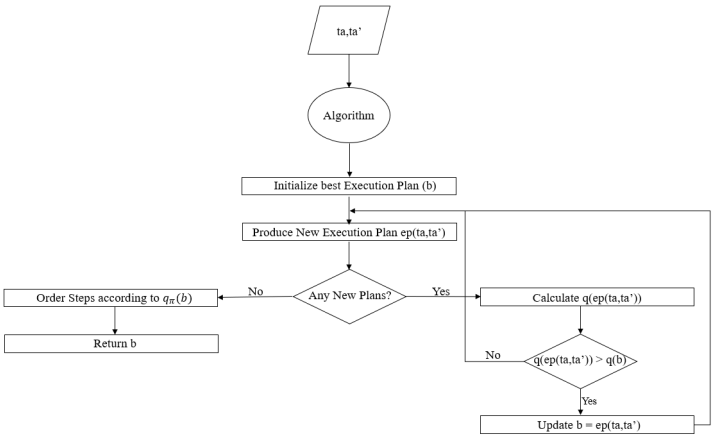


Figure 4.2: Flowchart for Evaluating Execution Plans

4.4.3 Process of Execution Plan Evaluation

The quality $q(ep(ta, ta'))$ of an execution plan is a weighted sum of three quality components: q_π, q_r and $q_{|\Delta_{tr,s}|}$ such that $w_\pi + w_r + w_{|\Delta_{tr,s}|} = 100$. Eq. 4.14 displays how $q(ep(ta, ta'))$ is calculated. For the evaluation process, q is normalized such that $q \in [0, 100]$. Each of these components address one of the aforementioned concerns for a good execution plan.

$$q(ep(ta, ta')) = w_\pi q_\pi + w_r q_r + w_{|\Delta_{tr,s}|} q_{|\Delta_{tr,s}|} \tag{4.14}$$

Figure 4.2 illustrates the process for evaluating the execution plans produced by any algorithm. As input, each algorithm takes ta as initial traffic assignment changes and ta' as the better traffic assignment changes. The first step initializes the best execution plan denoted by b. Upon producing new execution plans $ep(ta, ta')$, the quality $q(ep(ta, ta'))$ of each of these plans is calculated. In case the quality of the produced execution plan $q(ep(ta, ta'))$ is better than the best execution plan b, b is updated to be $ep(ta, ta')$. Until the algorithm has execution plans to produce, this process is repeated. When the algorithm has no more execution plans to generate, the steps of the execution plans denoted by b_s are ordered according to their contribution in reducing the overall cost denoted by $q_\pi(b_s)$ which reflects one of the conditions for a good execution plan. Finally, after ordering the steps of an execution plan, this execution plan is implemented and the respective quality metrics

are obtained.

4.5 Proposed Algorithms

4.5.1 Balanced Size Partitioning (BSP)

One approach for formulating the exhaustive search algorithm, used as a benchmark for comparing the formulated heuristics, is to generate all possible partitions of the traffic assignment changes defined as $dom(\Delta(ta, ta'))$. These partitions include the ways whereby a given set can be partitioned into subsets and the permutations of the elements of these subsets. However, the number of ways a given set can be partitioned explodes in a factorial fashion making this approach infeasible [26].

By definition, a good execution plan produces steps that are balanced in size, where equal number of traffic changes per step represent the most balanced execution plan. Following this premise and to mitigate the number of possible permutations, a divide and conquer approach was applied. This approach results in an algorithm that generates all possible permutations of balanced size steps. While the search space of BSP is still large, it poses a significant improvement over generating all possible permutations by discarding the unbalanced steps which serves the formulated evaluation criteria. For n number of traffic changes, the number of possible step sizes k is in the range of $[1, n]$. This formulation represents the Stirling number of the second kind [27]. For each of the step sizes k, the number of permutations is represented by Eq. 4.15 such that $\Gamma(n) = (n-1)!$ [28].

$$\prod_{i=0}^{k-2} \binom{n - \frac{i \times n}{k}}{\frac{n}{k}} = \binom{n}{\frac{n}{k}} \left(\frac{1}{\Gamma(\frac{k+n}{k})}\right)^{k-1} \Gamma(n + 1 - \frac{n}{k}) \tag{4.15}$$

Figure 4.3 shows an example of BSP where $X = \{1, 2, 3, 4\}$ is to be partitioned into balanced set of size $k = 2$. According to the formulation above the number of partitions is $\binom{4}{2}\binom{2}{2} = 6$.

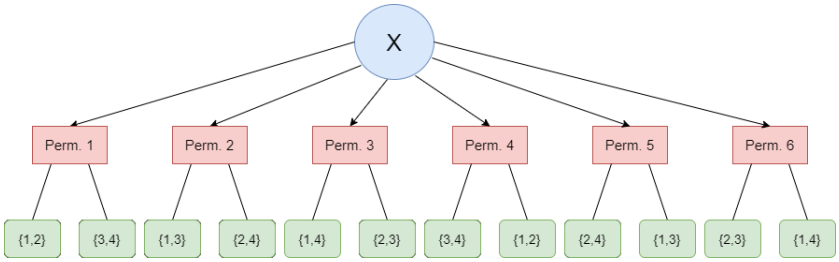


Figure 4.3: Example of BSP

4.5.2 Unbalanced Node Partitioning Heuristic (UNP)

The first proposed heuristic makes use of the graph representation of the traffic assignment changes denoted by Peer Links Change Graph $PLCG(ta, ta') = (V, E)$. In this representation, the vertices V are peer links and the edges E represent the traffic changes migrating from one peer link to another. The set of edges E is defined as $E = \{(j, j', (i, k)) \mid (i, k) \in dom(\Delta(ta, ta')), ta(i, k) = j, ta'(i, k) = j'\}$ such that $i \in I$, $k \in K$ and $j, j' \in J$ where $j \neq j'$.

Using this graph, the heuristic, UNP, shown in Alg. 4.1 calculates the difference in the number of inbound and outbound traffic flows defined as edges for each node and sorts them in decreasing order (line 2). Next, it chooses the nodes with the most difference in traffic denoted as the top unbalanced nodes (TU) and creates a partition of size that is equal to the top ones+1 (lines 4-6). After that, for each of these nodes n the set of edges e connected to it are part of a separate step (lines 7-11). All the unrelated edges to any of the unbalanced nodes are part of the last step P_{lenU} (lines 12-14). In case there exists common edges between the unbalanced nodes, the heuristic generates all possible combinations that attaches these edges to different k steps. Finally, SP is the set of partitions involving a combination of the top unbalanced nodes (lines 13-15).

The main intuition behind this approach is that there is an intrinsic reason for the high number of traffic flows involving a particular peer link. The reason is that these peer links are either overloaded which means that many changes involving them are classified as outbound or underloaded where most of the changes are inbound. These interpretations are based on that traffic assignment changes are devised to optimize the cost and the the resource use. Thus, isolating and executing

these changes first vastly contributes to reducing the network cost and optimizing the distribution of load over peer links.

The time complexity of this algorithm is $O(V^2E)$ where V represents the number of vertices and E denotes the number of edges or traffic changes. It creates $\frac{V}{2}$ number of partitions (line 4) such that in each iteration $\mid TU \mid$ is incremented by 1 till it reaches an order of V (lines 5-6). For each element of TU, all the edges E are iterated over. To formalize the time complexity between all the heuristics, the number of traffic changes are denoted by n which in this algorithm is equivalent to $\mid E \mid$. The best-case time complexity is attained when the number of nodes $\mid V \mid= 2$ which means that the complexity is $O(n)$. On the contrary, the worst-case complexity takes place when there exist only one edge between any two nodes which translates to $\mid V \mid= \frac{|E|}{2}$. In this case, the time complexity is $O(n^3)$.

Algorithm 4.1 Unbalanced Node Partitioning

```
1:  function UnblNodePartition((V, E))
2:      SN ← sortByDiff(V)
3:      SP ← ∅; TU ← ∅
4:      for i ← 0 to len(SN)/2 do
5:          P ← ∅; p ← ∅
6:          (TU, P) ← (TU ∪ SN_i, P ∪ p_1 ∪ ... ∪ p_{len(TU)+1})
7:          for k ← 0 to len(TU) do
8:              for e ∈ E do
9:                  P_k ← P_k ∪ e where e.j or e.j' = TU_k
10:             end for
11:         end for
12:         for e ∈ E do
13:             P_{len(TU)+1} ← P_{len(TU)+1} ∪ e where e.j or e.j' ∉ TU
14:         end for
15:         SP ← SP ∪ P
16:     end for
17:     return SP
18: end function
```

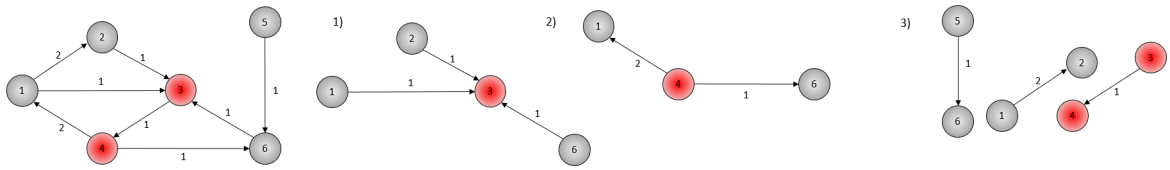


Figure 4.4: UNP Algorithm

Figure 4.4 illustrates an example of UNP. The first figure represents the graph representation of a set of traffic changes. Each node denotes a peer link and each edge in this diagram represents the number of traffic changes that altered their association from one peer link to another. The red nodes 3 and 4 are top unbalanced nodes where the difference between their inbound and outbound number of traffic flows is 2. The second figure which includes three separate graphs denoted by 1), 2), and 3) represents one of the potential steps that are partition members produced by UNP. In this partitions' generation process and following the notation of Algorithm 1, TU includes the nodes 3 and 4 and $\mid P \mid = 3$ which represents the number of steps. Next, the algorithm populates P_0 and P_1 representing the first and the second step in the diagram (lines 7-11). After that, the last step P_2, that includes all the edges not connected to either of the top nodes, is populated (lines 12-14). Lastly, P is appended to the set of all partitions. It is notable to mention that as part of the partitions' generation process, the common edges of the top unbalanced nodes (3 and 4) are isolated and distributed between steps. Each of these partitions represents an Execution Plan (EP).

4.5.3 Order Steps Heuristic (OSS)

This algorithm focuses on the intrinsic value displayed by the cost of a plan applying a set of traffic changes. As previously mentioned, the cost of a plan is the value of using the network elements that are part of the environment. In the implementation of an execution plan, the main concern is to avoid overloading peer links which is reflected by the eminent increase of the cost after applying an undesirable set of traffic assignment changes. Following this prelude, it is obvious that devising a proper heuristic, that considers the infrastructural limitations, should order the traffic assignments

in a way that reduces the costs. After that, the heuristic would partition the traffic changes ordering so that the resultant steps are as balanced as possible. This heursitic is called *OrderSteps*, and it consists of two steps shown in Alg. 4.2 and 4.3.

Alg. 4.2 returns the ordered set of traffic assignment changes denoted by $oTCs$. Firstly, it gets the set of traffic assingment changes denoted by TCs and computes the cost of assigning each of the individual traffic changes (i, k) to the traffic assignment plan ta using the $Assign$ function and stores the traffic assignment change and cost tuple in $costs$ (lines 6-9). After that, $costs$ elements are sorted in increasing order of assigning individual traffic costs, and the traffic change that contributes the most in reducing the cost, $asc\_costs_0(i, k)$, is assigned to the plan and appended to the ordered set of traffic changes (lines 10-13). The process is repeated recursively until the costs of the ta and ta' are equal.

This algorithm implements a recursive function whose input decreases by 1 with every iteration. The input represents the set of traffic changes $dom(\Delta(ta, ta'))$. In each iteration, the algorithm loops through all the traffic changes, thus incurring a time complexity of n representing the set of traffic changes $dom(\Delta(ta, ta'))$ (lines 7-9). Iteratively, this part will be equal to the sum of natural numbers $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$. Next, a merge sort is implemented of time complexity $O(nlogn)$ in line 10 which iteratively is equivalent to $\sum_{i=0}^{n}(n-i)log(n-i) = \Theta(n^2 logn)$. As a result, the time complexity of Alg. 4.2 is $O(n^2 logn)$.

The set of ordered traffic changes obtained from the first step is the input to the second step, shown in Alg. 4.3. Alg. 4.3 partitions the input set into balanced size steps and returns the set of execution plans denoted by sPs of different possible number of steps. The heuristic loops through all possible number of steps of an execution plan denoted by $size$. In that loop, the partition $part$ defining an execution plan and the size of each step sP are initialized (lines 4-6). Next, each of these partitions is populated according to its size with members of the ordered set (lines 7-10). Finally, each of the execution plans with $size$ number of steps is attached to sPs (line-14).

This algorithm takes order traffic changes $oTCs$ as input such that $\mid oTCs \mid = n$. The algorithm

Algorithm 4.2 Order Traffic Changes

Require: ta and ta' are initial and final plans

```
1:  oTCs ← ∅
2:  function OrderTrafficChanges(ta, ta')
3:      if c(ta) = c(ta') then
4:          return
5:      end if
6:      TCs ← dom(Δ(ta, ta')); costs ← ∅
7:      for ∃(i, k) ∈ TCs do
8:          costs ← costs ∪ ((i, k), c(Assign(ta, ta'(i, k))))
9:      end for
10:     asc_costs ← sortByCost(costs)
11:     ta ← Assign(ta, asc_costs_0(i, k))
12:     oTCs ← oTCs ∪ asc_costs_0(i, k)
13:     return OrderTrafficChanges(ta, ta')
14: end function
```

Algorithm 4.3 Partition Ordered Set

Require: Ordered Set of Traffic Changes $oTCs$

```
1:  function BalancedSizePartition(oTCs)
2:      sPs ← ∅
3:      for size ← 2 to len(oTCs)/2 do
4:          part ← ∅; s ← ∅
5:          part ← part ∪ s_1 ∪ ... ∪ s_size
6:          sP ← len(oTCs)/size
7:          for i ← 0 to size do
8:              for j ← i * sP to i * sP + sP do
9:                  part_i ← part_i ∪ oTCs_j
10:             end for
11:         end for
12:         sPs ← sPs ∪ part
13:     end for
14:     return sPs
15: end function
```

iterates over half possible number of partitions sizes equivalent to $\frac{n}{2}$ (line 3). In each iteration, the resulting number of iterations of lines 7-11 is equal to the number of traffic changes n. As a result, the time complexity of Alg. 4.3 is $O(n^2)$.

4.6 Experimental Setup and Procedure

4.6.1 Preliminaries

To evaluate the execution plans generated by the proposed algorithms, different networks with variable number of PEs, PLs and traffic changes were created. The Internal and External Route cost functions are equal to the distance (d) between the source and destination nodes each of the routes connect multiplied by the bandwidth (b). The model for creating the network includes a density parameter that represents the ratio of traffic bandwidth to the peer links' capacity. Existing traffic flows' bandwidth follows an exponential distribution that uniformly assigns them to the available number of PEs and PLs. Additionally, an ideal step size (isz) parameter representing the number of traffic changes in each partition is present. Throughout the implementation process, this parameter was manually tuned and is equal to some fraction of the total number of traffic changes.

4.6.2 Procedure

The best execution plan is produced by BSP as it generates all the possible balanced size partitions, and accordingly it is considered the oracle implementation used to compare and evaluate other heuristics. The heuristics were evaluated based on the quality q of their generated execution plans and their runtime. The small network was used to compare the results of the BSP, UNP and OSS algorithms due to the feasibility of producing the oracle execution plan within a reasonable period of time. Three variations of the small network were created: *Move*, *Bal*, and *3L2H*. The main difference between these networks is the distribution σ, representing the standard deviation, of the

Table 4.1: Small Network Configurations

Name	**PEs**	**PLs**	**Traffic Changes**	σ
Move	3	5	19	0.997
Bal	3	5	19	0.632
3L2H	3	5	19	0.79

number of traffic changes between peer links.

Each of the algorithms was evaluated on these network configurations combined with 50%, 70%, and 100% traffic density parameter which suggests mild and extreme traffic conditions [29] and eight different combinations of quality metrics weights and two assumptions of isz (7 and 10) for small size networks. After performing the comparisons, the best heuristic out of the two (UNP, OSS) was evaluated on a medium and large size networks.

To compare the oracle implementation versus the heuristics, execution plans were generated for three small networks whereby their respective metrics are presented in Table 4.1. The results obtained are the average of five independent runs. The implementations were compared based on the aforementioned evaluation criteria (q) and the run-time (ms). Depending on the run-time, the network operator would decide if the algorithm can offer solution offline or online. Due to network dynamicity, traffic assignments are expected to be updated constantly. Therefore, algorithms that take too long to execute its best solution might be solving a problem that does not exist anymore [8]. To reflect the state of the peer links and to show that the formulated evaluation criteria realistically mirrors the desired state of the network, the average utilization of the peer links per density parameter over the steps is reported.

4.6.3 Implementation

The process of creating networks, algorithms, and their evaluation was developed in Google's Go Programming Language [30] on a laptop with Intel Core i5, 3.1 GHz clock frequency and 16 GB of RAM. Traffic flows, old and new traffic assignment plans, peer links' capacity, and costs are

Table 4.2: Results on Small Networks

	Move		**Bal**		**3L2H**	
	q	**(ms)**	**q**	**(ms)**	**q**	**(ms)**
BSP	91.36	134,935	88.7	443,206	92.03	120,588
OSS	89.6	1.50	84.82	0.83	90.67	1.509
UNP	61.59	1.98	64.95	0.87	61.41	1.81

provided. To validate the results after implementing the best execution plans, a visualization tool that shows the process of executing each step provides insights about the state of peer links after each step. For each set of {network, algorithm, ideal step size, traffic density, weight metrics}, results of execution plans are generated and stored in a separate file. After that, a python script was created to extract relevant information from each of these files that includes the quality (q) of the best plan, run-time which is the difference in time between the start of the partitioning process and when the best plan is generated, and the ratio of average utilization of peer links for the best plan.

4.7 Results and Discussion

In addition to evaluating the algorithms, the following sections will discuss the effect of quality metrics weights, traffic density, ideal step size, and indepedent runs on the performance of the algorithms. The following section discusses and compares the algorithms' speed in converging to their best result which is defined as the algorithms' acceleration. After that, the next section highlights the relationship between quality and runtime. Finally, the last section presents the performance results of the best heuristic on medium and large size networks.

4.7.1 Algorithms Evaluation

The results of implementing BSP, UNP and OSS algorithms on the respective small networks are presented in Table 4.2 and Figure 4.5. The algorithms are evaluated according to the quality of their best execution plan and runtime. With respect to different networks, it is observed that BSP

and OSS have performed better on networks that have more skewed distribution of traffic assignment changes over peer links ($Move$, $3L2H$) which is not the case for UNP. Traffic changes are distributed more uniformly for *Bal* which means that execution plans need to migrate a balanced number of traffic changes of each peer link per step to avoid overloading any of them. Compared to $Move$ and $3L2H$, OSS and BSP perform worse on *Bal* which means they did not fully abide by the best aforementioned policy to migrate traffic for this network. Regarding the runtime, it is observed that BSP takes more time to produce the best result compared to other heuristics irrespective of the network under study. In the same context, BSP takes considerably more time to generate the partitions necessary to produce the best result for *Bal* network which is due to the nature of this network that considers more equal distribution of traffic changes over peer links. This implies that BSP has to access a wider search space to find the best execution plan for *Bal* compared to other networks. OSS and UNP do not display any noticeable differences in this regard.

Comparing the algorithms, it is observed that UNP and OSS produce their best results faster than BSP. This can be attributed to BSP being a generative algorithm that tries every possible combination of balanced size partitions rather than being tailored to the problem which is the case for OSS and UNP.

For the quality metric, UNP was the worst performing algorithm. This is the result in underperforming in either q_π or q_r and $q_{|\Delta_{tr,s}|}$. While q_π shows the algorithm's capacity for the early reduction of the cost of plan execution, q_r and $q_{|\Delta_{tr,s}|}$ display the algorithm's ability to abide by the expected execution plan principles in terms of the balance in the size of different steps. The results are expected due to the fact that this algorithm progressively chooses the top unbalanced nodes according to the initial state of the network and does not update the graph state in terms of the inbound and outbound traffic flows of each node after top nodes' associated edges are removed. Therefore, after choosing the first top node, the second top node would not necessarily be a top node according to the top node definition which has implications on the balance between steps of an execution plan. This can degrade the q_r and $q_{|\Delta_{tr,s}|}$ metric performance. By definition, a

top unbalanced node has considerable difference between inbound and outbound traffic flows. As such, the execution step including the traffic changes of top unbalanced node means that they can be mostly either inbound or outbound to that node. In case they are predominantly inbound, this can lead to this peer link, represented by a node, being overloaded after step execution. This can prominently increase the cost of plan execution. Conversely, if most of the traffic changes are outbound destined to a specific peer link, this would overload the destination peer link incurring an increase in the plan execution cost.

In the same context, OSS falls short off BSP because of this algorithm's second step that partitions the ordered set of traffic changes into balanced subsets. For odd number of traffic changes, which is the case for the small networks, the implementation does not generate all the possible combinations of partition sizes. For instance, for 19 traffic changes and two step execution plan, two balanced size parititions are possible: 9 for the first step and 10 for the second or vice versa. However, this is not the case for OSS which results in minor differences with respect to oracle implementation as presented in Table 4.2.

Figure 4.5 depicts the ratio of the actual to the expected peer link capacity following the traffic density parameter (50%, 70% and 100%). The results presented by Figure 4.5 further consolidate previous interpretations of the quality metric results of the algorithms. While the good performing algorithms BSP and OSS display close to 1.0 ratio for both traffic densities, UNP noticeably falls behind. During steps' execution, the top unbalanced nodes are offloading their traffic flows, which appears to be mostly outbound, leading to their under-utilization. This explains why UNP ratio is considerably under 1.0. Conversely, BSP and OSS have maintained peer link utilization close to the expected value which is reflected by the 1.0 ratio. This shows that both of these algorithms successfully found the combinations of traffic flows to be included in each step so that the bandwidth constraints are not violated or that they do not exceed their expected utilization. These results emphasize the validity of the evaluation criteria put forth which enforces the balanced split of traffic assignment changes that contributes in restraining bandwidth constraint violations.

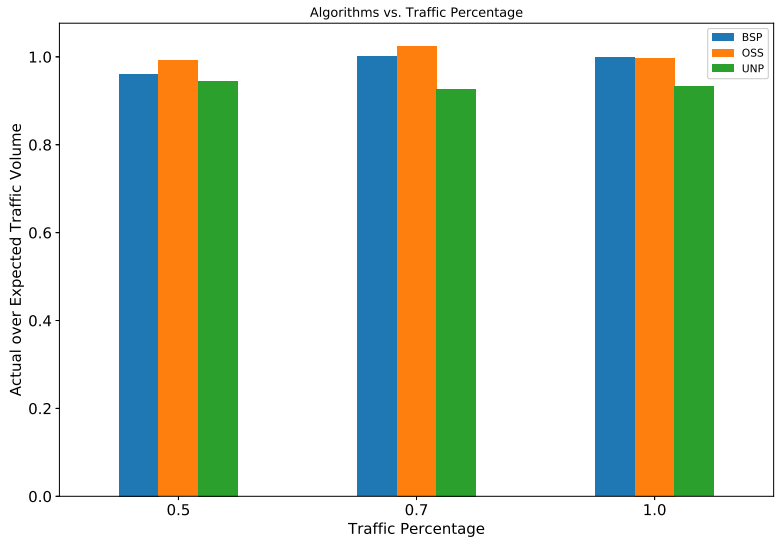


Figure 4.5: Ratio of Actual to Expected Peer Link Capacity for Different Algorithms

4.7.2 Effect of Quality Metric Weights

To compare the performance of algorithms, eight different combinations of quality metrics were used. The process of choosing the weights relied on distributing them in a way that ensures that none of the parameters is fully dominant on the other two. In this way, algorithms produce results that can be generalized in terms of the effect of each of the progress, balance radius, and ideal step size on the quality of their respective execution plans. The distribution of weight study can be leveraged by network operators based on their priorities for TE objectives and the network state. Assigning larger weight for progress metric w_π implies that network operators are prioritizing their business goals in terms of reducing TE costs over design objectives, represented by the ideal step size $w_{|\Delta_{tr,s}|}$ and balance quality w_r. Prioritizing any of the aforementioned metrics has implications on the network state and users' perceived QoS. Business objectives are of particular interest for network operators when execution plans are expected to run for an extended period of time which is the case for large networks with significantly large number of traffic changes. In this case, network operators prefer execution plans that can significantly decrease the costs without compromising the network state by including many reconfigurations in a single step. On the other hand, assigning

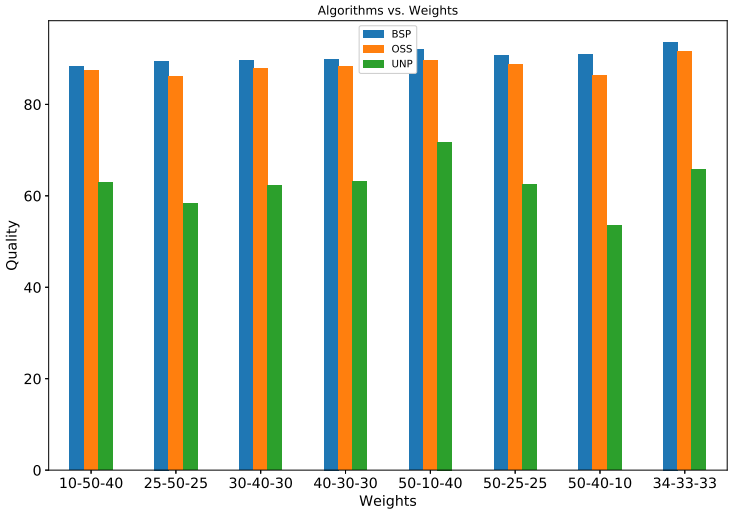
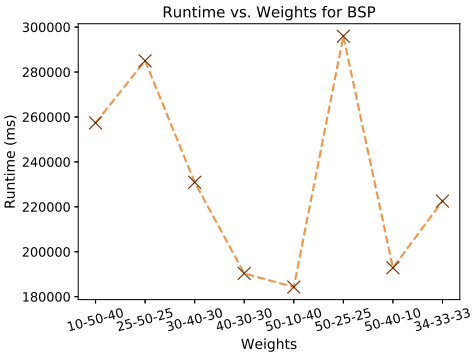
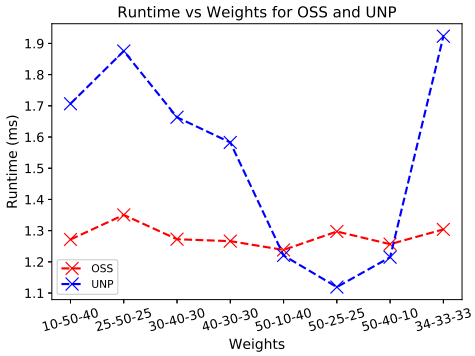


Figure 4.6: Weight Effect on Algorithm Peformance



(a) Runtime vs Weights for BSP



(b) Runtime vs Weights for OSS and UNP

Figure 4.7: Weight Effect on Runtime

higher weights for design-related metrics means that network operators prefer ensuring the safe and smooth transition from one plan to another. Guaging these metrics relies on a deep knowledge and thorough inspection of the capabilities of the defined internal routes and their respective switches to withstand drastic changes. Depending on the urgency of the business or design objectives which are based on the network state, the network operators decide the distribution of quality weight metrics.

Figure 4.6 shows the effect of calibrating weights applied in the following order $w_\pi - w_r - w_{|\Delta_{tr,s}|}$. The weights can be categorized into three different priorities: high (50%, 40%), medium (30%, 25%, 33%), and low (10%). Regardless of the algorithm applied, a general trend is observed where the performance improved with the increase of the progress weight w_π from low (10%) to high priority (50%). This means that the algorithms performed relatively well, compared to other metrics, in terms of reducing the cost progressively and as early as possible during step execution. For BSP and OSS, the improvement of their performance with respect to increasing the weight of w_π is not as significant as the improvement showed by UNP. This shows that BSP and OSS performance is not prominently affected by the design metrics which is not the case for UNP. It is observed that all algorithms are negatively affected by increasing the priority of the balanced quality metric (w_r) (50%, 40%), which is attributed to the extreme sensitivity of this metric in networks with small traffic changes. A difference in one traffic change would drastically degrade the quality of q_r. Conversely, the algorithms' performance benefitted from increasing $w_{|\Delta_{tr,s}|}$ from 10% to 40%. Additionally, when the execution plan evaluation is weight-independent, which the case for $34-33-33$, BSP and OSS produce their best execution plans. From these observations, it is clear that there exists a trade-off between business-related metrics (w_π) and design-related metrics (w_r and $w_{|\Delta_{tr,s}|}$), which is more prominent for the graph-based algorithm UNP, that need to be calibrated following the network's operator objectives.

Figure 4.7 shows the effect of different quality metric weights on the runtime of BSP, OSS, and UNP. For BSP, the time to find the best execution plan is significantly affected by w_r. The

Table 4.3: Effect of Ideal Step Size

	isz=7		**isz=10**	
	q	**(ms)**	**q**	**(ms)**
BSP	90.17	359,311	91.23	103,529
OSS	86.03	1.42	90.7	1.14
UNP	62.39	1.63	62.89	1.49

runtime decreased when the w_r decreases from 50% to 10%. BSP displays a similar trend when increasing w_π from 10% to 50% which is a direct result of the first observation. Increasing the value of w_r forces BSP to try more combinations of balanced size partitions to get the best result. A similar trend is observed for OSS despite the margins being prominently reduced and with minor differences regarding the combinations of weights that increase the runtime. Same observations apply for UNP.

4.7.3 Effect of Ideal Step Size Parameter

This section describes the effect of the step size on the quality of the best execution plans produced by each algorithm. The importance of this parameter stems from the ability of the underlying network infrastructure to bear extreme disruptions in the number of traffic changes per step. Given these circumstances and due to the absence of a clear representation of the state of the underlying network infrastructure, the ideal step size was manually tuned to include a fraction of the total number of traffic changes per step. In the case of a small network, the algorithms were evaluated on two assumptions of the ideal step size (7 and 10) representing one third and half of the total number of traffic changes (19).

Table 4.3 presents the results of the algorithms with respect to each of the ideal step sizes. In terms of performance, the heuristics OSS and UNP improve with less execution steps involved. With 10 as the ideal step size, there exist only two combinations of the balanced number of traffic changes in a single step; it is either 9 in the first and 10 in the second or vice versa. However, this cannot be applied when considering 7 as the ideal step size which produces more combinations

Table 4.4: Effect of Traffic Density Parameter

	0.5		**0.7**		**1.0**	
	q	**(ms)**	**q**	**(ms)**	**q**	**(ms)**
BSP	93.16	146,436	89.76	151,569	88.82	439,832
OSS	92.19	1.23	88.88	1.32	84.73	1.28
UNP	64.69	1.73	60.81	1.48	62.79	1.48

of the number of traffic changes per step. Due to these circumstances, the ideal step size quality metric ($q_{|\Delta_{tr,s}|}$) is much more prone to aggravate the differences between the desired and the actual step size when small number of changes per step is involved. In terms of runtime, *BSP*'s time taken to produce its best execution plan is cut in half when considering more traffic changes in a single step. This can be attributed to the reduction of the search space for this algorithm which facilitates the production of the best result. For 7 as the ideal step size, *BSP* has to produce all the combinations of balanced size partitions of size 2 before producing the ones of size 3 which will naturally mean that the runtime will be greater than that for 10 as the ideal step size.

4.7.4 Effect of Traffic Density Parameter

This section outlines the effect of traffic density parameter, representing the traffic volume percentage of peer link's capacity, on different algorithms in terms of the quality metric q and runtime (ms). In current implementation, mild and extreme traffic volume conditions were assumed which put the algorithms under both realistic and rigorous network traffic volumes. This allows network operators to evaluate their systems under unprecedented increase in traffic which needs swift but efficient polices to deal with. Such an increase in traffic can be encountered in the case of broadcasting unplanned live events which necessitate contingency plans to cope with the exponential increase in the number of viewers.

Table 4.4 presents the results of the algorithms as a function of varying traffic densities. In terms of runtime, *BSP* algorithm took triple the time needed to find the best execution plan with 1.0 traffic density compared to the networks with 0.7 and 0.5 traffic densities. This algorithm

had to access wider search spaces to generate its best possible results with the increase of traffic density. In terms of quality, both BSS and OSS's quality have deteriorated with the increase of traffic density. Both OSS and BSP quality of execution plans decreased with the increase in traffic density. For OSS, the decrease in the quality was by about 10% from 92.19 for 50% traffic density to 84.73 for 100% traffic density. Compared to OSS,BSP's decline in performance with the traffic density is less drastic but still noticeable (4.6%).

This decrease is to be expected given that the algorithms while executing the steps should reduce the cost progressively which requires moving the traffic between peer links in some way to avoid overloading them. With 1.0 traffic density, implementing a hard line on the steps so that the peer links' capacity is not violated is challenging. This degradation of the quality proves that it is a conclusive indicator of not only the design requirements of an execution plan but rather on the state of the peer links.

Regarding UNP, there is no clear correlation between the traffic density and the quality of its best execution plan. This shows that the main issue for UNP's bad performance is not only the possible overload of peer links, but rather not adhering to design quality metrics. These result show that due to the distribution of the migrating traffic bandwidth, violating any of the quality metrics is inevitable.

4.7.5 Effect of Independent Runs

Figures 4.8 and 4.9 show the effect of the independent runs on the algorithms performance in terms of quality (q) and runtime. Each of the configurations {network, ideal step size, traffic density, weight metrics} was evaluated five times independently for each algorithm. These figures depict the average quality and runtime for each of these runs of each algorithm. These configurations were executed five times to detect the existence of possible huge variations in quality and runtime between one run and another. Regarding the quality, the performance of UNP remained almost

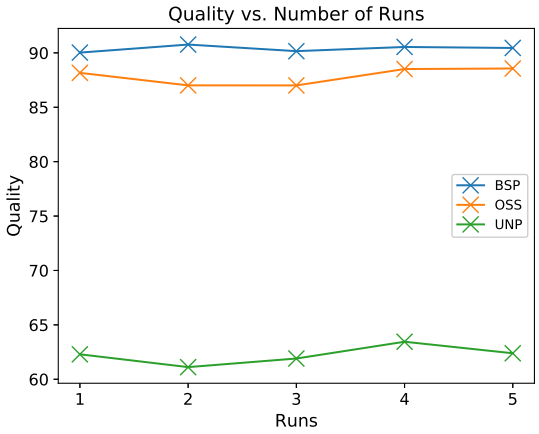
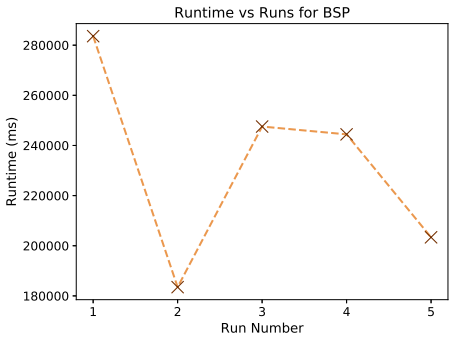
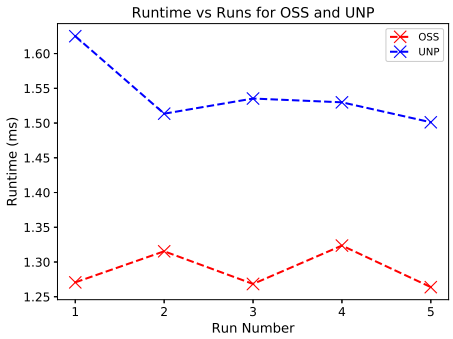


Figure 4.8: Quality vs. Independent Runs



(a) Runtime of BSP vs. Independent Runs



(b) Runtime of OSS and UNP vs. Independent Runs

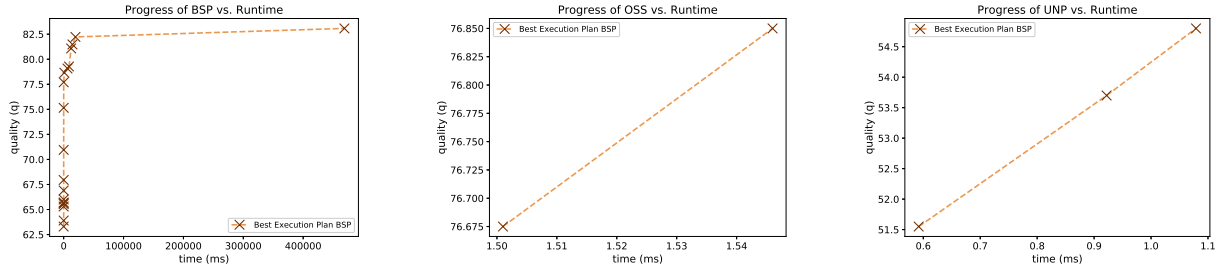
Figure 4.9: Runtime vs. Independent Runs

static with some minor differences displayed by runs 2 and 4 that are slightly worse and better than others respectively. Similarly, OSS's second and fifth run slightly perform better than the rest of the runs. For BSP, the first is the worst performing run while the second run is the best. The small differences can be attributed to rounding the resultant quality when extracting the results. However, these small differences do not produce much of discerpancy compared to the average of all runs.

Figure 4.9 shows the performance of algorithms in terms of runtime with respect to independent runs. The differences between runs are more significant for BSP. While the first run needs on average around 285,000 ms to produce its best execution plan, this value noticeably drops for the second run such that it hovers around 185,000 ms. For the third to the fifth run, the differences are less significant whereby the runtime is between 210,000 ms and 240,000 ms. The differences can be attributed to the process of allocating, freeing, and occupying threads as the program is written to execute all of these algorithms leverages the concurrency capabilities offered by Go. It is observed that for BSP, thread-related processes were relatively faster for the second run. Due to the negligible differences (fractions of a millisecond) of the independent runs for OSS and UNP, the above-mentioned interpretation for the variations does not hold. However, a valid explanation for the differences can be attributed to background processes that are executed synchronously which may cause some delays during execution of the program.

4.7.6 Algorithms' Acceleration

While the previous sections showed the superiority of UNP and OSS in producing their best results in significantly shorter period, this section will study each of the algorithms' capacity to progress rapidly to their best execution plan. It is intended to fairly compare BSP's progress and quality of its execution plan in the same time frame whereby UNP and OSS produced their best results. As previously outlined in section IV, each of the algorithms generate execution plans that



(a) Best Execution Plan of BSP vs. time

(b) Best Execution Plan of OSS vs. time

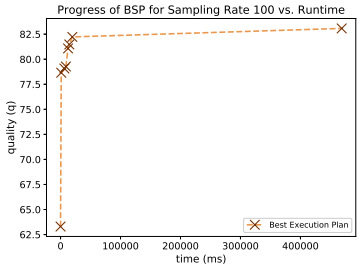
(c) Best Execution Plan of UNP vs. time

Figure 4.10: Best Execution Plan vs. Runtime

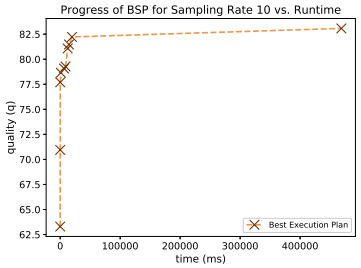
are compared according to their quality. Each interim best execution plan quality and runtime is recorded as part of the process.

Figure 4.10 shows the best execution plans produced by each of the algorithms for the network configuration of ideal step size 7, traffic density 1.0, network Move, and weights 50-25-25. According to previous sections, this combination took the most time to produce the best execution plan for BSP. For this reason, it was chosen for comparison purposes. In each of the figures, the label X denotes the best execution plan produced at that point of time with that quality. As a first observation, BSP (Fig.4.10-a) produces significantly more execution plans than OSS and UNP. This is due to BSP being a generative algorithm. The first execution plan produced by OSS (76.675) has a better quality than both BSP (63) and UNP (51.5). These initial results discard the comparison between UNP and BSP due to UNP's extremely poor results. Such a comparison is important for network operator when extremely fast network reconfigurations is required with good quality which resembles the one produced by OSS. Despite BSP's slow start, it displays a significant progress in producing better quality results. In a short period of time, the quality of the best execution plan improved from around 63 to close to 80.0.

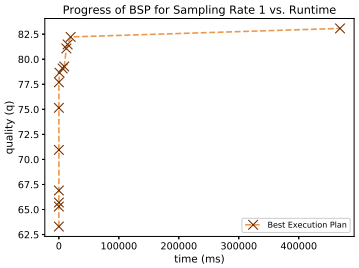
In order to better visualize the progress of BSP, sampling of the best execution plan according to its time is implemented. To that end, sampling rates of 100, 10, and 1 ms are considered meaning that each X in the diagram represents the best execution plan within the timeframes defined. The



(a) Best Execution Plans for Sampling Rate 100 of *BSP* vs. time



(b) Best Execution Plans for Sampling Rate 10 *BSP* vs. time



(c) Best Execution Plans for Sampling Rate 1 *BSP* vs. time

Figure 4.11: Best Execution Plan After Sampling for *BSP* vs. Runtime

main intention of the sampling process is to accurately determine the time window where *BSP*'s performance starts to get near to *OSS*'s. The sampling rate reduced the number of the best execution plan to 13 from 19 for 100 ms sampling rate, 9 for 10 ms, and 7 for 100 ms. The first execution plan is added to the each of the figures to showcase the progress of *BSP* within the sampling rates. For 100 as the sampling rate, the best execution plan quality within the first interval is around 79 which is better than *OSS*. However, since the time frame is significantly larger than the runtime of *OSS*, the sampling rate is reduced to 10. In the case for 10 as the sampling rate displayed by Fig.4.11-b, within 2 intervals *BSP* produces a better-quality execution plan (~78). Such results are not satisfactory, and they do not highlight the magnitude of the complexity difference between *BSP* and *OSS*. To achieve that, a stricter sampling rate of 1 ms is applied. As shown in Fig. 4.11-c, *BSP* needed five execution plans according to the sampling rate to achieve a comparable result (76) to *OSS* (76.8). However, according to the definition of a sampling rate, this does not exhibit differences in complexity. As a result, an interval [0 : 50] is chosen to show the best execution plans produced by *BSP* to compare it to *OSS*.

Figure 4.12, shows the best execution plans of *BSP* within the [0 : 50] ms interval and compares it with *OSS*. To produce better execution plan, *BSP* needs 50 ms which is 33x the time required by *OSS*. This further emphasizes on *OSS*'s ability to produce much better in significantly shorter period of time.

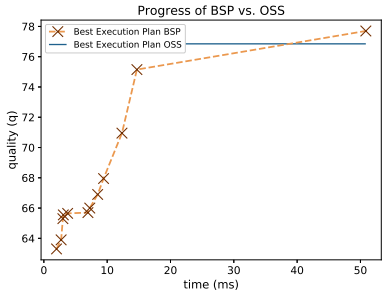
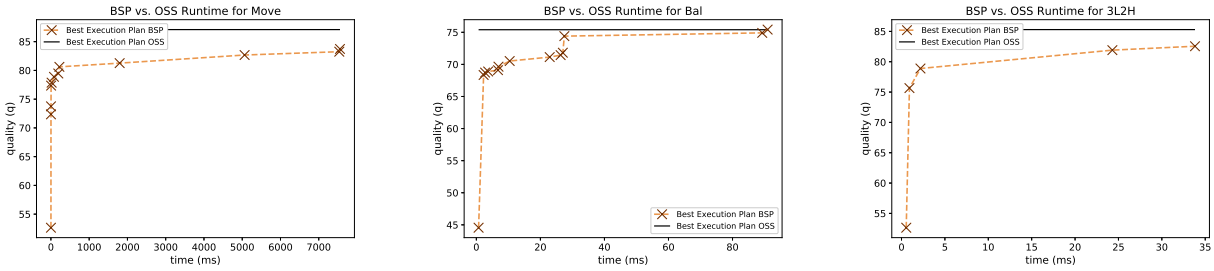


Figure 4.12: Best Execution Plans for BSP vs OSS



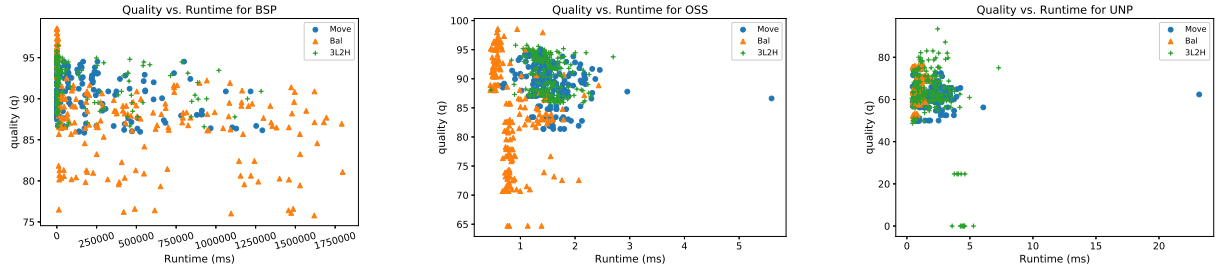
(a) BSP vs. OSS for *Move* (b) BSP vs. OSS for *Bal* (c) BSP vs. OSS for $3L2H$

Figure 4.13: Best Execution Plans for BSP vs. OSS

Fig 4.13-a, 4.13-b and 4.13-c show BSP's progress for the best execution plan compared to OSS. The network configuration is similar to that of Fig. 4.12 except the weights are $34-33-33$ with different networks. The results depicted by Fig. 4.13 emphasizes the superiority of OSS in terms of producing very good execution plans in significantly less time compared to BSP. For OSS, the time required to produce the best result is $\{1.25, 0.505, 1.3\}$ ms compared to $\{7, 200, 90, 33\}$ ms for BSP to get to comparable results. While Figures 4.12 and 4.13 only exhibit one of the runs for a single network configuration, they clearly demonstrate that OSS is much more efficient algorithm in producing good execution plans compared to BSP.

4.7.7 Quality vs. Runtime

This section oversees the potential relationship that may exist between the quality of execution plans generated and the time taken to generate these plans. If any correlation is found, the network operators can leverage such findings for predictive performance according to runtime which allows



(a) Quality of BSP vs. Runtime (b) Quality of OSS vs. Runtime (c) Quality of UNP vs. Runtime

Figure 4.14: Quality vs. Runtime

them to set termination time for execution plans where specific quality is desirable. The analysis is carried out on each individual algorithm for all the network configurations. Fig. 4.14-a, 4.14-b and 4.14-c are scatter plots that depict the quality of execution plans (q) as a function of their runtime (ms) for BSP, OSS and UNP respectively.

For BSP, it is observed that the best execution plans are concentrated in $[0 : 5,000]$ ms runtime with some relatively poor execution plans within the 75 range. The quality of execution plans is more distributed and dispersed in the interval $[250,000 : 750,000]$. For the interval $[750,000 : 1,000,000]$, the quality of executions plans is more concrete and falls in the range of $[88 : 92]$. After 1,000,000 ms runtime, the quality experiences the same disparity as in $[250,000 : 750,000]$ with significantly less data points. Also, it is observed that the network *Bal* is prone to extreme fluctuations in runtime due to the change of other network parameters. Most of the execution plans implemented on network $3L2H$ are executed within a small period. While there is no universal relationship existing between the quality of execution plans of BSP and their runtime, it is evident that some correlation can be inferred for specific time intervals. These intervals may belong to network configuration where some of the parameters vary while the others remain unchanged such as the used network parameter.

For UNP, no correlation between quality and runtime can be extracted as the dispersion is huge for any interval. Execution plans implemented on $3L2H$ experience the most dispersion in terms of quality, and those implemented on *Bal* experience the less dispersion in terms of runtime.

Table 4.5: Results of OSS on Medium and Large Size Networks

	rand75			**rand2010**		
	q	**(ms)**	**Ratio**	**q**	**(ms)**	**Ratio**
OSS	90.97	4.4	1.002	93.167	906,821	1.000

Regarding OSS, good execution plans in the [88 : 97] range are produced when the runtime is really small (<1ms). For runtime significantly < 1ms, there exists a positive relationship between the runtime and the quality especially with a single network involved (*Bal*). However, fast approaching runtime of 1 ms, a negative correlation exists between quality and runtime where execution plans mostly belonging to network *Bal*. For other intervals, no clear correlation exists between quality and runtime.

4.7.8 Effect of Network Size

Since the results on the small network have proven the superiority of OSS over UNP in terms of quality and over BSP in terms of runtime while producing comparable results, OSS was further tested on medium and large size networks. The reason for performing this rigorous testing is to evaluate the scalability of OSS when the network size and the associated traffic changes increase. The medium size network denoted by *rand*75 consists of 7 provider edges, 5 peer links and 44 traffic changes. The set of $isz = \{22, 15, 11, 9\}$. The large size network denoted by *rand*2010 consists of 20 provider edges, 10 peer links and 1,000 traffic changes. The set of $isz = \{250, 125, 60, 30, 15\}$. OSS was evaluated under the same conditions of traffic density and the combinations of quality metric weights of the small network. The results of these experimentations are presented in table 4.5.

The results show that OSS is performing well in terms of quality of execution plans, run-time and ratio of actual to the expected peer link capacity. As for both networks, the OSS produced high quality (91 and 93) results meaning that the heuristic generated an execution plan that reduces the cost as early as possible while balancing the number of traffics between partitions. Additionally,

these results showed that OSS can scale well with the increase in network size as it is producing its best result for the largest network in ~15 minutes. Furthermore, the ratio shows that the heuristic barely violated (0.02%), $\frac{ratio-1}{1} \times 100$, the constraints imposed on the maximum allowed utilization of peer links.

Compared to the small networks, the OSS performs better in terms of the quality of execution plans. This can be attributed to the balanced quality and ideal step size quality metrics. Due to the significant increase in the number of traffic changes, small differences between the number of these changes per steps themseleves or between these changes and the ideal step size will not drastically affect the quality metric involved as opposed to small networks with small traffic changes. An execution plan of quality 100 is unattainable because that means that it fully reduced the cost in a single step while adhering to the design parameters set. Executing a plan in a single step violates the balance quality and ideal step size quality metrics which results in a clear degradation of the quality of an execution plan.

4.8 Conclusion

This study addressed the efficient implementation of execution plans that promise to change the network to a more desirable state from the users' and network operators' perspectives. To that end, this study formulated three different algorithms based on mathematical and analytical understanding of the problem. Furthermore, evaluation criteria were put forth to study the efficacy of the execution plans under peer link bandwidth and operator costs constraints. The approaches were first tested on a small-scale network that served to elect the best performing heuristic. An extensive analysis of the effect of different parameters involved in the execution plan evaluation on the performance of the developed algorithms was conducted. Next, the best heuristic was re-evaluated; this time on networks with more traffic changes to test the reliability and the scalability of this approach. The results have shown that one of the proposed heuristics, Order Steps (OSS), produces

significantly better results while guaranteeing network-related constraints.

Bibliography

[1] Y. Rekhter, T. Li and S.Hares, "A Border Gateway Protocol 4 (BGP-4)", IETF, RFC 4271, Jan. 2006.

[2] N. Feamster, J. Borkenhagen and J. Rexford, "Guidelines for interdomain traffic engineering", *ACM SIGCOMM Computer Communication Review*, vol.33, no. 5, pp.19-30, 2003.

[3] S. Li and J. Huang, "Price Differentiation for Communication Networks," *IEEE/ACM Transactions on Networking*, vol.22, no.3, pp. 703-716, June 2014, doi: 10.1109/TNET.2013.2258173.

[4] K. Wang *et al.*, "Betweenness Centrality Based Software Defined Routing: Observation from Practical Internet Datasets," *ACM Transactions on Internet Technolology*,19 (2019), 1-19.

[5] D. Awduche, A. Chiu, A. Elwalid, I. Widjaja, and X. Xiao. 2002. *Overview and Principles of Internet Traffic Engineering*. RFC 3272. RFC Editor

[6] N. Feamster, J. Winick, J. Rexford, "A Model of BGP Routing for Network Engineering," in *Proc. ACM SIGMETRICS*, June 2004.

[7] A. Mendiola, J. Astorga, E. Jacob and M. Higuero, "A Survey on the Contributions of Software-Defined Networking to Traffic Engineering," *IEEE Communications Surveys Tutorials*, vol. 19, no. 2, pp. 918-953, Secondquarter 2017, doi: 10.1109/COMST.2016.2633579.

[8] K. Foerster, S. Schmid and S. Vissicchio, "Survey of Consistent Software-Defined Network Updates," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1435-1461, Secondquarter 2019, doi: 10.1109/COMST.2018.2876749.

[9] S. Misra, S. Goswami, "Interior Gateway Protocols," in *Network Routing: Fundamentals, Applications, and Emerging Technologies* , Wiley, 2014, pp.131-157, doi: 10.1002/9781119114864.ch6.

[10] K. Kadiyala and J. Cobb, "Inter-as traffic engineering with SDN," in *IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2017, pages 1-7. IEEE, 2017.

[11] Y. Guo, Z. Wang, X. Yin, X. Shi, J. Wu, and H. Zhang, "Incremental deployment for traffic engineering in hybrid SDN network," in *Proc. IEEE 34th International Performance Computing and Communication Conference(IPCCC)*, Dec. 2015, pp. 1-8.

[12] Y. Takahashi, K. Ishibashi, M. Tsujino, N. Kamiyama, K. Shiomoto, T. Otoshi, Y. Ohsita, and M. Murata, "Separating Predictable and Unpredictable Flows via Dynamic Flow Mining for Effective Traffic Engineering," in *IEEE International Conference on Communication*, 2016, pp. 1-7.

[13] S. Brandt, K. Forster and R. Wattenhofer, "On consistent migration of flows in SDNs," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, San Francisco, CA, 2016, pp. 1-9, doi: 10.1109/INFOCOM.2016.7524332.

[14] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN", in *Proc. ACM SIGCOMM Conf.*, pp. 15-26, 2013.

[15] X. Jin *et al.*, "Dynamic scheduling of network updates", in *Proc. ACM Conf. SIGCOMM*, pp.539-550, 2014.

[16] R. Mahajan and R. Wattenhofer, "On consistent updates in software defined networks," in *ACM SIGCOMM HotNets Workshop*, 2013.

[17] S. Ghorbani and C. Matthew, "Walk the line: consistent network updates with bandwidth guarantees", in *Proc. 1st workshop on Hot topics in software defined networks HotSDN 12*, pp.67-72, 2012.

[18] N. Katta, J. Rexford and D. Walker, "Incremental consistent updates", in *ACM SIGCOMM HotSDN Workshop*, 2013.

[19] X.Wang, Y. Sui, C. Yuen, X. Chen, C. Wang, "Traffic-Aware Task Allocation for Cooperative Execution in Mobile Cloud Computing", in *Proc. Of IEEE/CIC International Conference on Communications*, 2016.

[20] J. Liu and Q. Zhang, "Offloading Schemes in Mobile Edge Computing for Ultra-reliable Low Latency Communications," *IEEE Access*, vol. 6, pp.12825-12837, Feb. 2018.

[21] H. Wu, W. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions of Parallel Distributed Systems*, vol. 30, no.7, pp.1464-1480, Jul. 2019.

[22] M. Deng, H. Tian, and B. Fan, "Fine-granularity Based Application Offloading Policy in Small Cell Cloud-enhanced Networks", in *IEEE International Conference on Communications Workshops (ICC)*, 638-643, 2016.

[23] S. Cao, X. Tao, Y. Hou and Q. Cui, "An energy-optimal offloading algorithm of mobile computing based on HetNets," in *International Conference on Connected Vehicles and Expo (ICCVE)*, Shenzhen, 2015, pp. 254-258.

[24] G. Sidebottom, R. Nekvi and A. Haque, "Safely Engineering Egress Traffic Changes".

[25] D. Sanvito, I. Filippini, A. Capone, S. Paris and J. Leguay, "Adaptive Robust Traffic Engineering in Software Defined Networks," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*, Zurich, Switzerland, 2018, pp. 145-153, doi: 10.23919/IFIPNetworking.2018.8696406.

[26] C.Wagner, "Partition Statistics and q-Bell Numbers," *Journal of Integer Sequences*, Vol. 7, (2004).

[27] 1.8 Stirling numbers. [Online]. Available: https://www.whitman.edu/mathematics/cgt online/book/section01.08.html. [Accessed: 11-May-2020].

[28] "Gamma Function," from Wolfram MathWorld. [Online]. Available: https://mathworld.wolfram.com/GammaFunction.html. [Accessed: 12- May-2020].

[29] J. Sushant and *et. al* ,"B4: Experience with a globally-deployed software defined wan", in *Proc. ACM SIGCOMM 2013 conf on SIGCOMM*, pp.3-14,2013.

[30] "Go", [Online]. Available: https://golang.org.

Chapter 5

Conclusions and Future Directions

This thesis covered two problems pertaining to the automation of network traffic assignment changes and placement of vehicular applications. Each of these problems required finding the best possible solution for a specific objective under environment-imposed constraints. The two related articles presented novel ideas and implementations and promising results in their respective fields. The first study investigated the individual components of vehicular applications and devised an optimization model responsible for the placement of these components considering the delay requirements of the applications and the resource requirements of their components in edge environment. Two different implementations of an optimization model were compared, and one of these implementations, RDP, showed that it can satisfy the set requirements in terms of resources and delay. The second study examined the implementation of inter-domain network traffic assignment changes subject to monetary and infrastructural constraints. Three approaches were created and tested based on evaluation criteria that best reflect the imposed environment constraints. The best performing algorithm, Order Step Size (OSS), satisfied the quality requirements for big networks in significantly short period of time and produced comparable results to the oracle implementation for small networks. The final chapter of this thesis outlines some of the shortcomings of the conducted work and proposes future directions that can be examined and experimented

with in each study.

5.1 V2X Applications Placement in Edge Computing Environment

This article presented the first study that examined the structure of vehicular applications and the placement of its components. While Resource and Delay-aware V2X service Placement (RDP) produced satisfactory results on average for all applications, it violated the delay requirements of the Forward Collision Warning application in 25% of the cases. For that and for other reasons, the future work in this field will address the following topics:

1. A scalability study of the RDP approach that considers an environment with significantly more edge servers used for the deployment of vehicular services. This will put the approach under test and will show if it can be implemented in more realistic scenarios where full highways are considered. Also, this testing environment will examine the viability of this approach to be applied in real-time scenarios by reporting the run-time required for solving large-scale problems. In case long run-time is observed, this opens the horizon for creating intelligent heuristics that can reduce the run-time but can produce comparable and satisfying results.

2. A Data-driven approach that utilizes Machine Learning (ML) algorithms can be leveraged to address the possible scalability issues that may arise. Data can be gathered by running the formulated optimization models under different environment parameters which include in the current formulation: edge server computing resources, delay between edge servers, and vehicle density. This method has been applied in the domain of network function placement which displays some similar characteristics to V2X applications placement [1, 2]. Additionally, Moubayed *et. al.* [3] have emphasized the applicability of ML in the domain of V2X

communications.

3. Because the best proposed algorithm did not fully satisfy the delay requirements of all applications, a vertical extension of computation resources can mitigate this shortcoming. One approach for this extension is the inclusion of centralized cloud computing resources used for the placement of extra vehicular services. Furthermore, the idle resources of vehicles can be opportunistically leveraged to deploy vehicular services extending those deployed on edge servers. Conceptually, such an architecture spans an extensive research effort that encompasses the formation of vehicle clusters virtualizing their resources and studying the heterogeneous communication protocols mandated by this architecture.

4. While the research conducted is a good first step in this field, the future work should integrate more dynamic and complex environment elements. Here, the main elements targeted are the vehicles and the edge servers that displayed static characteristics in terms of requesting services and the availability of their resources respectively. The future work will integrate service request distribution that reflects realistic scenarios and examines the correlations that may exist between different applications. Similarly, after considering service request distributions, the availability of computation resources should mirror the requests' distribution and the vehicle density in proximity to each edge service. This new more dynamic and complex environment will inspire more intelligent methods for the deployment of V2X services.

5.2 Efficient Execution of Egress Traffic Engineering Changes

This article presented a comprehensive study to automate the implementation of egress network traffic assignment changes efficiently. The efficiency of these implementations is measured based on the evaluation criteria that integrate monetary concerns and bandwidth and infrastructural con-

straints. The best algorithm referred to as OSS adhered to the quality metrics of the evaluation criteria while producing these results in significantly short period of time; however, it overloaded the peer links by small margin for small and medium size networks. Therefore, the future work in this topic will address the following points:

1. Evaluate the best performing algorithm (OSS) using a wider range of network configurations that include the size of the network, number of traffic changes, and cost functions. This will test the viability of this solution in more rigorous scenarios especially when considering traffic changes in the order of 1,000s. Additionally, a more comprehensive set of cost functions will be considered that include but not limited to linear cost functions, piecewise functions, and exponential functions. The myriad of combinations possible for the cost functions can mirror different scenarios applied by network operators to map the bandwidth passing through their domain peer links.

2. As a direct consequence of the slight overload of the peer links displayed by OSS, future work will address this shortcoming by integrating the temporary bandwidth consumed by peer links when some combination of traffic is executed into the evaluation criteria. Adding this parameter is pertinent to close the gap that can be caused by a cost function that is not representative of the peer links condition. Weight distributions of the metrics will be applied and evaluated in a fashion similar to the current implementation.

3. Consider more refined graph-related mapping of traffic assignment changes and apply partitioning algorithms that jointly account for the graph structure and the set evaluation criteria. The graph representations will include incorporating multi-commodity flow and max-flow principles and adjusting them so that they can suit the defined problem. In the same context, vertices' related characteristics such as the number of edges and total bandwidth change between vertices can be leveraged to draw similarities or differences between them. The relationships extracted using similarity studies can be utilized to group set of vertices in separate

steps as part of an execution plan.

4. Current implementation considers that the network traffic flows defined are changing their assignment from one peer link to another. Future work will integrate the network flows that have changed their association from being unassigned to assigned to a peer link or vice versa. The inclusion of these traffic flows will consolidate the authenticity of the networks in mirroring real-life scenarios to test the approaches previously defined.

Bibliography

[1] D. M. Manias, H. Hawilo, M. Jammal and A. Shami, "Depth-Optimized Delay-Aware Tree (DO-DAT) for Virtual Network Function Placement," *IEEE Networking Letters*, doi: 10.1109/LNET.2020.2997320.

[2] D. M. Manias *et al.*, "Machine Learning for Performance-Aware Virtual Network Function Placement," in *2019 IEEE Global Communications Conference (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9013246.

[3] A. Moubayed and A. Shami, "Softwarization, Virtualization, Machine Learning For Intelligent Effective V2X Communications," *IEEE Intelligent Transportation Systems Magazine*.

Curriculum Vitae

Name:	Ibrahim Shaer
Post-Secondary Education and Degrees:	American University of Beirut Beirut, Lebanon 2017 Bachelor of Computer Science (With Distinction)
Honours and Awards:	Dean's Honour List American University of Beirut 2014-2017 USAID Scholarship American University of Beirut 2014-2015
Related Work Experience:	Graduate Teaching and Research Assistant University of Western Ontario 2018-2020 Research Assistant American University of Beirut 2017
Publications:	I. Shaer, A. Haque, and A. Shami, "Multi-Component V2X Applications Placement in Edge Computing Environment," in ICC, 2020. (accepted)