Efficient Resource Allocation Frameworks for Data and Service Delivery to Connected Vehicles in Vehicular Edge Computing

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

Joseph John Cherukara
2018111009
`joseph.cherukara@research.iiit.ac.in`

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
June 2024

International Institute of Information Technology

Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled “ **Efficient Resource Allocation Frameworks for Data and Service Delivery to Connected Vehicles in Vehicular Edge Computing** ” by **Joseph John Cherukara**, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date: June 29, 2024

Adviser: Dr. Deepak Gangadharan

Connecting Vehicles to Connect the World

Acknowledgments

First and foremost, I would like to thank God Almighty for showering his blessings upon me throughout this whole college journey. One of which was choosing the perfect advisor for me. I'm eternally grateful to my advisor, Dr Deepak Gangadharan, whose constant guidance, patience, support, and care were the major factors in completing my thesis. Thank you for pushing me and never giving up on me, even at times when I gave up myself. Your motivation and guidance every time I hit a roadblock led me to complete my research without giving up entirely. You were there for me every time I needed help, not only in research but also in my academic and personal life. Your teachings of not just about the various research domains but also how to present papers, what to look for during a paper reading, etc., are all invaluable lessons which have helped throughout my research journey, and I am sure will be of help in the future. Thank you for taking a chance and accepting me as your student.

I cannot begin to express my thanks to my senior Akshaj Gupta, who, along with Deepak sir, introduced me to the world of research and IoV, and helped me to get my first co-authorships, thus giving me the first taste of success in research. I want to express my deepest appreciation to my colleague Suhas, who supported me at every turn, be it writing the paper, running the experiments or brainstorming new ideas. I would also like to extend my deepest gratitude to Dr BaekGyu Kim [currently affiliated to DGIST, South Korea], who has been a great co-advisor for my research. Your suggestions and ideas helped us to publish our works successfully.

I'm incredibly grateful to my closest friends, Adarsh and Arpan, who also helped me with my research, writing the papers, coding and debugging, and all aspects of my college life. Many thanks to my lab mates Sridhar, Usha, Ashish, Rahul, Rhuthik, Pranav, Abhinav, Sesha and Pawan. You guys brought different ideas to the table and created our CPS research group.

I very much appreciate the guidance offered by professors Dr Oleg Sokolsky and Dr Insup Lee [both currently affiliated with UPenn, USA] in making our papers and published works. I also had the great pleasure of working with Dr Pradeep C and Ajai Mathew [currently affiliated to Saintgits, India].

Special mention goes to Dr Vignesh Sivaraman [currently affiliated with IIT- BHU, India] for his initial guidance on one of my projects.

I'm also grateful to have worked with Shri Sarvesh and Uma Damodaran, who collaborated with me as interns in my research.

I would also like to thank my dear friends Sahil, Shinde, Gautham, Anubhav, Srinath, Naren, George, Ashwin, Giri, Vignesh, Bhuvnesh, Gaurang, and the rest of my gang for your constant support. You guys brought all the joy to my college life.

I'm deeply indebted to my parents, who have been a constant source of support, love, and motivation in my life. Thank you for always encouraging me to reach new heights and never to lose hope. I am also grateful to the rest of my family, especially my scientist grandfathers, who inspired me to join the research field.

Thank you all for your contributions to this wonderful research journey and for completing my Master's thesis.

Abstract

As the world progresses rapidly in terms of innovation and technologies, one of the fields that has seen a significant change in the past few years is the Internet of Things. IoT has changed the scope of connectivity across the world. As more and more devices connect to the internet, the ease of use and level of automation increase for those devices, leading them to use the term ’smart’ device.

Similarly, one sub-field of IoT which has been getting a lot of attention is the Internet of Vehicles (IoV), primarily with the idea of bringing self-driven cars to reality. However, IoV, as with IoT, has its own set of challenges, constraints, and use cases to consider. Due to the increase in connectivity and sophisticated software, modern vehicles are able to leverage different kinds of services provided by the environment. These services include but are not limited to, data delivery and computation offloading. While cloud computing was initially considered to deliver these services, it was realized that communication with the cloud would require constant high bandwidth requirements and could incur high latency issues, especially while travelling through low network regions. Thus, to solve these issues, instead of Cloud Computing, Edge Computing was considered, as it brought the required computation units closer to the user vehicles, thus requiring lesser bandwidth and facing low latency. This technology of using Edge Computing to deliver services to connected vehicles while considering their dynamic network topology is called Vehicular Edge Computing (VEC).

VEC requires the use of multiple edge nodes throughout the road network to facilitate the services for connected vehicles. These edge nodes can also be connected to a central cloud server. However, VEC also has its own set of challenges and constraints to consider when delivering each type of service. Data delivery to connected vehicles would require the allocation of memory resources at the edge nodes that the vehicle will pass through and receive and store the data from the cloud to be delivered to the vehicle. It should also have enough bandwidth to transfer the data to the vehicle when it passes through the coverage region of that edge node. Computation offloading would require proper scheduling of tasks across all the edge nodes in the network so as to facilitate efficient execution of the tasks and ensure the results are delivered within the deadlines given the resource constraints. All these services have to consider multiple vehicles at the same time at the coverage of an edge. Thus, resources should be properly allocated to service the maximum number of vehicles.

This work tries to address these challenges by contributing solutions considering the vehicle flow constraints of the network as well as resource and timing constraints when delivering data to the vehicles or scheduling the tasks of the vehicles for computation offloading. The first solution proposes a two-stage optimization framework for efficient data delivery to connected vehicles via edge nodes while considering dynamic route changes. This framework optimizes the bandwidth utilized to send data from the cloud to the edge nodes. It also prioritizes vehicles with more data to receive and fewer edge nodes to pass through to reach their destination. The second solution proposes a Global Earliest Deadline First (GEDF) based scheduler for scheduling offloading tasks to the edge nodes. This approach considers the vehicle flow constraints, resource constraints, and timing constraints while assigning tasks to edge nodes that can execute the offloaded task, while prioritizing vehicles with shorter deadlines. Both these works have been tested on a real-world dataset (Luxembourg dataset) and compared with other approaches, including an optimal approach with various parameters such as the number of vehicles fully serviced, run time, etc.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 MCC and MEC

1.1.1 Mobile Cloud Computing (MCC)

There has been an explosive growth of application models such as Cloud Computing, Software as a Service (SaaS) etc, due to the advances in the field of network based computing[31]. Cloud computing, in essence, can be described as a variety of services provided by a group of low cost servers or Personal Computers (PCs) hosted in some remote data centre, that can be accessed by the local machine via internet. Cloud computing gained popularity as it provided ease of access to additional computational resources, anywhere in the world with internet access, thus allowing the execution of computation heavy tasks without having the infrastructure locally. Once smartphone technology rapidly advanced along with internet connectivity, we were able to have mobile devices with computational capacity of PCs and access to the internet, and by extension the cloud. This led to the rise of a new computing mode combining the mobility of smartphones and the accessibility of resources offered by Cloud computing, called Mobile Cloud Computing (MCC).

But with the advancement in technologies, it was identified that MCC has a few challenges to face. Namely the issue concerning latency, connectivity and energy consumption. To solve these challenges, a new paradigm called Edge Computing was introduces as a middle layer between the user devices and the cloud. This led to the rise of Mobile Edge Computing (MEC).

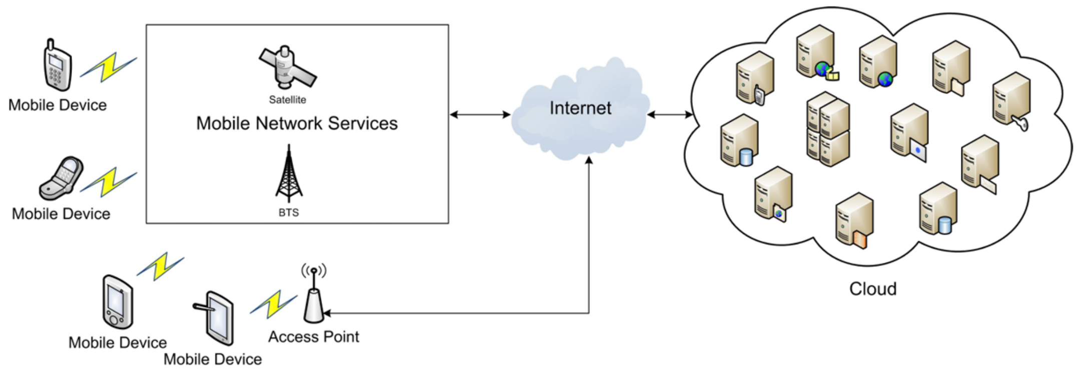


Figure 1.1: MCC Network [30]

1.1.2 Mobile Edge Computing

The advancements in internet and connectivity technologies, led to the rise of new paradigms such as Internet of Things(IoT) [4]. But as the number of devices connected to the internet increased, so did the amount of large-scale data being transferred. This led to cause problems such as increased bandwidth requirements, latency issues, privacy concerns, security concerns etc., in traditional cloud models. Thus, as traditional Cloud computing became insufficient to handle these new technologies, a new computing paradigm for performing calculations at the edge of the network was introduced called Edge Computing [7]. To handle mobile smart devices and their requirements and to facilitate IoT, MCC also similarly shifted to Mobile Edge Computing (MEC).

But MEC is too broad a paradigm, since it can consider almost any device, at the edge of the network, with computational capacity that can connect to the internet as either a user device/client or as and edge node or both. Thus, for specialized use cases, we would need a more specialized paradigm since requirements, constraints and conditions will change depending upon the type of user device, type of edge node considered and general environment of use. So, the scenario of considering moving vehicles to be the user devices and Road Side Units (RSUs) or other vehicles to be the edge nodes in a traffic environment, was termed as Vehicular Edge Computing (VEC). VEC considers the specialized requirements and constraints of moving vehicles, fast entry and exit through the coverage region of the edge nodes, limited resources of the vehicles and edge nodes etc unlike the general constraints and use cases of MEC.

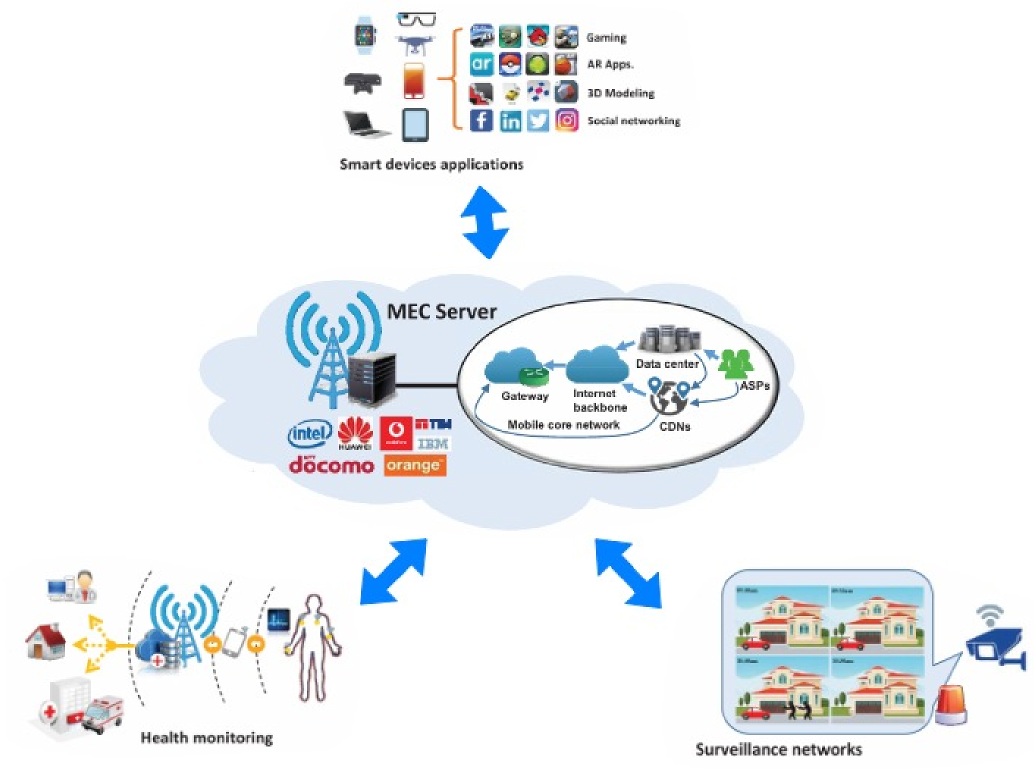


Figure 1.2: MEC Network [28]

1.2 Vehicular Edge Computing (VEC)

1.2.1 Connected Vehicles

The evolution of an automotive from a manually driven standalone platform to an intelligent platform that interacts with its environment has been primarily realized by novel advances in software and communication technologies. A connected vehicle is therefore able to communicate with other vehicles, devices, surroundings etc and thus help facilitate better comfort, security, learning and entertainment for the users. The connected vehicles ecosystem is a result of the rise of Internet of Vehicles (IoV) from Internet of Things (IoT). This advancement of the connected vehicles ecosystem brings with it, a whole new set of challenges and problems to be solved in terms of communication methods, security requirements, energy constraints, Quality of Service requirements etc.

1.2.2 Modes of Communication

The modes of communication in the VEC framework can be broadly divided into two categories based on the recipient of the communication from the vehicle. They are:

- **V2V communication**: When a vehicle is communicating with another vehicle in its vicinity, it is known as V2V communication.
  eg: A vehicle sending an accident nearby signal to other vehicles in its vicinity
- **V2I communication**: When a vehicle is communicating with the nearby infrastructure (buildings, traffic signals, roadside units etc), it is known as V2I communication.
  eg: Traffic lights giving priority to road segments with more vehicle density at an intersection based on the number of signals it receives from the vehicles at each raod segment

1.2.3 Services

The connected vehicles ecosystem provides multiple services which can lead to multiple applications which can enhance safety, security and comfort for the driver and passengers of the vehicles. These services include but are not limited to :

- Data delivery and communication: Connected vehicles facilitate the transmission of data across vehicles and infrastructure with the help of various protocols like DSRC (Direct Short Range

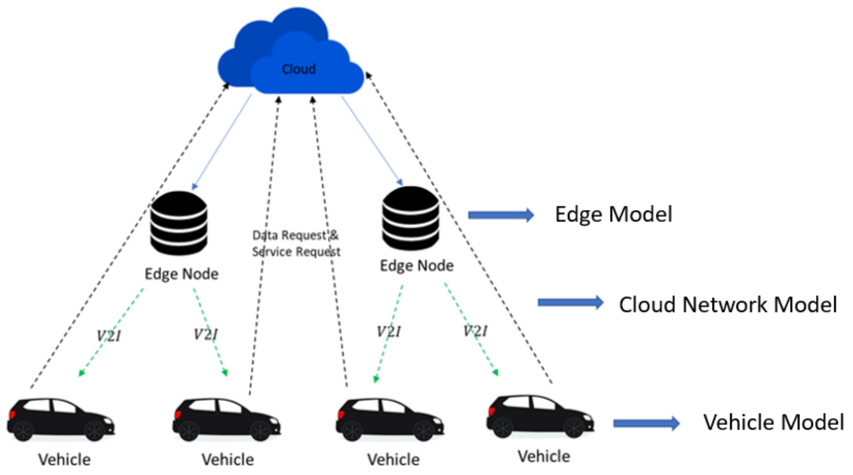


Figure 1.3: VEC Model

Communication). This data could be in the form of texts, requests, images, videos, signals etc. Thus connected vehicles can request for specific data from the nearby infrastructure, nearby vehicles and the cloud. This can include map data, movies etc.

- Computation Offloading: Resource intensive computational tasks can be broken up and transferred to nearby infrastructure or other vehicles to be processed and the results sent back to the original vehicle. This process is called Computation Offloading. This service of the connected vehicles ecosystem helps the vehicles to run heavy tasks, even with the resource constraints of each vehicle, by utilizing the free (non-working) resources of the surroundings to run part of the task or the whole task. This service is especially useful to run high computation tasks like image processing tasks or natural language text recognition tasks while on the move in the vehicle.

- Smart Data Collection: IoV facilitates the collection of data from the various sensors in the vehicle. This data could be terrain data, vehicle condition data, driving experience data etc. This can help the user to know about any problems at the earliest. This collected data can also be shared with the government for their analysis and actions in bettering road safety and maintenance, and with insurance companies for their applications.

The rise in the connected vehicles ecosystem and the data gained from it lead to the evolution of automated vehicles or self driving cars.

1.2.4 Applications

The connected vehicles ecosystem facilitates multiple applications that can enhance user experience, safety and security. Some of these applications are:

- HD Maps: The Automotive Edge Computing Consortium (AECC) [1] recommends using High Definition Map application [2] to obtain accurate real-time information on the streets and traffic conditions that would help in better navigation. The request for this map data is sent to and received from the nearby Roadside Units (RSUs) or the cloud.
- Entertainment: Connected vehicles enhance passenger experience by providing streaming services through the internet. These vehicles are able to access content in the internet directly, thus removing the need to use another device such as the phone or laptop while in the vehicle to watch movies, play songs etc.
- HUD: Heads-up Display or HUD can be useful in a vehicle, where the HUD provides additional information to what the driver can see. This can include warning signals to indicate another vehicle, person or obstacle is approaching the trajectory of the vehicle which is very helpful in blind spots or corners. It can also highlight pits, bumps or other such obstacles on the road, in the trajectory of the vehicle.
- Sign board translation: Connected vehicles with cameras can take a picture of the roadside signboards and use text recognition to identify its content. It can also help translate the text from regional languages while travelling. These tasks require high computational capacity and thus require offloading the tasks to nearby RSUs or vehicles to get the result in the required amount of time without loss in integrity of data.

1.3 Motivation

1.3.1 VEC

The connected vehicles ecosystem has many services and applications which enhance user experience. But these services have challenges that have to resolved to get the best results. Some of these challenges include how to handle situations where the vehicle is not connected to the internet at all times, and another is how to resolve the computation resources constraint. These challenges can be resolved to an extent by utilising VEC or Vehicular Edge Computing concept. Vehicular Edge Computing considers the use of edge devices to minimize latency issues that can arise during communication between cloud and vehicle. The edge devices can also provide extra resources for the computation purposes. The edge devices can include mobile devices, other vehicles and even infrastructure such as RSUs. In the following sections we will consider the edge servers/nodes that are used to be RSUs only.

1.3.2 Challenges and Solutions

- **Data Delivery**: One of the services of the connected vehicles ecosystem, the data delivery requires internet to receive the requested data. But there will be situations where the vehicle will not be able to access the internet for long periods of time during the journey due to loss of network. This can impact data delivery if we try to send data continuously. Instead of continuous data transmission we can utilize the edge nodes to transmit chunks of data to the vehicle when it comes into the coverage region of that edge. Thus instead of continuous data transmission, we split the data into multiple parts and each part is sent to the vehicle by an RSU it passes through during its travel. But to handle this data delivery framework, we need to consider some aspects such as the route of the vehicle, resource constraints of the edge node including bandwidth, memory etc, and time taken to send each part of the data requested. Our work proposes an optimal data delivery framework which takes into consideration all these aspects and is an online approach, thus able to handle the dynamic route changes of the vehicles.

- **Computation Offloading**: Another service of the connected vehicles ecosystem, computation offloading, requires the use of external edge devices which have resources that can be used for executing the task. In this scenario we consider only task offloading to RSUs. But for efficient utilization of all the edge nodes and their resources, we need to take into consideration the routes of the vehicles, the amount of usable resources of each edge node, the task deadline etc. Our proposed approach takes into consideration all these constraints and tries to schedule the tasks of the vehicles at the various edges in an efficient manner at the shortest amount of time.

1.4 Contributions

In summary, our key contributions are:

1.4.1 Dynamic Data Delivery Framework

1. *We propose a time slot based just in time dynamic data delivery framework.*

2. *We consider a vehicle flow model along with edge model and cloud network model.*

3. *The proposed approach is executed at each time slot and the data allocations are made to the edge nodes based on the locations of the vehicles at the start of each time slot. Thus it is able to handle any route changes that the vehicle makes in between its journey.*

4. *The proposed approach is compared with various existing optimal and heuristic approaches (using real-world data sets), thereby demonstrating its advantages.*

1.4.2 Global EDF Based scheduler for Computation Offloading

1. *We propose a time slot based Global EDF based scheduling algorithm*

2. *The proposed approach uses a priority queue to give priority to the tasks of the vehicles with the earlier deadlines*

3. *The proposed approach is compared with various existing optimal and heuristic approaches (using real-world data sets), thereby demonstrating its advantages in terms of execution time or number of vehicles serviced*

1.5 Thesis Organization

The remaining thesis is organized as follows :

In Chapter 2 we have briefly reviewed previous works on applications of VEC. In Chapter 3 we present the proposed two-stage optimization framework for data delivery to connected vehicles via edge nodes. In Chapter 4 we present the proposed heuristic scheduler that is based on of a popular scheduling algorithm, which incorporates the conditions and constraints of the task offloading scenario for connected vehicles. In Chapter 5 we conclude our work and propose a couple of new problems to consider in the Connected vehicles domain as future works.

Chapter 2

Related Works

This chapter presents the existing literature on Vehicular Edge Computing (VEC) followed by computation offloading and data delivery. This chapter then presents the existing approaches for task scheduling that can be adopted in a VEC scenario.

2.0.1 Vehicular Edge Computing

With the advent of Mobile Edge Computing (MEC), several low-latency services and distributed computing have become promising for data delivery and task-offloading methods. Acheampong et al. [3] have developed an online system that can offload a task in real-time scenarios. The authors have highlighted the improvement in the system performance with reduced computation time. Although the work has greatly emphasized the capabilities of MECs, in a vehicular scenario, it has not considered the wastage of allocated resources that can occur due to a change in the vehicle route. Unlike MEC, Vehicular Edge Computing (VEC) has a directed node as RSU or Edge which can be used for data delivery and task offloading [29].

2.0.2 Computation Offloading

Various works have emphasized the capabilities of VEC for computation offloading [29] [40][11][43]. Chen et al. [8] have proposed a three-layer end-edge-cloud architecture to assist vehicles for task computation. This work has formulated an efficient task allocation algorithm considering fairness, efficiency in computation and is evaluated considering the dynamic nature of tasks and computation resources. Although the proposed algorithm has improved the average time delay, a vehicular flow model is not considered, which is a crucial requirement for task allocation to multiple edges.

Many works have focused on energy and time delay optimization for task offloading. Sun et al. [32] have demonstrated considerable effectiveness in energy and time delay improvements. They have formalized a multi-stage optimization problem for predictive task offloading. Further, they have also used a genetic algorithm to reach an optimal global value. However, this work has not considered the effect of overlapping vehicles that can enter simultaneously for VEC services.

Tang et al. [33] have emphasized computation balance across multiple edges while optimizing the time delay. A greedy algorithm identifies a suitable offloading path with a lesser computation load while minimizing the timing requirements. However, it considers only static scenarios, which can differ significantly from a real-world data offloading scenario where the vehicle route can change.

In [25], two dynamic offline algorithms are introduced that showcase significant energy-saving capabilities using experiments in a Mobile Edge Computing (MEC) network. Additionally, in [23], the paper delves into the trade-off between reducing execution time and prolonging the battery life of mobile devices.

There are multiple strategies to address the task offloading problem. One such approach is to use a greedy heuristic method. A heuristic algorithm was used in [24] for task offloading to a nearby Mobile Edge Computing (MEC) server in order to minimize the task completion time. In [39], the authors attempted to minimize the total energy consumption for mobile devices by using a Select Maximum Saved Energy First (SMSEF) algorithm that chooses the most energy-saving task during each iteration.

Some have introduced diverse offloading strategies with varying objectives. The algorithm introduced in [26] focused on minimizing task execution time delay by using a 1-D search algorithm in the MEC network. This approach leveraged factors such as the application buffer queue state, mobile intelligent terminal resources, and edge computing network server capabilities to achieve the goal of reducing execution time to a minimum.

A machine learning (ML) based approach offers a more dynamic and adaptable solution for the task offloading problem with the drawback of increased computation costs [37]. ML models can be used to predict network conditions, workload characteristics, and server availability, allowing for improved decision-making regarding task offloading. In [19], a deep reinforcement learning-based joint task offloading and bandwidth allocation was used to minimize the total delay in finishing the tasks. A deep reinforcement learning-based task offloading in MEC was used in [36] to learn an optimal offloading strategy for minimizing the overall latency of the tasks.

In [42], the authors carried out a joint computation partial offloading and resource allocation algorithm for latency-sensitive applications in mobile edge clouds. The algorithm optimizes task allocation by dividing the application into two levels, cloud and edge, considering network bandwidth, computing and storage resource constraints. However, due to the high mobility of the vehicles, the algorithm needs to consider the information regarding the vehicle's route and the vehicle flow parameters.

But these works focus on using MEC networks and do not consider the vehicle flow model and vehicular constraints of VEC networks.

Multiple works have proposed different offloading policies, such as static, dynamic, and hybrid approaches, to optimize task execution in vehicular networks as seen in [34]. These studies have primarily focused on optimizing task placement decisions without considering the deadline of the tasks.

2.0.3 Data Delivery using VEC

The work by Gangadharan et al. [13] have introduced several constraints necessary for data delivery to connected vehicles via edges. The constraints addressed the memory and timing requirements for efficient data delivery through an optimization approach. The work considers the worst-case scenario that all vehicles with routes passing through an edge appear simultaneously in the edge coverage region. The work by Gupta et al.[16] have introduced several constraints necessary for data delivery to connected vehicles via edges and proposed a heuristic for the earlier work. The constraints addressed the memory and timing requirements for efficient data delivery through an optimization approach. Another work by Gupta et al.[17] proposed a cost-based gradient approach to handle data delivery. Here the overlapping vehicle set is populated to analyze the resource requirements, after which a heuristic approach is used to minimize the overall cost gradient for the bandwidth. This work uses pre-defined routes and considers an offline approach for data offloading and thus cannot handle dynamic route changes in real time.

Jeong et al. [21] have proposed a trajectory-based packet forwarding scheme to deliver data. The work considers the packet delivery delay distribution and the vehicle travel delay distribution to delivery data in a moving vehicle scenario. Wang et al. [38] have adapted an online resource allocation strategy for computation offloading. The authors have considered arbitrary user movement and resource price variation for efficient task offloading. Both [38] and [21] deal with a specific scenario where the vehicle's path is unaltered throughout its journey, allowing them to use the route information to optimize the data offloading. However, in a general scenario, a vehicle can randomly change its route to the destination depending on various factors.

2.1 Task Scheduling

The work by Yang et al.[41] proposes optimal GEDF-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. The work by Jiang et al[22] has adopted the GEDF approach for scheduling parallel real time tasks with arbitrary deadlines. This work is used for real-time systems with multi-core processors with the assumption that the task is split into multiple sub-tasks with the dependencies given in the form of a DAG. But these works, while giving an idea on how to use GEDF for task scheduling, do not look into using it in a distributed systems approach, and do not consider any of the constraints that are necessary in the VEC scenario.

Similarly other works cited in [5] such as [15],[12], [18] also look into task scheduling in Cloud computing, which do not consider the advantages and challenges of Edge Computing network. Our proposed approach aims to utilize the commonly used GEDF scheduling approach and modify it to work in the VEC framework following the constraints of vehicle flow model, timing constraints etc to schedule the computation tasks offloaded to the edge network by connected vehicles.

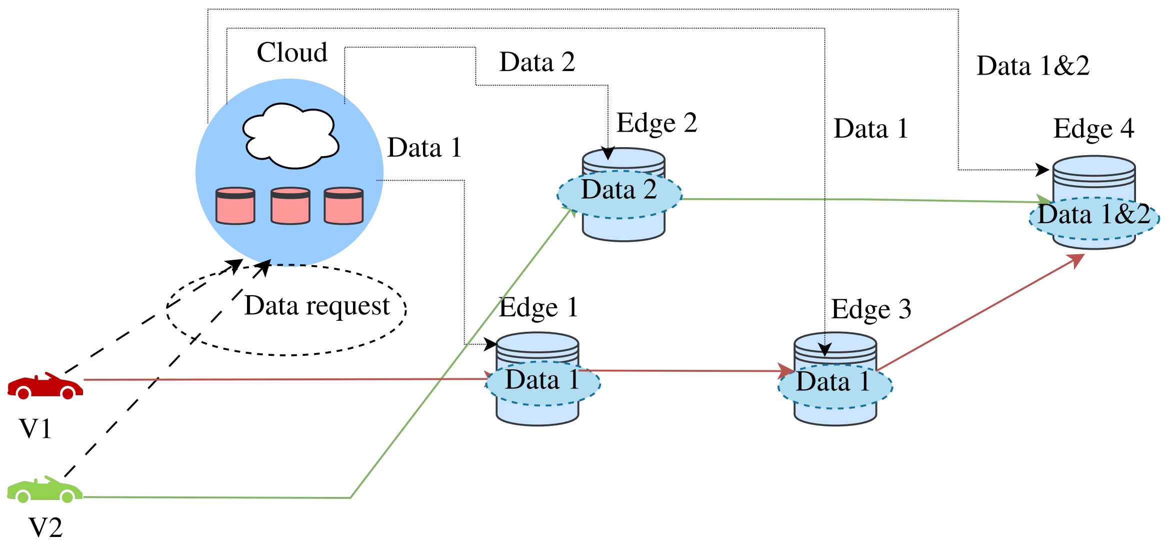
Chapter 3

Dynamic Data Delivery Framework for Connected Vehicles via Edge Nodes with Variable Routes

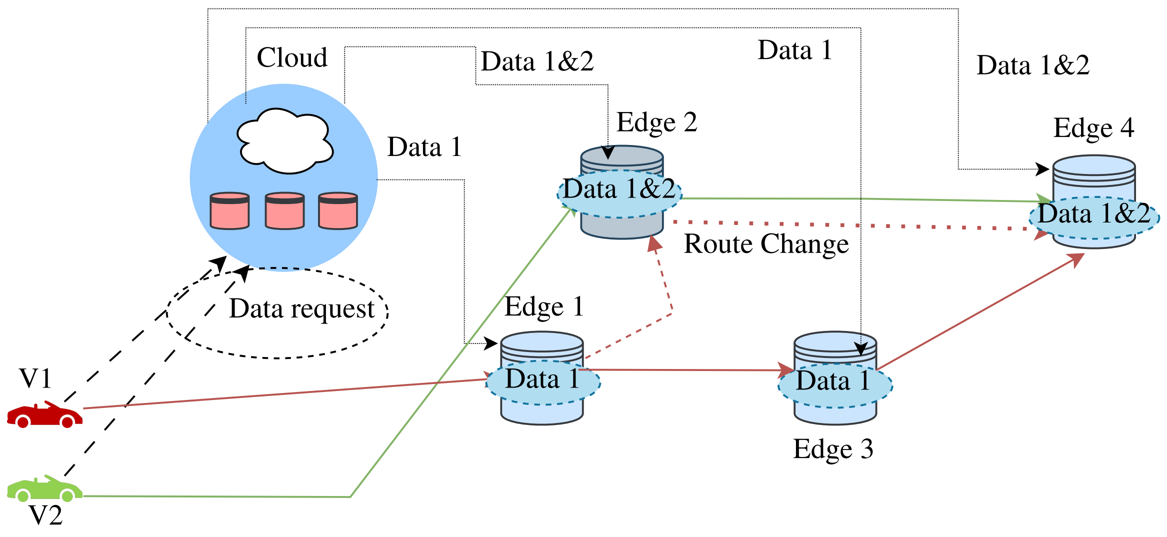
3.1 Introduction

The evolution of an automotive from a manually driven standalone platform to an intelligent platform that interacts with its environment has been primarily realized by novel advances in software and communication technologies. [20] These advances have enabled several applications in modern vehicles that require data transmission and computation offloading to nearby infrastructure or other vehicles. These applications are primarily used to improve the safety and driving experience of the driver. For instance, the Automotive Edge Computing Consortium (AECC) [1] recommends using High Definition Map application [2] to obtain accurate real-time information on the streets and traffic conditions that would help in better navigation. However, there is a large volume of data in high-definition (HD) maps, which need to be delivered efficiently to the vehicles requesting them. This problem is even more challenging if many vehicles in the environment are attempting to download high-definition maps for their trip.

One common approach to facilitate such high-volume data delivery is to send the data directly from the cloud to the vehicles. However, this can adversely affect the transmission bandwidth and latency due to duplicate transmissions of the same HD map data to nearby vehicles, thereby affecting the quality of the driving experience. Instead, the map data could be fetched from the cloud and stored in a few intermediate nodes (such as roadside units (RSUs) or edge server nodes) close to the connected vehicles. By prefetching the data to the nodes, the vehicles can receive it when they pass through its coverage region with improved transmission latency and reduced redundant data transmissions.



(a)



(b)

Figure 3.1: (a) Fixed Route (b) Dynamic Route Change Scenario

The main challenge for the realization of the aforementioned hierarchical architecture (consisting of the cloud, edge nodes and vehicles) is to develop an optimal data delivery framework. This framework allocates adequate resources (memory, bandwidth, etc.) on the intermediate edge nodes for every vehicle requesting data while optimizing some system performance parameters such as average data delivery time or bandwidth cost, etc. [27] Such a framework should consider the vehicle flow parameters to deliver data efficiently considering realistic vehicle densities and speed values. There have been few works that have considered data delivery mechanisms for connected vehicles via edges in various scenarios considering vehicle flow information [17]. However, these works perform well when the vehicle routes are fixed. To the best of our knowledge, there is no prior work that considers the resource allocation problem for data delivery to vehicles via edges supporting dynamic route changes taken by the drivers.

3.1.1 Motivation

In this work, we highlight the advantages of resource allocation for data delivery to connected vehicles via edges while considering route changes. The problem setting is depicted in Fig. 3.1(a). There are 2 vehicles (shown as $V1$ and $V2$), 4 edges (shown as $Edge$ 1, $Edge$ 2, $Edge$ 3 and $Edge$ 4) and the cloud in the VEC system. The data requested by the vehicles is distributed and delivered from the cloud to the edge nodes, as shown. The vehicles download the data from the edge nodes as they pass through the coverage areas of the edges. For example, vehicle $V1$ gets its data while it passes through the coverage region of edges $Edge$ 1, $Edge$ 3 and $Edge$ 4, respectively. A recent work [17] has considered vehicle flow model to optimally deliver data to the edge nodes for each vehicle $V1$ and $V2$ as shown in Fig. 3.1(a), when the route of the vehicles is fixed.

However, there is no guarantee that the vehicles always take a predetermined path. Due to various factors, such as congestion and road construction delays, the drivers can dynamically change their routes. This dynamic route change is shown using dotted arrows in Fig. 3.1(b), where vehicle $V1$ changes its route from $Edge$ 1 and goes to $Edge$ 2 instead of $Edge$ 3. If the framework does not handle these dynamic changes and perform resource allocations accordingly, it will not be possible to serve the data requirements of the vehicles. In this case, the offline optimization-based data delivery framework [17] will allocate resources for data delivery on $Edge$ 3 for vehicle $V1$, which will eventually be unavailable since $V1$ will not pass through $Edge$ 3. Therefore, we propose a time slot-based dynamic data delivery framework that considers the vehicle flow model and dynamic route changes to deliver data to the vehicles via edges.

In this work, our contributions are as follows:

1. *We propose a time slot based just in time dynamic data delivery framework. It consists of two optimisation stages that consider a vehicle flow model while accounting for route changes.*

2. *The proposed approach is compared with various existing optimal and heuristic approaches (using real-world data sets), thereby demonstrating its advantages..*

The chapter is further organized as follows.

The formulation of the models in the problem discussed in this paper is presented in Section 3.2. Subsequently, we present our proposed solution approach including the two-stage optimization in Section 3.3. The experimental setting and the results obtained are shown in Section 3.4.

3.2 Problem Formulation

In this section, we present our VEC model consisting of the cloud, edges, and vehicles. All the RSUs/edges will try to send the requested data to the vehicles when they enter the coverage region of those respective edges at some particular time slot/s. This data is sent to the vehicles from the cloud via the edges. In each time slot, data is allocated to several edges from the cloud, according to the vehicle requests

In this work, we are considering that any request by a vehicle is served across time slots through the edges only. This requested data is delivered as chunks from the different edges it passes on its route. The request for data by the vehicle will only contain the destination of the vehicle as metadata.

The cloud maintains a map of the entire network, can mark the possible paths between any two points, and gets information on the location of each vehicle at the start of every time slot. Thus, the cloud can preemptively deliver the data chunks to the relevant edges, through which a vehicle may pass at each time slot. We model our VEC system in terms of a vehicle model, an edge model, and a cloud network model.

Vehicle Model: Every vehicle can be described as a tuple α with its attributes including vehicle identifier, location data, route data (origin and destination), and data requested.
$\alpha = \langle i, location_{i,t}, start\_loc_i, dest_i, mem\_req_i \rangle$. Details of each variable are shown in Table 3.1.

Edge Model: Every RSU/edge will have a limited amount of resources and a limited coverage length. Each edge has the attributes edge identifier, location of the edge, coverage length of the edge, total memory of the edge, processing memory of the edge, memory utilized by the edge for data delivery at time slot t and bandwidth of the edge. Thus each edge can be described as a tuple $\sigma = \langle j,\ loc\_edge_j, l\_cov_j, mem\_edge_j, mem\_proc_j, mem\_util_{j,t},\ bw\_edge_j, \rangle$. Details of each variable are shown in Table 3.1.

Cloud Network Model: The cloud network maintains the status of the entire network in each time slot. This includes the current time slot, number of vehicles and edges in the network, the time slot period, bandwidth of the cloud, minimum number of edges across all possible paths from the edges to the destination for each vehicle, and the set of vehicles that have a probability to pass through or stay at a particular edge in each time slot, the length of that set of vehicles for each edge at each time slot and the leftover data to be sent to each vehicle at each time slot. The Cloud network is represented by the tuple $\omega = \langle t, N_{vehicles}, N_{edges}, \tau, bw\_cloud, min\_edge\_dest_{i,j}, Vec\_set_{j,t}, len\_Vec\_set_{j,t}, left\_mem_{i,t}\rangle$. Details of the variables are shown in Table 3.1.

Contrary to other works such as [17], for which the pre-defined route of each vehicle was known apriori, the current work focuses on the problem of handling dynamic route changes by the vehicles during their travel, which are not in the pre-defined route. Hence, at each time slot, the network only collects the current location of the vehicle and not the entire route that the vehicle is going to take.

Goal: Given the vehicle model α, the edge model σ and the cloud network model ω, the goal of this paper is to solve the time slot-based data allocation problem, so as to handle the possible dynamic route changes of vehicles and thus, try to successfully allocate data for delivery to all the vehicles that requested data.

Table 3.1: Model variables and their description

Variable Name	**Description**
$N_{vehicles}$	Number of Vehicles in the VEC system
i	Vehicle identifier
N_{edges}	Number of Edges in the VEC system
j	Edge/RSU identifier
t	Time slot identifier
$start\_loc_i$	Starting location of vehicle i
$location_{i,t}$	Current location of vehicle i at time slot t
$dest_i$	Destination of vehicle i
mem_req_i	Data requested by vehicle i
loc_edge_j	Location of edge j
l_cov_j	Coverage length of edge j
mem_edge_j	Memory of edge j
mem_proc_j	Processing memory of edge j. This is the memory reserved by the edge for its working and cannot be used for data delivery
$mem\_util_{j,t}$	The memory reserved by edge j for delivering data to the vehicles that are in its coverage region at time slot t
bw_edge_j	Bandwidth of edge j
τ	Time slot period/duration
bw_cloud	Bandwidth of the cloud to send data to the edges
$min\_edge\_dest_{i,j}$	Minimum number of edges that vehicle i has to pass through, to reach its destination $dest_i$ from edge j, when considering all possible paths/routes
$Vec\_set_{j,t}$	Set of vehicles which are nearby or at edge j in time slot t and thus have a possibility of reaching or staying at edge j in time slot $t + 1$ (We call it as Overlap set of edge j at time slot t)
$len\_Vec\_set_{j,t}$	Length of the set of vehicles which are nearby or at edge j in time slot t and thus have a possibility of reaching or staying at edge j in time slot $t + 1$
$left\_mem_{i,t}$	Data left to be sent to vehicle i at time slot t. It is the difference of requested data of vehicle i - sum(delivered data) to vehicle i, across the time slots

3.3 Proposed Solution

In this section, we present our two-stage optimization framework, which will be executed in every time slot. It consists of the optimization functions and the constraints that need to be satisfied so as to deliver data to the vehicles from the edges in the next time slot. Of the two stages, the first stage is to allocate data from the cloud to the edges according to the requests of the vehicles that are in proximity to the coverage region of those edges. The second stage is to allocate the data received by each edge to the various vehicles, which may pass through those particular edges in the next time slot. The set of vehicles that are near/at the coverage region of an edge and thus have a possibility of coming to that edge, or staying at that edge, in the next time slot will henceforth be called the overlap set of that edge, as mentioned in Table 3.1. The description of additional variables used here is given in Table 3.2.

Now, we present each stage of optimization with the objective function and constraints. Given the vehicle model α, edge model σ, and cloud network model ω, the optimization framework that will run in each time slot t is given below.

3.3.1 First Stage Optimization - Data to Edge from Cloud

This optimization runs once, every time slot, at the start of the time slot. The output of this optimization is the amount of data that is being sent to each edge by the cloud, denoted by $data\_edge_{j,t}$. The objective function is as shown below

$$minimize \sum_{j=1}^{N_{edges}} bw_{j,t}^{cost} + wr_{j,t}^{cost}$$

s.t.

$$data\_edge_{j,t} \geq 0 \quad \forall j = 1...N_{edges} \tag{3.1}$$

$$data\_edge_{j,t} \leq mem\_free_{j,t} \quad \forall j = 1...N_{edges} \tag{3.2}$$

$$data\_edge_{j,t} \leq \sum_{i \in Vec\_set_{j,t}} left\_mem_{i,t} \forall j = 1...N_{edges} \tag{3.3}$$

$$data\_edge_{j,t} \leq \sum_{i \in Vec\_set_{j,t}} D_{i,j,t}^{min} \quad \forall j = 1...N_{edges} \tag{3.4}$$

$$\sum_{j=1}^{N_{edges}} data\_edge_{j,t} \leq \tau \times bw\_cloud \tag{3.5}$$

Table 3.2: Solution variables and their description

Variable Name	**Description**
$adj\_loc\_edge_j$	The adjacent or directly connected locations of the edge j
$path_i$	The route of the vehicle i. It is a list with the location of the vehicle at each time slot
$density\_jam$	Maximum number of vehicles that can be accommodated at the coverage region of any edge
T_final	The last time slot till which the algorithm will run
β	The bandwidth cost factor
δ	The waste ratio cost factor
$mem\_free_{j,t}$	The memory of the edge j that is available for data delivery allocation at time slot t
$D^{min}_{i,j,t}$	The minimum number of bytes the vehicle i can receive from edge j at time slot t, considering the RSU's limited bandwidth and the density of vehicles in the overlap set of edge j at time slot t.
$bw^{util}_{j,t}$	Bandwidth utilized for sending $data\_edge_j$ to edge j from the cloud in time slot t
$bw^{cost}_{j,t}$	Bandwidth cost incurred for sending $data\_edge_j$ to edge j from the cloud in time slot t
$wr^{cost}_{j,t}$	Waste ratio cost incurred by sending $data\_edge_j$ to edge j from the cloud in time slot t
$usage^{cost}_t$	Sum of the bandwidth cost and waste ratio cost incurred by sending data to all the edges from the cloud at time slot t
$data\_edge_{j,t}$	The data that was allocated by the optimizer to be sent from the cloud to edge j at time slot t
$data\_vehicle_{i,j,t}$	The data that was allocated by the optimizer to be sent to the vehicle i from the edge j at time slot t

The objective function shown is the minimization of the total usage cost considering all the edges in the VEC system at time slot t. The total usage cost is given by:

$$usage_t^{cost} = \sum_{j=1}^{N_{edges}} bw_{j,t}^{cost} + wr_{j,t}^{cost} \tag{3.6}$$

We use a non-linear pricing model for bandwidth cost [14] given by :

$$bw_{j,t}^{util} = \frac{data\_edge_{j,t}}{\tau \times bw\_cloud} \tag{3.7}$$

$$bw_{j,t}^{cost} = \beta \times (1 + bw_{j,t}^{util})^2 \tag{3.8}$$

We introduce a balancing variable known as the waste ratio cost, to increase the amount of data that is being sent to the edges, thus ensuring maximum utilization or minimum wastage of resources. It is used to balance the effect of the bandwidth cost variable, which tries to limit the amount of data being sent to the edges, based on the limited bandwidth each edge has to send data to the vehicles that pass their coverage region. Thus, the waste ratio cost is given by :

$$wr_{j,t}^{cost} = \delta \times (1 - \frac{data\_edge_{j,t}}{mem\_free_{j,t}}) \tag{3.9}$$

where δ is the waste ratio cost factor. This factor is used to balance the weight of the bandwidth cost factor. δ is multiplied with the fraction of unused memory of that edge.
Therefore, the objective function has to balance the two costs and thus tries to reduce the bandwidth cost (reduce the amount of data sent) as well as the waste ratio cost (reduce the cost incurred for not utilizing the available resources to the maximum, thus increasing the amount of data sent).

Eq. (3.1) sets the lower bound of the data to be sent to each edge as zero. Eqs. (3.2)(3.3)(3.4) represent the three upper bound constraints for the data to be sent to each edge. Eq. (3.2) ensures that data sent to each edge by the cloud does not exceed the free memory available for data allocation at that edge at time slot t. This variable is defined as:

$$mem\_free_{j,t} = mem\_edge_j - mem\_proc_j - mem\_util_{j,t} \tag{3.10}$$

Eq. (3.3) ensures that data sent to each edge by the cloud does not exceed the sum of the remaining requested memory of all the vehicles present in the overlap set of the edge at time slot t. The remaining/leftover requested memory of vehicle i at time slot t is the amount of data left to be received from the VEC network by vehicle i at time slot t. Thus, it is the difference between the requested memory of the vehicle and the memory it received from all edges it passed through across the time slots until the current time slot t. It is given by:

$$left\_mem_{i,t} = mem\_req_i - \sum_{k=1}^{t-1} \sum_{j=1}^{N_{edges}} data\_vehicle_{i,j,k} \qquad \forall location_{i,k+1} = j \tag{3.11}$$

These two constraints ensure that no more resources than required are utilized. Eq. (3.4) ensures that the data received by the edge should not exceed the minimum bytes that every vehicle in the overlap set of that edge can receive under an equal bandwidth distribution while in the edge's coverage area. This is to satisfy the bandwidth schedulability constraint given in [17]. This constraint States that if the memory allocated for a vehicle is less than this value $D^{min}_{i,j,t}$, then all the allocated data for delivery can be successfully sent to the vehicles without fear of data not being sent due to insufficient bandwidth of the edge. This is because $D^{min}_{i,j,t}$ denotes the memory that will be allocated if we consider the equal distribution of bandwidth of the edge, bw_edge_j, across all vehicles in the overlap set $Vec\_set_{j,t}$. $D^{min}_{i,j,t}$ is defined as:

$$D^{min}_{i,j,t} = \frac{bw\_edge_j \times \tau}{len\_Vec\_set_{j,t}} \tag{3.12}$$

Eq. (3.5) is to ensure the time constraint, which is that the total data sent by the cloud to all edges should not exceed the maximum amount of data that the cloud can send with its limited bandwidth in the time period of one time slot (τ).

3.3.2 Second Stage Optimization - Data to Vehicles from Edge

This optimization runs for each edge at every time slot. The amount of data received by each edge, the output of the first stage optimization, is one of the inputs for this stage. The output of this optimization is the data allocation for every vehicle that is part of the overlap set of that edge denoted by $data\_vehicle_{i,j,t}$. The objective function is shown below:

$$minimize \sum_{i \in Vec\_set_{j,t}} \frac{(left\_mem_{i,t} - data\_vehicle_{i,j,t})}{(left\_mem_{i,t} \times min\_edge\_dest_{i,j})}$$

s.t.

$$data\_vehicle_{i,j,t} \geq 0 \quad \forall i \in Vec\_set_{j,t} \tag{3.13}$$

$$data\_vehicle_{i,j,t} \leq left\_mem_{i,t} \quad \forall i \in Vec\_set_{j,t} \tag{3.14}$$

$$data\_vehicle_{i,j,t} \leq D^{min}_{i,j,t} \quad \forall i \in Vec\_set_{j,t} \tag{3.15}$$

$$\sum_{i \in Vec\_set_{j,t}} data\_vehicle_{i,j,t} \leq data\_edge_{j,t} \tag{3.16}$$

$$\sum_{i \in Vec\_set_{j,t}} data\_vehicle_{i,j,t} \leq (\tau \times bw\_edge_j) \tag{3.17}$$

The objective function shown is to minimize the sum of leftover data across all vehicles in the overlap set while giving priority to vehicles that have a possible path to their destination with a lesser number of edges. For this, the cloud network maintains track of the number of edges in each path from an edge to the destination of the vehicle and selects the minimum of that, which is stored as the variable $min\_edge\_dest_{i,j}$. This is done to ensure that no matter what path a vehicle takes to its destination, it will receive data from the edges of its path while trying to allocate more data to vehicles with lesser edges remaining in its path at slot t.

Eq. (3.13) sets the lower bound of data to be sent to each vehicle as 0. Eqs. (3.14) and (3.15) represent the two upper bound constraints for the data to be sent to each vehicle by that edge j. Eq. (3.14) ensures that data sent to each vehicle by that edge does not exceed that vehicle's leftover requested memory, Eq. (3.11), at time slot t. Eq. (3.15) ensures that the data received by the vehicle does not exceed Eq. (3.12), the minimum number of bytes that it can receive at the coverage area of that edge. Eq. (3.16) ensures that the total data allocated to all the vehicles in the overlap set of that edge does not exceed the data that the edge received from the cloud. Eq. (3.17) is to ensure the time constraint, which is that the total data sent by the edge to all vehicles in the overlap set should not exceed the maximum amount of data that the edge can send with its limited bandwidth in the duration of one time slot (τ).

3.3.3 Algorithm to run the two-stage optimization framework

In Lines 1-2, we initialize the variables depicting memory utilized at edge $mem\_util_{j,t}$ and leftover requested memory $left\_mem_{i,t}$. Line 3 decides the current time slot. From Line 4, the whole algorithm runs in each time slot until T_final. For each time slot t, we extract the locations of the vehicles.

Lines 4 and 5 loop through each edge and vehicle respectively. The overlap set for each edge is created by checking if the location of the vehicle is near or at the edge and if the leftover memory requested is greater than zero. All such vehicles are added to the overlap set of that particular edge, and its length is stored. This is done for all edges in Lines 6-11.

Algorithm 1 Two-stage optimization framework Algorithm

```
Require: Dataset containing $N\_edges$, $N\_vehicles$, $loc\_edge_j$, $mem\_req_i$, $mem\_edge_j$, $mem\_proc_j$, $bw\_edge_j$, $l\_cov_j$, $adj\_loc\_edge_j$, $path_i$, $start\_loc_i$, $dest_i$, $min\_edge\_dest_{i,j}$, $density\_jam$, $bw\_cloud$, $\tau$, $T\_final$, $\beta$, $\delta$
1: Initialise : $mem\_util_{j,t} = 0\ \forall j = 1...N_{edges}$ and $left\_mem_{i,t} = mem\_req_i\ \forall\ i = 1...N_{vehicles}$
2: for $t = 1$ to $T\_final$ do
3:     for $j = 1$ to $N\_edges$ do
4:         for $i = 1$ to $N\_vehicles$ do
5:             if ((($location_{i,t} == loc\_edge_j$) or ($location_{i,t} \in adj\_loc\_edge_j$)) &
6:              $left\_mem_{i,t} > 0$) then
7:                 Add vehicle $i$ to overlap set of $j$ in time slot $t$
8:         Add overlap set of $j$ to $Vec\_set$
9:         Add length of overlap set of $j$ to $len\_Vec\_set$
10:        $mem\_free_{j,t} = mem\_edge_j - mem\_proc_j - mem\_util_{j,t}$
11:        Add $mem\_free_{j,t}$ to $mem\_free$
12:        Set $mem\_util_{j,t} = 0$
13:    Call $data\_to\_edge$ function
14:    for $j = 1$ to $N\_edges$ do
15:        if ($len\_Vec\_set_{j,t} > 0$) then
16:            Call $data\_to\_vehicle$ function
17:        for $i \in Vec\_set_{j,t}$ do
18:            if $path_{i,t+1} == loc\_edge_j$ & $t < T_{final}$ then
19:                $left\_mem_{i,t} -= data\_vehicle_{i,j,t}$
20:                $mem\_util_{j,t} += data\_vehicle_{i,j,t}$
21: Allocation for all time slots Finished
```

Line 12 calculates the available memory for data allocation for each edge, as $mem\_free_{j,t}$, by subtracting the processing memory of the edge and utilized memory of the edge from the total memory of the edge. This available memory is added to the global list in Line 13. Line 14 then sets utilized memory $mem\_util_{j,t}$ as zero for the next allocation.

With this, we close the for loop across the edges and then call the $data\_to\_edge$ function in Line 16, which is the first stage optimization, using the calculated values as inputs to get the allocated data for each edge from the cloud.

Then, for each edge, if the overlap set of that edge contains one or more vehicles, we call the $data\_to\_vehicle$ function (Line 19), which is the second stage optimization, with the calculated values as inputs so as to allocate data to vehicles in overlap set. If the vehicle passes through that edge in $t + 1$ slot, then it will receive that allocated data $data\_vehicle_{i,j,t}$

Thus, this algorithm is able to continuously deliver data to the vehicles based on their current location and handle the dynamic route changes a vehicle can make in its travel.

3.4 Experimental Setup and Results

In this section, we present the experimental setup for the optimization framework and compare its performance with various algorithmic approaches using a dataset generated over a region in New York.

The experiments have been conducted in Matlab 2022b using SDPT3 [35] as the optimization solver. The experiments are assessed using an Intel(R) Xeon(R) CPU E5-2640-based server, which has 20 cores, with each running at 2.6GHz. This device acts in place of the central cloud, where we assume the whole Algorithm 1 to be running in each time slot. The experimental evaluation compares the results obtained using our approach with three other approaches.

3.4.1 Dataset considered

3.4.1.1 Real Dataset Generation

For the real dataset, a bounding box area in the Manhattan region is considered, as shown in Fig. 3.2. OSMnx [6], a Python module for geospatial processing of OpenStreetMap data, is used to extract road intersections and the lanes connecting them along with their distances, as shown in Fig. 3.3 For each unique intersection, an incremental intersection-id number is assigned. A CSV file with row elements as start-id, end-id, and distance is populated, where each row signifies the intersections connected directly, followed by a separation distance. The RSUs are placed randomly at any of the intersections. The CSV gives all the paths that can be taken by a vehicle at each intersection. This CSV file is given as input to the dataset generator.

3.4.1.2 Dataset generator

The dataset generator generates the required input information for the optimizer as a text file that contains the vehicle paths, memory requested, destination, etc., followed by various edge parameters such as memory capacity and cloud network data transfer values. We consider requests from the vehicles only at the start of the scenario and not as dynamic request workloads. We set the waste ratio cost factor δ to 8 (as it yielded the best results in comparison to other values during the various experiments. Other values incur higher bandwidth cost or lesser resource utilization), the minimum distance between neighbouring intersections to 100 meters, the time slot duration τ as 10 seconds, the bandwidth of the cloud as 400 Mbps, the number of vehicles that change routes as 10% of $N_{vehicles}$ and the final time

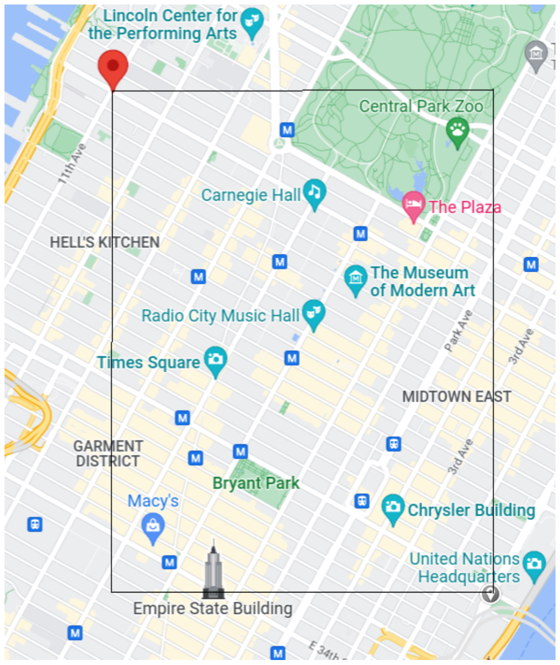


Figure 3.2: Manhattan region considered (Google Maps)

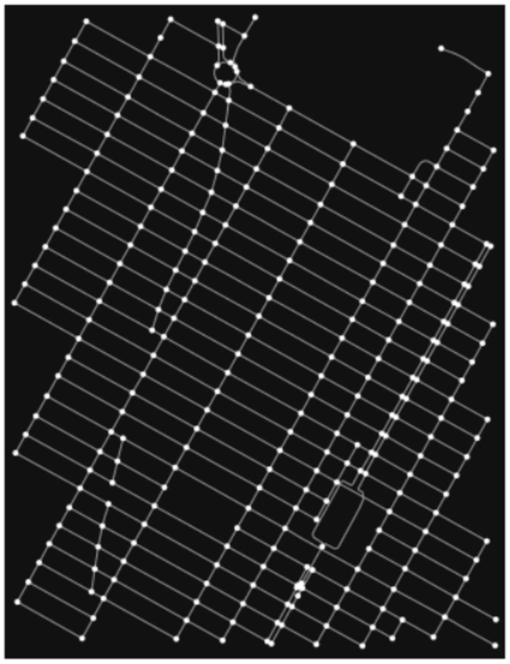


Figure 3.3: Intersections and road network made by OSMnx using OpenStreetMap data

slot (time slot till which the algorithms will run) as 120. Based on [17], the bandwidth cost factor β is set to 0.36 along with similar edge and vehicular parameters.

3.4.2 Comparison Algorithms

3.4.2.1 Greedy Allocation of data considering route changes of vehicles

With the routes unknown, a natural approach to deliver data, would be to utilize the maximum resources of the edges near the vehicle in every time slot, such that the vehicle will receive the maximum data when they pass through the edges. This online greedy approach (**greedy-dyn-route**) can therefore handle route changes and service more vehicles at the expense of increased resource utilization cost. It is based on a similar concept to the online greedy approach in [38].

3.4.2.2 Optimal allocation for reducing bandwidth cost, which does not consider route changes of vehicles

This is an offline global optimization algorithm where minimal bandwidth cost is incurred for data delivery to vehicles(**opt-fix-route**). Here, the pre-defined route information of the vehicles is required apriori. On changing their routes, the vehicles will not pass through the edges where data was allocated for them by the algorithm (which only runs once) and hence will not be serviced. Therefore, it cannot handle dynamic route changes of vehicles. It is similar to the optimal overlap scenario used in [17], known as Base Overlap (BO), since the overlap set of vehicles at an edge is considered here too.

3.4.2.3 Ideal allocation for reducing bandwidth cost, which assumes to have the final routes of the vehicles beforehand

Optimal bandwidth allocation with minimal bandwidth cost (**clair-fix-route**) is an offline algorithm where the entire route of the vehicle, including future route changes, is known apriori. Thus, the allocation will be done to the correct edges through which the vehicle will pass. This is based on the overlap computation used in [17] with the actual route of the vehicle as input instead of a pre-defined route.

3.4.2.4 Online heuristic of the two-stage optimization handling dynamic route changes of vehicles

This refers to the proposed two-stage optimization framework (**prop-dyn-route**). This method adapts a time-slot-based approach and needs the vehicle's current location at each time slot. Data is allocated to an edge when there are one or more vehicles nearby, and data delivery happens only if the vehicle passes through the coverage region of that edge, thereby handling route changes, unlike the offline approaches, which need the entire route apriori. The allocation happens in time slot t, and delivery happens in $t + 1$. Thus, the whole two-stage optimization runs within each time slot period.

3.4.3 Experimental Results - Real Dataset

The experiments on the real dataset were done by taking the CSV file generated using the map Fig. 3.2, as mentioned earlier and by taking (60,100), (70,150), and (80,200) as the $(N\_edges, N\_vehicles)$ pair.

The parameters used for comparing the algorithms are:

3.4.3.1 Number of vehicles serviced

As seen in Fig. 3.4, which showcases prop-dyn-route and greedy-dyn-route to overlap with clair-fix-route, prop-dyn-route is able to service as many vehicles as clair-fix-route (Ideal case with actual routes known apriori) and greedy-dyn-route (sends maximum amount of data at all time slots), which are the two approaches that will service the most vehicles. It does considerably better, around 5-10$\%$, than opt-fix-route (which is not able to service the vehicles that changed their routes).

3.4.3.2 Time taken to service the last vehicle

As seen in Fig. 3.5, prop-dyn-route is able to service the vehicles at almost the same time taken by greedy-dyn-route, which is the fastest algorithm at servicing data since it sends the most data at all time slots, at an average of 17$\%$ slower than it. But prop-dyn-route is significantly faster (avg of 2.3 times) at servicing all the vehicles, when compared to the optimal approaches, which tends to allocate the data to more edges, thus making the vehicles travel more to get the complete requested data.

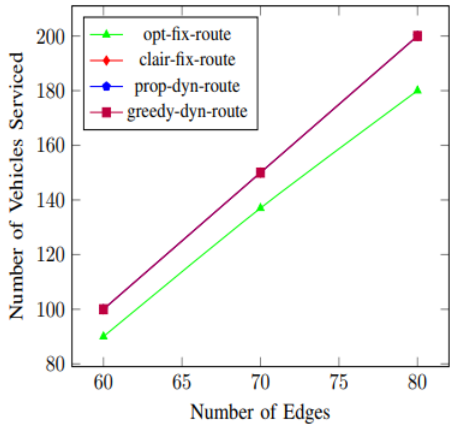


Figure 3.4: Comparison of total number of vehicles serviced successfully for a given number of edges by the opt-fix-route, clair-fix-route, prop-dyn-route greedy-dyn-route approaches

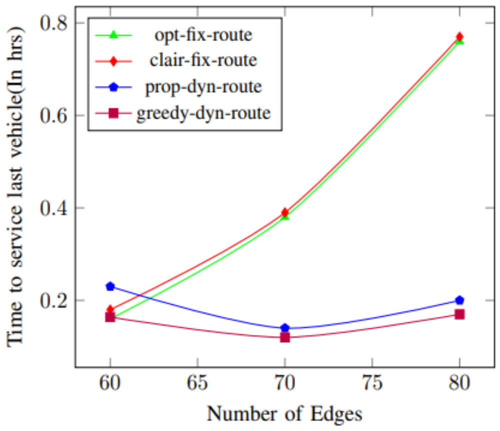


Figure 3.5: Comparison of time taken to service the last vehicle by the opt-fix-route, clair-fix-route, prop-dyn-route greedy-dyn-route approaches

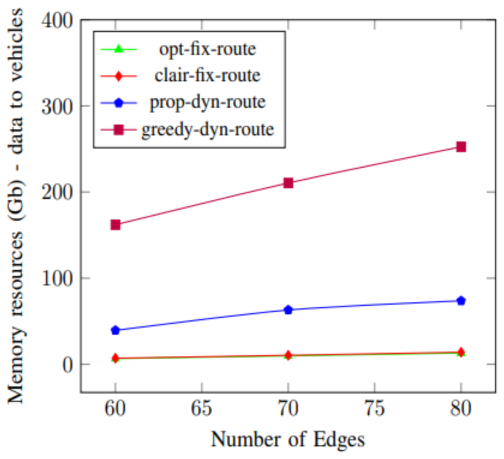


Figure 3.6: Comparison of total Memory resources of the edges used to deliver data to the vehicles by the opt-fix-route, clair-fix-route, prop-dyn-route greedy-dyn-route approaches

3.4.3.3 Total memory resources used at the edges

As seen in Fig. 3.6, prop-dyn-route requires on average 6 times more memory than the optimal approaches. This is due to the fail-safe memory allocations that happen at various edges in the expectation that nearby vehicles might pass through them, thus requiring more memory to handle the possible route changes of vehicles. But this memory consumption is significantly lesser when compared to greedy-dyn route(70% lesser) due to it wasting more memory resources to allocate more data faster.

3.5 Concluding Remarks

In this work we proposed a dynamic data delivery allocation framework for delivering data to connected vehicles via edge nodes, that can handle real time route changes of the vehicles and hence does not require the routes of the vehicles beforehand. We compare our proposed work with an optimal approach that cannot handle dynamic route changes, an idealistic optimal approach which knows the final changed routes apriori and allocates accordingly, and a greedy approach . We generated a route dataset over a selected region of Manhattan, using OSMnx, and used this dataset to chart the possible routes of all the vehicles. We used this dataset along with vehicle flow models similar to the ones used in [17] as the real dataset for testing. The proposed approach was able to almost service as many vehicles as the idealistic optimal approach, including those which had changed their routes unlike the optimal approach that cannot handle route changes. It was able to service the vehicles faster than the optimal approaches and utilises lesser memory than the greedy approach. In future, unlike the current assumption that the algorithm 1 is completely running on the central cloud, we can split the algorithm into two parts where the first part of allocating data to edge, is executed at the cloud, and the second part of allocation of data to each vehicle by an edge is handled by the edges themselves individually. But this would have to solve the synchronization challenges between cloud and edge.

In the experiments we have only considered an urban scenario. In a highway scenario, the number of intersections will be lesser and thus the possibility of route changes is lesser. Thus, in a highway scenario it might be more efficient, in terms of resource utilization, to use the optimal approach for minimizing bandwidth cost than the proposed dynamic approach.

Chapter 4

Global Earliest Deadline First Algorithm Based Scheduler for Computation Offloading in Vehicular Edge Computing

4.1 Introduction

The Connected Vehicles ecosystem is seeing rapid growth with the advancements in software and communication technologies. Many more consumer vehicles are now being equipped with means of connectivity, thus spreading the Internet of Vehicles (IoV) domain across all people. These innovations lead to better safety measures, for driver, passengers and pedestrians alike, more user assistance capabilities such as the Heads-up-display (HUD), augmented reality features, terrain and local environment analysis, etc. It also enriches passenger experience by allowing more entertainment options with better connectivity. But many of these features are computationally heavy, and thus might not be utilised to their full potential, with the limited in-vehicle computation capacity acting as a constraint. One of the more commonly used solutions for this problem is Computation Offloading, which is the technique of 'offloading' or executing a computation task outside of the source, in this case the vehicles. Computation Offloading is now possible with the advances in V2I and V2V technologies, which are based on Direct Short Range Communication (DSRC). The offloaded task can be executed in any nearby available computation device with enough resources. Some examples of possible offloading destinations include Road Side Units(RSUs or edge nodes), nearby vehicles, passenger devices etc. But each of these will have their own advantages and constraints.

Considering the case of offloading computation tasks to edge nodes, it is straightforward in the case of a single vehicle offloading a task to a nearby edge node. But the problem becomes non-trivial when it comes to a multi vehicle multi edge node system. We will have to consider many factors such as number of vehicles concurrent in the coverage region of an edge node at a time, the time each vehicle stays in the coverage region, the execution and data transfer times of the task and results, and how to utilise all the resources efficiently so as to service the maximum number of vehicles while ensuring Quality of Service (QoS) requirements such as executing task within the task deadline etc. We also have to consider all the constraints across multiple edge nodes and effectively utilize most of them. Thus the problem of scheduling the tasks across all the edge nodes in the system with all these constraints is non trivial.

Motivation: There have been several works which look at the computation offloading problem, each focusing on a particular approach with specific constraints. But there are not many works which focus on implementing existing task scheduling algorithms of a multi core system on a global scale, equating each edge node to a core, while considering vehicular flow constraints and QoS constraints. We aim to provide a distributed systems scheduler based on the popular Earliest Deadline First algorithm on a global scale, which schedules the tasks in a global queue and assigns each of them to a particular edge node considering the various constraints of vehicle flow and time constraints. We also compare our proposed approach with two partitioning based approaches and an optimal approach based in bandwidth utilized using a real world dataset and tabulate the results in terms of number of vehicles serviced and run time of the heuristic.

The chapter is further organized as follows.

In Section 4.2 we present the formulation of the models of the problem discussed in this paper. Subsequently, we present our proposed solution approach in Section 4.3. In Section 4.4 we present a brief on the other approaches used for computation offloading in VEC ecosystem, which we use for comparison with our proposed approach. The experimental setting and the results obtained are shown in Section 4.5

4.2 Problem Formulation

In this section we present our 3 layer VEC model consisting of the cloud layer, edge node/RSU layer and the vehicle layer. The edges will try to handle the service requests of the vehicles when they enter the coverage region of those respective edges at some particular time slot. The decision of which edge will handle the service request of which all vehicles in each time slot is taken by the global scheduler in the cloud.

In this work we are considering that any request by a vehicle is served by only one edge. This computational offloading takes place at one of the edges in the route of the vehicle. The service request sent to the cloud will only contain the metadata of the task such as the requesting vehicle's id and the deadline of the task, execution time of the task etc.

The cloud maintains a map of the entire network, and thus knows all the edges in the route of the vehicle as well as the nearby edges to the location of the vehicle in each time slot. Thus the cloud can check if all the conditions necessary for successful computational offloading of the task of the vehicle is satisfied at the edge, that is in the route of the vehicle and nearby its current location, and if so then allocate that task to that particular edge in that time slot. We model our VEC system in terms of of vehicle model, edge model and cloud network model.

Vehicle Model: Every vehicle can be described as a tuple $\alpha = \langle i, d_i, r_i, p_i, exec\_time_i, deadline_i \rangle$. Here i stands for the vehicle identifier. The variable d_i stands for the input data of the task of vehicle i to be sent to the edge for processing and r_i is the result data of the task of vehicle i, sent to the vehicle from the edge that processed the task. p_i stands for the amount of resources (in terms of VMs) are required for the execution of the task. $exec\_time_i$ is the execution time required to process the task of vehicle i. $deadline_i$ is the deadline for the task of vehicle i to be executed by the VEC network. Details of each variable are shown in Table 4.1.

Edge Model: Every edge in the network can be modelled as a tuple $\sigma = \langle j, l\_cov_j, mem\_edge_j, P_j, bw\_edge_j, mem\_proc_j, \rangle$. Here j stands for the edge identifier. l_cov_j is the coverage length of the edge j. mem_edge_j is the memory capacity of the edge j. P_j is the processing capacity(in terms of VMs) of the edge j. bw_edge_j is the network bandwidth provided by the edge j. Some of the resources of an edge may be already occupied. We denote mem_proc_j as the occupied memory of the edge, reserved for the working of the edge. Therefore the available memory for task offloading is $mem\_edge_j - mem\_proc_j$. These resources get utilized when a vehicle requests for service delivery. These variables are shown in Table 4.1.

Cloud Network Model: The cloud network maintains the status of the entire network in each time slot. It maintains the current time slot, number of vehicles and edges in the network, the time slot period. The Cloud network is represented by the tuple $\omega = \langle t, N_{vehicles}, N_{edges}, \tau \rangle$. Details of the variables are shown in Table 4.1.

Table 4.1: Model variables and their description

Variable Name	**Description**
$N_{vehicles}$	Number of Vehicles in the VEC system
i	Vehicle identifier
N_{edges}	Number of Edges in the VEC system
j	Edge/RSU identifier
t	Time slot identifier
l_cov_j	Coverage length of edge j
mem_edge_j	Memory of edge j
mem_proc_j	Processing memory of edge j. This is the memory reserved by the edge for its working and cannot be used for data delivery
bw_edge_j	Bandwidth of edge j
τ	Time slot period/duration
d_i	Data sent to the VEC network from the vehicle i as input data for the task offloading
r_i	Data sent to vehicle i as the result, from the edge that executed its task
p_i	Amount of resources in terms of VMs, required for the processing of the task of vehicle i
P_j	Total number of processing units, in terms of VMs, that the edge has
$exec\_time_i$	Time taken by any edge to process the task of vehicle i
$deadline_i$	The global deadline for the completion of the task of vehicle i. Used as the priority deciding factor in the scheduler.

4.3 GEDF based Scheduler for Service Delivery

In this section we present our proposed approach, which is the Global Earliest Deadline First algorithm based scheduler for service delivery. The scheduler is executed in the form of a priority queue which gets updated every time slot. The framework consists of the heuristic algorithm and the constraints to be satisfied for the successful allocation of the task to a particular edge. In each time slot the scheduler in the cloud will first choose the tasks with the highest priority in the priority queue, and then check the if all the constraints are satisfied for each task by a particular edge. If the conditions are satisfied then the task is allocated to that particular edge for computational offloading, if not the task is kept in a separate queue to be added back into the priority queue scheduler to check for allocation in the next time slot.

Given the vehicle model α, edge model σ, and cloud network model ω, we now present the scheduling framework. First we present the constraints and then the scheduling algorithm.

4.3.1 Constraints

The constraints that have to be satisfied to allocate a task to an edge are as follows:

1. **Vehicle is near or at the coverage of an edge:** For each time slot, we can approximate the location of the vehicle i based on the arrival time at edge j, $arr\_t_{i,j}$, and departure time from edge j, $dept\_t_{i,j}$. The arrival time can be approximated by knowing the distance of edge from the starting point of the vehicle and the average speed of the vehicle on the road. Similarly the departure time can also be estimated given the distance and velocity information. Thus we assume the cloud to already have the arrival and departure times of vehicle i at all the edges it passes through as part of its route. Therefore coverage time, as given by shown by Eq 4.1, is the time taken for vehicle i to cover the the coverage length of edge j.

$$cov\_time_{i,j} = arr\_t_{i,j} - dept\_t_{i,j} \tag{4.1}$$

   Using the arrival time, $arr\_t_{i,j}$, and departure time, $dept\_t_{i,j}$, we can check if the vehicle will pass through the coverage region of a particular edge or not in the current time slot t using the following condition:

$$t \ni arr\_t_{i,j} <= (t+1) * \tau \quad and \quad dept\_t_{i,j} >= (t+1) * \tau \tag{4.2}$$

Eq 4.2 states that the current time slot t should be such that the arrival time of the vehicle i at edge j should be before the start of the next time slot $(t+1)$, as well as the departure time from the coverage region should be after the current time slot ends. This ensures that the vehicle considered is present in the coverage region of that particular edge j in the current time slot t, thus making it feasible to schedule the offloading of the task to this particular edge in this particular time slot.

2. **Considered edge has enough free resources to execute the task:** Each edge in the network has a limited number of resources that can be used for service delivery. Thus in each time slot, to schedule a task at an edge for offloading, we need to ensure that the edge has enough resources to execute the task, which is not currently occupied with another task.

$$p_i <= P_{j,t}^{free} \tag{4.3}$$

where

$$P_{j,t}^{free} = P_j - \sum_{i=1}^{N} p_i * serv_{i,j,t} \tag{4.4}$$

Here $serv_{i,j,t}$ is the binary variable which denotes whether the task of vehicle i has already been scheduled for execution at edge j in the current time slot t. By the taking the sum of the processing resource requirements all such task already scheduled, we get the number of occupied resources. So the available number of resources of edge j in time slot t for service delivery, $P_{j,t}^{free}$, is given by the total capacity of the edge, P_j, minus the number of occupied resources. Thus the second constraint, Eq 4.3, is that the processing units requirement of the task of vehicle i should be less than or equal to the number of available units at the edge, in that time slot.

For an edge to process a service request, some amount of data transmission is also required between the edge and the vehicle. Namely the transfer of the input data d_i from the vehicle to the edge and the transfer of the result data r_i from the edge to the vehicle. Thus, there must be enough free/available memory resources, mem_free_j for data transmission at the edge for the service delivery of a particular task. This is shown by Eq 4.5.

$$d_i + r_i <= mem\_free_{j,t} \tag{4.5}$$

where

$$mem\_free_{j,t} = mem\_edge_j - mem\_proc_j - \sum_{i=1}^{N} (d_i + r_i) * serv_{i,j,t} \tag{4.6}$$

Eq 4.6 states that the available memory resources at edge j in time slot t is the memory capacity of the edge, mem_edge_j, minus the processing memory, mem_proc_j, and the sum of the memory resources used for the data transmission of the already scheduled tasks' data.

3. **Task can be executed and results sent back to the requesting vehicle from the edge within the coverage time:** Due to the limited amount of time the vehicle is present in the coverage region, we need to ensure that the total time taken for the service request delivery is within the coverage time of the vehicle. Total time for the service delivery is the sum of the data transmission time and the execution time. Since coverage time is calculated using the arrival and departure times, $arr\_t_{i,j}$ and $dept\_t_{i,j}$, and execution time is already given as input, we can get the maximum available time for data transmission as the difference between the two as shown in Eq 4.7

   $$data\_trans\_t_{i,j,t} <= cov\_time_{i,j} - exec\_time_i \tag{4.7}$$

   where

   $$data\_trans\_t_{i,j,t} = \frac{(d_i + r_i)}{alloc\_bw_{i,j,t}} \tag{4.8}$$

   As seen in Eq 4.8 data transmission time is dependent on the amount of input data d_i and result data r_i as well as the amount of bandwidth of the edge that is allocated, $alloc\_bw_{i,j,t}$. Since the bandwidth allocated may vary, we consider the pessimistic scenario of the coverage region being filled with vehicles, also called the jam density situation and the bandwidth being split across all the vehicles. Thus to ensure data transmission does not cross the maximum available time, given the limited bandwidth we have the bandwidth schedulability constraint given by Eq 4.9:

   $$d_i + r_i <= D\_min\_serv_{i,j} \tag{4.9}$$

   where

   $$D\_min\_serv_{i,j} = \frac{bw\_edge_j * (cov\_time_{i,j} - exec\_time_i)}{density\_jam_j * l\_cov_j} \tag{4.10}$$

   Eq 4.10 finds the maximum amount of data that can be transmitted between the vehicle i and edge j in the coverage region of j, given the limited amount of time and bandwidth. By ensuring that the sum of d_i and r_i is less than this maximum, we ensure that the data can be transmitted, and the task executed at edge j within the coverage time itself. Eq 4.10 is derived from the bandwidth schedulability constraint given in [17].

These constraints have to be checked for each task in the priority queue at every time slot. Further details on the variables used to form the constraints are given in Table 4.2

Table 4.2: Constraint variables and their description

Variable Name	**Description**
$arr\_t_{i,j}$	Arrival time of vehicle i at edge j
$dept\_t_{i,j}$	Departure time of vehicle i from edge j
$cov\_time_{i,j}$	Time taken for vehicle i to cover the coverage length of edge j
$serv_{i,j,t}$	Binary variable (1,0) which denotes whether the task of vehicle i is being executed by edge j at time slot t
$P_{j,t}^{free}$	Amount of free/available processing units (VMs) of edge j at time slot t
mem_free_j	Amount of available memory of edge j at time slot t, that can be utilized for data transmission during task offloading
$data\_trans\_t_{i,j,t}$	Total time taken for data transmission between vehicle i and edge j at time slot t. This includes sending of input data to the edge and receiving the result from the edge.
$alloc\_bw_{i,j,t}$	Allocated bandwidth of edge j for the data transmission between vehicle i and edge j at time slot t
$density\_jam_j$	Max density scenario, where $density_j$ is defined as the amount of vehicles present in each unit length of the coverage region of edge j
$D\_min\_serv_{i,j}$	The minimum amount of data any vehicle i can receive in the coverage region of edge j. This is due to the consideration of the pessimistic scenario of jam density in its calculation. It is used to satisfy the bandwidth schedulability constraint of [17]

4.3.2 Scheduler

Other optimal scheduling approaches such as the one presented in [13] might offer better solutions in terms of the number of vehicles serviced. But these optimal approaches tend to take a long time to find the most optimal allocation. Thus we try to address this issue by proposing a faster heuristic scheduler approach. The global scheduler in the cloud is in the form of a priority queue, with the deadline of the task as the priority deciding factor. This queue is maintained by the cloud. The vehicles send their service requests to the cloud. The requests include metadata such as the vehicle identifier, route of the vehicle, task deadline etc. The task request is then added to the priority queue. This queue is reordered at the start of every time slot. The following Algorithm 2 depicts the steps the scheduler follows to schedule the tasks to the appropriate edges.

First we load all the input data from the dataset, including all model variables. We then create a priority queue and add all the tasks with their respective deadlines into the queue and order it as shown in Lines 1-2. In Line 3 we initialize the amount of free resources for each edge considering no task has been scheduled yet at any edge as per Eq 4.4 and Eq 4.6. We also initialize two null sets to keep track of the tasks currently running at any of the edges, and the ones that need to be added back to the priority queue for possible scheduling in the next time slot.
Line 4 depicts each time slot starting from the first going up to T_Final, to show that the scheduler runs in each and every time slot. Line 5 shows that we run the scheduler until the priority queue is empty using the while loop to iterate through all the tasks in the queue. We extract the element with the highest priority in the queue, to check if it satisfies the conditions for scheduling. Line 7 denotes the iteration over edges, since we need to check if any of the edges satisfy the conditions for scheduling that particular task in that time slot.
Lines 8,9 and 11 are the constraints to be satisfied for successful scheduling of the task of vehicle i at edge j. These constraints are shown in Eq 4.2,Eq 4.3, Eq 4.5 and Eq 4.9. Line 10 is for calculating $cov\_time_{i,j}$ and $D\_min\_serv_{i,j}$ as per Eq 4.1 and Eq 4.10. If all the three constraints are satisfied, then we allocate that task of vehicle i to that edge j at time slot t. Therefore we reduce the amount of required resources for task of vehicle i, from the available resources of edge j, and assign $serv_{i,j}$ the value 1 to indicate that edge j has been allocated to handle the execution of the task of vehicle i. Then we add the task of vehicle i to the $running\_set$ to indicate it is currently being executed, and then break the for loop, since this particular task has already been assigned to one edge and hence we do not need

Algorithm 2 GEDF based scheduler Algorithm

Require: Dataset containing N_edges, $N\_vehicles$, τ, mem_edge_j, mem_proc_j, P_j, bw_edge_j, l_cov_j, $density\_jam_j$, T_final, $arr\_t_{i,j}$, $dept\_t_{i,j}$, d_i, r_i, p_i, $exec\_time_i$, $deadline_i$

```
 1: Create empty priority queue - PQ
 2: Enqueue : Priority queue, PQ, with all the tasks and their deadlines, $deadline_i$
 3: Initialize : $P_{j,t}^{free}$ as $P_j$, $mem\_free_{j,t}$ as $(mem\_edge_j - mem\_proc_j)$ for all edges, $serv_{i,j}$ as zero
    for all $i, j$ pairs, and two null sets called $running\_set$ and $reassign\_set$
 4: for $t = 1$ to $T\_final$ do
 5:     while PQ.size>=0 do
 6:         Pop the highest priority element from PQ and store its value (Vehicle identifier) as $i$
 7:         for $j = 1$ to $N\_edges$ do
 8:             if $arr\_t_{i,j} <= (t+1) * \tau$ and $dept\_t_{i,j} >= (t+1) * \tau$ then
 9:                 if $p_i <= P_{j,t}^{free}$ and $(d_i + r_i) <= mem\_free_{j,t}$ then
10:                     Calculate $cov\_time_{i,j}$ and $D\_min\_serv_{i,j}$
11:                     if $(d_i + r_i) <= D\_min\_serv_{i,j}$ then
12:                         $P_{j,t}^{free} -= p_i$, $mem\_free_{j,t} -= (d_i + r_i)$ and $serv_{i,j} = 1$
13:                         Add $i$ to $running\_set$ and then break
14:         if $serv_{i,j} == 0 \ \forall \ j = 1...N_{edges}$ then
15:             Add task to $reassign\_set$
16:     for tasks in $reassign\_set$ do
17:         Enqueue : PQ with that task and its respective deadline
18:     for task in $running\_set$ do
19:         if $dept\_t_{i,j} <= (t * \tau)$ then
20:             $P_{j,t}^{free} += p_i$, $mem\_free_{j,t} += (d_i + r_i)$,
21:             Remove $i$ from $running\_set$
22:     if PQ.size $<= 0$ then
23:         break
24: Scheduling Finished
```

to check whether the allocation constraints are satisfied for any other edge.
Line 14 checks if the task of vehicle i has been assigned to any edge. If not then in Line 15, it is added to the $reassign\_set$ to be added back to PQ in the next time slot so as to be scheduled later.
Line 16, is after all the tasks have been traversed and popped from PQ. Line 16 and 17 we enqueue PQ again with all the tasks which yet to be allocated any edge. Line 18 traverses all the tasks which are currently running. In line 19, we check if the vehicle exits the coverage region of the allocated edge. If so, we free up the resources allocated for that particular task since we assume the task has already finished execution since the vehicle is leaving the coverage region. Then in line 21 we remove that task from the $running\_set$. Line 22 and 23 is to break the time slot loop, if all the tasks have already been scheduled. With that we finish the execution of the algorithm.

4.4 Comparison Approaches

4.4.1 Fully Partitioned Approach

The fully partitioned approach utilizes a bin packing heuristic called First Fit Decreasing Utilization (FFDU) in conjunction with temporal deadline, vehicular flow constraints and edge bandwidth constraints. Given the tasks of the vehicles in a non decreasing order of execution times, we assign tasks to the edge nodes in this order in each time frame. Assuming $i-1$ tasks have been scheduled, we consider the the i^{th} task to be scheduled at edge j if certain conditions are satisfied. The conditions are:

1. After the execution of the tasks already assigned to edge j, there is still enough time for executing task i as well as transfer its data before the vehicle leaves the coverage region of edge j

2. Execution of tasks already assigned to edge j and the execution of task i can be done within the current time frame

3. Edge j has enough resources (memory, bandwidth etc) to executed the tasks already assigned to it as well as execute task i

If all these conditions are satisfied, the task i is scheduled to be executed at edge j in the current time frame. Fully partitioned approach ensures that each task is scheduled to be executed at only one edge.

4.4.2 Semi Partitioned Approach

The semi partitioned approach extends from the fully partitioned approach. In this approach we relax the restrictions of fully partitioned approach, by allowing some of the tasks to be split and scheduled across multiple edge nodes instead of being completely executed on one edge node. We first run fully partitioned scheduling and assign all the tasks that can be scheduled completely at the edge nodes. Then for the remaining tasks that were not scheduled, we arrange them in a non decreasing order based on the time the leave the coverage of the last edge node in their routes. Then in each time frame, we calculate the available time for execution of the task i at the edge j which it is currently passing in the current time frame. Then we execute part of the task according to this available time. And then we check the next edge node it passes through, in the same time frame and calculate its respective available time. Another part of the task is executed at this edge node according to the available time, after reducing the required time to transfer the intermediary data from the previous edge node to the current one (referred to as migration policy). This keeps repeating until the task is completely executed. But the constraint is that it should be fully executed withing one time frame. If it cannot be executed across multiple edge nodes, in one time frame, then the task is not scheduled. This approach increases task execution efficiency compared to the fully partitioned approach.

4.4.3 Optimal Approach

The optimal approach tries to optimize the global bandwidth incurred to execute the tasks and transfer the required data between the edges and the vehicles. This approach is a modified version of the optimal approach proposed in [13]. Unlike the pessimistic scenario of all the vehicles passing through an edge, covering it at the same time, we consider the vehicles to pass at different times, and check the constraints for each overlap set as defined in [17]. This optimal approach also considers the vehicle flow constraints, resource constraints and timing constraints.

4.5 Experimental Setup and Results

In this section, we demonstrate the experimental setup followed by a comparison of the proposed approach with two other scheduling approaches, namely Full Partitioning scheduling and Semi Partitioning scheduling and an optimal approach which was proposed in [17].

The experiments for the optimal approach have been conducted in Matlab 2022b using Gurobi as the optimization solver. While the heuristic scheduling approaches were executed using Python. The experiments are assessed using an Intel(R) Xeon(R) CPU E5-2640-based server, which has 20 cores, with each running at 2.6GHz. The experimental evaluation compares the results obtained using our approach with the three other approaches.

We conducted our experiments on vehicular traces from a data set of traffic traces from Luxembourg City [9][16] that was simulated using SUMO. From this dataset, we extracted the vehicle routes, distances travelled by the vehicles, and the count of vehicles present in each road segment. The parameters such as the execution time, coverage distance, bandwidth etc, were generated as described below due to the unavailability of these parameters in the real data set. We defined the parameter ranges to values that align with real-world scenarios commonly observed.

The following are the parameters and the setup used to run the experiments:

- We used different values for the number of edges, N_{edges} ranging from 30 up to 70.
- The routes of 400 vehicles were generated randomly such that the vehicles pass through different edges. We use different combinations of N_{edges}:$N_{vehicles}$ pairs for comparison
- We place the edges at random locations in the Luxembourg map region that the selected vehicles pass through
- The vehicle density at jam ($density\_jam_j$) was set as 35 for all edges.
- Coverage distance (l_cov_j) of the edges was randomly generated using uniform distribution between 0.6 miles to 1.6 miles.
- The bandwidth of the edges (bw_edge_j) was randomly generated between 8 Mbps to 15 Mbps.
- Memory capacity (mem_edge_j) of the edges was randomly generated between 400 Mbits to 500 Mbits using uniform distribution and the occupied memory (mem_proc_j) was randomly generated between 0 Mbits to 150 Mbits.
- The computation capacity of the edges (P_j in number of VMs) was set as 1, to make it such that each edge can only execute one task at a time. This to ensure similar conditions for comparison with the other scheduling approaches, which do not consider parallel execution of tasks at an edge.

- The vehicle's task's processing requirement (p_i) was also set as 1 for similar reasons.
- The data that the vehicle sends (d_i) to the edge and the data that it receives (r_i) from the edge were randomly generated between 1 Mbits to 15 Mbits.
- The execution time of task ($exec\_time_i$) was randomly generated between 1 - 10 sec.
- Deadline of the task, $deadline_i$ was randomly generated between 500 sec and 1500 sec
- Arrival time, $arr\_t_{i,j}$, and departure time, $dept\_t_{i,j}$, at each edge for each vehicle were estimated using the distances from the starting point to that edge and assuming an average velocity of 60mph
- Time slot duration, τ, is set as 10 sec.

We have compared the number of vehicles serviced and the run time complexity of the the proposed approach vs the two partitioning approaches and the optimal approach. The results are shown in Table 4.3. The details are as follows:

4.5.1 Run-time complexity

The proposed approach significantly reduces the amount of time required to schedule the tasks at the edges compared to the optimal approach, which takes a long time to find the optimal solution. The time taken is reduced by as much as 250 times and is closer to the times taken by the partitioning approaches. The proposed approach takes nearly 2-3 times the run time to schedule the tasks as compared to the partitioning approaches. It can also be seen that as the complexity of the network increases, in terms of number of vehicles and edges, the run time to schedule the tasks also increases.

4.5.2 Max number of tasks scheduled

For the tasks requested by the vehicles for offloading, more tasks can be scheduled with the increase in the number of available edges as seen in the cases with 300 vehicles requesting for their tasks to be offloaded with 40 and 50 edges in the network. In the 50 edges case, the number of tasks that were scheduled to be offloaded increased for all approaches. The number of tasks scheduled by the proposed approach is more than both of the partitioning approaches but not as much as the optimal approach which is able to schedule all the vehicles.

Table 4.3: Comparison of Proposed Partitioned Algorithms with Optimal Approach [17]

	M	N	Partitioned	Semi-Partitioned	Optimal	GEDF
Number of tasks scheduled	70	400	370	377	400	390
	50	370	332	342	370	359
		300	286	286	300	293
	40	300	250	255	300	275
		250	215	216	250	233
Run Time (in sec)	70	400	2.495	2.507	2791	5.761
	50	370	1.541	1.554	1197.7	4.371
		300	0.988	0.993	751.93	2.907
	40	300	1.003	1.018	602.6	2.879
		250	0.707	0.716	409.31	1.972

4.6 Concluding Remarks

In this work, we proposed a heuristic solution for scheduling offloaded computation tasks at the edge nodes by using a popular task scheduling algorithm called Global Earliest Deadline First algorithm (GEDF) and modifying it to suit the VEC scenario considering the distributed edge nodes, vehicle flow constraints and timing constraints. We considered other scheduling approaches such as Fully Partitioned Scheduling and Semi-Partitioned scheduling at the edge nodes, presented in a co-authored paper, and an optimal approach for comparison. We tested the performance of the proposed approach against the other approaches using a real world dataset, the Luxembourg data scenario [10]. The proposed approach was able to service more vehicles than the partitioning approaches at the cost of slightly more run time, and able to schedule the tasks of the vehicles much faster than the optimal approach, at the cost of slight dip in performance, in terms of the number of tasks scheduled.

Chapter 5

Conclusion and Future Works

In this work, we proposed a route-agnostic dynamic data delivery framework for vehicles with route changes. The proposed approach is able to serve considerably more vehicles than other offline algorithmic approaches. It utilises lesser memory usage than other approaches which handle route changes,such as the greedy approach. It demonstrated faster service time to deliver data to vehicles, unlike the existing optimal approaches, which take more time due to distributing the data across more edges, leading the vehicle to travel for more time to get the complete data. It also illustrates the optimality of handling dynamic vehicular route changes using a real dataset.

We also proposed a heuristic scheduler for vehicles which request for service deliveries. The proposed approach is based on the Global Earliest Deadline First algorithm for scheduling. It is able to service more vehicles compared to the partitioning algorithms for scheduling as well as giving priority to vehicles with an earlier deadline at the cost of a slight increment in the time taken to reach this solution. It is also faster than the optimal approach and thus makes it more suitable in real world scenarios with lots of vehicles and edges in the network.

These works while trying to propose approaches for data delivery and computation offloading, still do not consider all real world parameters. Some of these parameters include energy consumption, network parameters, channel loss etc. In the case of data delivery, we also abstract the type of data that is being delivered and consider all data to be equal. But in real world, different types of data can be requested by the vehicles. Similarly, some forms of data can be requested by multiple vehicles at the same time. This could include map data of the region, weather data, local news etc. When the same data is requested by multiple vehicles, especially multiple vehicles passing through the same edge, instead of utilising more resources of the edge node to store multiple copies of the same data to be delivered to multiple vehicles, we can cache this data once and just send this cached data to all the vehicles requesting it, thus efficiently utilising the memory resources of an edge. Thus one future direction to work on would be considering the different types of data and its implications on data delivery via edges by optimizing caching.

Another future work can consider not just road side units as the edge nodes in the VEC framework but also the nearby vehicles as edge nodes. This would lead to the formation of an ad-hoc cloud, where the surrounding vehicles provide the additional computation resources of the edge node. This is useful for computational offloading in areas without much RSU infrastructure, but a lot of connected vehicles.

But these works also will have another set of challenges such as taking care of privacy and security constraints during V2V communication, deciding which data to be cached and which to delete etc that have to be solved.

Appendix A

Other Works

A.1 CG Heuristic

To solve the problem of data delivery to connected vehicles via edge nodes in the VEC ecosystem, we proposed an optimal approach for data delivery with the minimization of the bandwidth cost as the optimal function. This approach was different from the proposed work in [13], since this approach took into consideration the multiple overlap sets of vehicles at the coverage region of all edges, unlike the pessimistic approach of considering all the vehicles passing through an edge, to pass the coverage region at the same time. This approach was able to service more vehicles than the optimal approach which did not consider vehicle overlaps at an edge, since the resource utilization at an edge, was not as constrained as the pessimistic approach. But this optimal approach still took a long time to arrive at the optimal solution, which would not be feasible in real world scenarios. Hence we also proposed a Global Edge Cost-Gradient based Heuristic for fast data delivery allocation. The proposed heuristic was able to calculate the cost gradient for allocating some amount of data m^* to a pair of edges, and thus extrapolate it to the global scenario. This heuristic drastically reduced the run time for the allocation framework, compared to the optimal approach, at the cost of performance, in terms of the number of vehicles serviced. This paper is cited as [17].

Appendix A

Publications

A.1 Relevant Publications

1. **Joseph John Cherukara, SVSLN Surya Suhas Vaddhiparthy, Deepak Gangadharan, BaekGyu Kim**. *”Dynamic Data Delivery Framework for Connected Vehicles via Edge Nodes with Variable Routes”* **Vehicular Technology Conference (VTC) Fall 2023**

2. **Akshaj Gupta, Joseph John Cherukara, Deepak Gangadharan, BaekGyu Kim, Oleg Sokolsky, Insup Lee**. *”Global Edge Bandwidth Cost Gradient-based Heuristic for Fast Data Delivery to Connected Vehicles under Vehicle Overlaps”* **Vehicular Technology Conference (VTC) Spring 2022**

3. **Joseph John Cherukara, Tanniru Abhinav Siddharth , Kethu Sesha Sarath Reddy , Deepak Gangadharan, BaekGyu Kim**. *”Global Earliest Deadline First Algorithm Based Scheduler for Computation Offloading in Vehicular Edge Computing”* **Submitted for Review**

A.2 Unrelated Publications

1. **SVSLN Surya Suhas Vaddhiparthy, Joseph John Cherukara, Deepak Gangadharan, BaekGyu Kim**. *”Collision-Aware Data Delivery Framework for Connected Vehicles via Edges”* **Vehicular Technology Conference (VTC) Fall 2023**

2. **Akshaj Gupta, Joseph John Cherukara, Deepak Gangadharan, BaekGyu Kim, Oleg Sokolsky, Insup Lee**. *”E-PODS: A Fast Heuristic for Data/Service Delivery in Vehicular Edge Computing”* **Vehicular Technology Conference (VTC) Spring 2021**

Bibliography

[1] Automotive edge computing consortium. `https://aecc.org/`.

[2] Operational behavior of a high definition map application. `https://aecc.org/wp-content/uploads/2020/07/Operational_Behavior_of_a_High_Definition_Map_Application.pdf`.

[3] A. Acheampong, Y. Zhang, and X. Xu. A parallel computing based model for online binary computation offloading in mobile edge computing. *Computer Communications*, 203:248–261, 2023.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, 2015.

[5] A. Arunarani, D. Manjula, and V. Sugumaran. Task scheduling techniques in cloud computing: A literature survey. *Future Generation Computer Systems*, 91:407–415, 2019.

[6] G. Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017.

[7] K. Cao, Y. Liu, G. Meng, and Q. Sun. An overview on edge computing research. *IEEE access*, 8:85714–85728, 2020.

[8] C. Chen, H. Li, H. Li, R. Fu, Y. Liu, and S. Wan. Efficiency and fairness oriented dynamic task offloading in internet of vehicles. *IEEE Transactions on Green Comm and Networking*, 6(3), 2022.

[9] L. Codeca, R. Frank, and T. Engel. Luxembourg sumo traffic (lust) scenario: 24 hours of mobility for vehicular networking research. In *2015 IEEE Vehicular Networking Conference (VNC)*, pages 1–8, 2015.

[10] L. Codecá, R. Frank, S. Faye, and T. Engel. Luxembourg sumo traffic (lust) scenario: Traffic demand evaluation. *IEEE Intelligent Transportation Systems Magazine*, 9(2):52–63, 2017.

[11] Y. Dai, D. Xu, S. Maharjan, and Y. Zhang. Joint load balancing and offloading in vehicular edge computing and networks. *IEEE Internet of Things Journal*, 6(3):4377–4387, 2019.

[12] M. Faruk and D. Sivakumar. Multi-layer qos based task scheduling algorithm for cloud environments. *International Journal of Advanced Computer Technology*, 2014.

[13] D. Gangadharan, O. Sokolsky, I. Lee, B. Kim, C.-W. Lin, and S. Shiraishi. Bandwidth optimal data/service delivery for connected vehicles via edges. In *11th IEEE International Conference on Cloud Computing (CLOUD)*, pages 106–113, 2018.

[14] A. Ghavami, Z. Li, and H. Shen. On-demand bandwidth pricing for congestion control in core switches in cloud networks. In *9th IEEE International Conference on Cloud Computing (CLOUD)*, 2016.

[15] L. Guo, S. Zhao, S. Shen, and C. Jiang. Task scheduling optimization in cloud computing based on heuristic algorithm. *Journal of networks*, 7(3):547, 2012.

[16] A. Gupta, J. J. Cherukara, D. Gangadharan, B. Kim, O. Sokolsky, and I. Lee. E-pods: A fast heuristic for data/service delivery in vehicular edge computing. In *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, pages 1–6, 2021.

[17] A. Gupta, J. J. Cherukara, D. Gangadharan, B. Kim, O. Sokolsky, and I. Lee. Global edge bandwidth cost gradient-based heuristic for fast data delivery to connected vehicles under vehicle overlaps. In *95th IEEE Vehicular Technology Conference (VTC)*, pages 1–7. IEEE, 2022.

[18] V. Hayyolalam and A. A. P. Kazem. A systematic literature review on qos-aware service composition and selection in cloud environment. *Journal of Network and Computer Applications*, 110:52–74, 2018.

[19] L. Huang, X. Feng, C. Zhang, L. Qian, and Y. Wu. Deep reinforcement learning-based joint task offloading and bandwidth allocation for multi-user mobile edge computing. *Digital Communications and Networks*, 5(1):10–17, 2019.

[20] A. Islam, A. Debnath, M. Ghose, and S. Chakraborty. A survey on task offloading in multi-access edge computing. *Journal of Systems Architecture*, 118:102225, 2021.

[21] J. Jeong, S. Guo, Y. Gu, T. He, and D. H. Du. Trajectory-based statistical forwarding for multihop infrastructure-to-vehicle data delivery. *IEEE Transactions on Mobile Comp*, 11:1523–1537, 2012.

[22] X. Jiang, N. Guan, D. Liu, and W. Liu. Analyzing gedf scheduling for parallel real-time tasks with arbitrary deadlines. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1537–1542. IEEE, 2019.

[23] S. Jošilo and G. Dán. Wireless and computing resource allocation for selfish computation offloading in edge computing. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 2467–2475. IEEE, 2019.

[24] T.-Y. Kan, Y. Chiang, and H.-Y. Wei. Task offloading and resource allocation in mobile-edge computing system. In *2018 27th wireless and optical communication conference (WOCC)*, pages 1–4. IEEE, 2018.

[25] W. Labidi, M. Sarkiss, and M. Kamoun. Energy-optimal resource scheduling and computation offloading in small cell networks. In *2015 22nd international conference on telecommunications (ICT)*, pages 313–318. IEEE, 2015.

[26] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief. Delay-optimal computation task scheduling for mobile-edge computing systems. In *2016 IEEE international symposium on information theory (ISIT)*, pages 1451–1455. IEEE, 2016.

[27] Q. Liu, H. Zeng, M. Chen, and L. Liu. Cost minimization in multi-path communication under throughput and maximum delay constraints. In *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pages 2263–2272, 2020.

[28] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE communications surveys & tutorials*, 19(4):2322–2358, 2017.

[29] R. Meneguette, R. De Grande, J. Ueyama, G. P. R. Filho, and E. Madeira. Vehicular edge computing: Architecture, resource management, security, and challenges. *ACM Comput. Surv.*, 55(1), 2021.

[30] M. Othman, S. A. Madani, S. U. Khan, et al. A survey of mobile cloud computing application models. *IEEE communications surveys & tutorials*, 16(1):393–413, 2013.

[31] H. Qi and A. Gani. Research on mobile cloud computing: Review, trend and perspectives. In *2012 second international conference on digital information and communication technology and it's applications (DICTAP)*, pages 195–202. ieee, 2012.

[32] Y. Sun, Z. Wu, D. Shi, and X. Hu. Task offloading method of internet of vehicles based on cloud-edge computing. In *2022 IEEE International Conference on Services Computing (SCC)*, pages 315–320, 2022.

[33] C. Tang, C. Zhu, N. Zhang, M. Guizani, and J. J. P. C. Rodrigues. Sdn-assisted mobile edge computing for collaborative computation offloading in industrial internet of things. *IEEE Internet of Things Journal*, 9(23):24253–24263, 2022.

[34] L. Tang, B. Tang, L. Zhang, F. Guo, and H. He. Joint optimization of network selection and task offloading for vehicular edge computing. *Journal of Cloud Computing*, 10(1):1–13, 2021.

[35] K. C. Toh, M. J. Todd, and R. H. Tütüncü. Sdpt3 — a matlab software package for semidefinite programming, version 1.3. *Optimization Methods and Software*, 11(1-4):545–581, 1999.

[36] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas. Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning. *IEEE Communications Magazine*, 57(5):64–69, 2019.

[37] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas. Fast adaptive task offloading in edge computing based on meta reinforcement learning. *IEEE Transactions on Parallel and Distributed Systems*, 32(1):242–253, 2020.

[38] L. Wang, L. Jiao, J. Li, and M. Mühlhäuser. Online resource allocation for arbitrary user mobility in distributed edge clouds. In *37th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 1281–1290, 2017.

[39] F. Wei, S. Chen, and W. Zou. A greedy algorithm for task offloading in mobile edge computing system. *China Communications*, 15(11):149–157, 2018.

[40] R. Xie, Q. Tang, Q. Wang, X. Liu, F. R. Yu, and T. Huang. Collaborative vehicular edge computing networks: Architecture design and research challenges. *IEEE Access*, 7:178942–178952, 2019.

[41] K. Yang and J. H. Anderson. Optimal gedf-based schedulers that allow intra-task parallelism on heterogeneous multiprocessors. In *2014 IEEE 12th Symposium on Embedded Systems for Real-time Multimedia (ESTIMedia)*, pages 30–39. IEEE, 2014.

[42] L. Yang, B. Liu, J. Cao, Y. Sahni, and Z. Wang. Joint computation partitioning and resource allocation for latency sensitive applications in mobile edge clouds. *IEEE Transactions on Services Computing*, 14(5):1439–1452, 2019.

[43] J. Zhang, H. Guo, J. Liu, and Y. Zhang. Task offloading in vehicular edge computing networks: A load-balancing solution. *IEEE Transactions on Vehicular Technology*, 69(2):2092–2104, 2020.