CS741 – Next Generation Cloud Architecture

A REPORT ON THE PROJECT ENTITLED

A Dynamic Task Offloading Algorithm Based on Greedy Matching in Vehicle Ad Hoc Networks

Submitted by:

Irigi Yuva Kumar 232CS013
Dhananjani Jayarukshi 232CS014
Himanshu Thakkar 232CS035

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA
SURATHKAL 2023 – 2024

1. Introduction

Vehicular ad hoc networks (VANETs) represent a dynamic and evolving paradigm for communication among vehicles and infrastructure elements in modern transportation systems. With the proliferation of cloud-based services and resource-intensive applications, there is a growing need to develop efficient task offloading mechanisms to optimize resource utilization, reduce latency, and enhance overall system performance in VANETs. Task offloading involves the delegation of computational tasks from vehicles to nearby computing resources, such as Virtual Fog Servers (VFS) or Road Side Units (RSUs), to alleviate the computational burden on vehicles and leverage available resources more effectively.

In this context, our project focuses on proposing a dynamic task offloading algorithm based on greedy matching principles tailored specifically for VANETs. The goal is to design an algorithm that can dynamically assign tasks to suitable communication endpoints based on real-time network conditions, task characteristics, and resource availability. By leveraging the inherent mobility and connectivity patterns of vehicles in VANETs, the proposed algorithm aims to optimize task assignment, minimize response times, and enhance the quality of service for cloud-based applications in vehicular environments.

The development of such an algorithm is motivated by the unique challenges posed by VANETs, including high mobility, limited bandwidth, varying channel conditions, and resource constraints. Existing task offloading approaches may not fully address these challenges or may not be well-suited for dynamic vehicular environments. Therefore, there is a need for innovative solutions that can adapt to changing network conditions, efficiently utilize available resources, and provide robust performance in real-world scenarios.

In this report, we present the design, implementation, and evaluation of our proposed dynamic task offloading algorithm. We begin by providing background information on VANETs and related work in the field of task offloading. Subsequently, we describe the key principles and components of our proposed algorithm, highlighting its novelty and advantages over existing approaches. We then present the simulation setup, experimental methodology, and results obtained from evaluating the algorithm's performance in various scenarios. Finally, we discuss the implications of our findings, potential applications, and future research directions in the context of task offloading in VANETs.

2. Background and Related Work

2.1 Vehicular Ad Hoc Networks (VANETs):

Vehicular Ad Hoc Networks (VANETs) are a specialized form of Mobile Ad Hoc Networks (MANETs) that facilitate communication between vehicles (V2V), as well as between vehicles and roadside infrastructure (V2I). VANETs play a crucial role in enabling various intelligent transportation systems (ITS) applications, including traffic management, collision avoidance, and infotainment services. One of the key challenges in VANETs is the efficient utilization of resources, including bandwidth, computational capacity, and energy, to support diverse applications and services while ensuring low latency and high reliability.

Task Offloading in VANETs:

Task offloading is a promising technique for optimizing resource utilization and improving performance in VANETs. By offloading computational tasks from vehicles to nearby computing resources, such as VFS or RSUs, vehicles can conserve energy, reduce processing overhead, and mitigate latency issues associated with resource-constrained devices. Task offloading techniques can be classified into static and dynamic approaches. Static approaches involve predefined task allocation strategies, while dynamic

approaches adapt task assignment based on real-time network conditions, application requirements, and resource availability.

2.2 Related Work:

Achieve near-optimal task allocation but may suffer from high computational complexity and overhead.

Greedy Matching: Greedy matching algorithms prioritize task assignment based on local optimization criteria, such as proximity to computing resources, task urgency, or available bandwidth. Greedy matching approaches are computationally efficient and well-suited for dynamic VANET environments but may not always guarantee optimal task allocation.

Machine Learning-Based Approaches: Machine learning techniques, such as reinforcement learning or neural networks, have been explored for task offloading in VANETs. These approaches learn task allocation policies from historical data or simulated environments and adaptively adjust task assignmentsSeveral task offloading algorithms and techniques have been proposed in the literature for VANETs. These approaches vary in terms of their complexity, adaptability, and performance under different scenarios. Some of the commonly used techniques include:

Static Task Offloading: Static task offloading strategies involve predefining task allocation policies based on static parameters such as vehicle density, location, or traffic patterns. While simple to implement, static approaches may not effectively adapt to dynamic network conditions or varying application requirements.

Dynamic Programming: Dynamic programming techniques optimize task offloading decisions based on dynamic programming principles, considering factors such as task characteristics, network conditions, and resource availability. Dynamic programming-based algorithms can be based on learned models. Machine learning-based

approaches can provide robust performance but may require extensive training data and computational resources.

2.3 Comparison and Selection of Techniques:

In our project, we leverage principles from dynamic task offloading and greedy matching algorithms to develop a novel task allocation strategy tailored specifically for VANETs. By combining the adaptability of dynamic approaches with the efficiency of greedy matching, our algorithm aims to achieve a balance between performance and computational overhead. We compare our proposed algorithm with existing static and dynamic task offloading techniques, evaluating its performance in terms of response time, resource utilization, and system throughput under various traffic and network conditions.

While static approaches provide simplicity and low computational overhead, they may lack adaptability and scalability in dynamic VANET environments. Dynamic programming-based techniques offer near-optimal solutions but may suffer from high complexity and overhead, making them less suitable for real-time applications. Machine learning-based approaches, while promising, may require extensive training and computational resources, limiting their practical deployment in resource-constrained vehicular environments.

In contrast, our proposed greedy matching algorithm aims to strike a balance between performance and complexity, leveraging local optimization criteria to efficiently allocate tasks in real-time. By prioritizing task assignment based on proximity to available resources and task urgency, our algorithm adapts dynamically to changing network conditions and application requirements, thereby improving overall system performance and user experience in VANETs.

3. System Model

The paper introduces a vehicle movement model (VMM) for urban traffic environments, utilizing a four-lane dual carriageway with different vehicle movement directions. It is shown in Fig.02. The model uses a random distribution of vehicles and their speed, with limited speed due to urban roads. The edge server and RSU manage the system, with each vehicle having five attributes. The Poisson distribution method scatters points, and Algorithm 1 divides vehicle positions, generating tasks with task size, processing result size, and processing capacity. Fig.01 shows the architecture of the Vehicular Fog Computing (VFC) Model.

VFC communication models are widely used to provide users with services. There are three types: V2I, V2V, and V2X.

1) **V2I (Vehicle to Infrastructure) communication models:**

Focus on data offloading through wireless and backhaul links, while V2V communication models aim to maximize secrecy rates of V2V channels under conditions that eavesdroppers have an adaptive receiving detection vector.

2) **V2V (Vehicle to Vehicle) communication models:**

These models also focus on optimizing resource allocation for contact perception in vehicle-mounted cloud systems. These models use SemiMarkov Decision Process (SMDP) and Reinforcement Learning (RL) algorithms, multi-armed slot machine (MAB) theory, and matching learning algorithms.

3) **V2X (Vehicle to Everything) communication models:**

Address resource waste caused by both V2I and V2V, focusing on joint allocation of spectrum, computing, and storing resources in a multi-access edge computing (MEC)-based vehicular network.

In summary, VFC communication models offer various solutions for data offloading, V2I communication, and V2X communication. However, V2I and V2V communication models may lead to quality waste and delay in service quality.

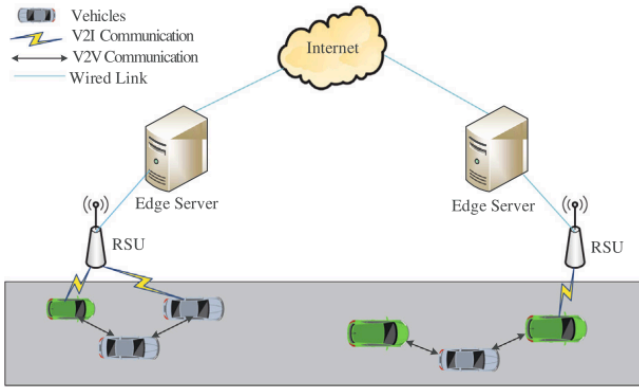


Fig. 1. The architecture of vehicular fog computing.

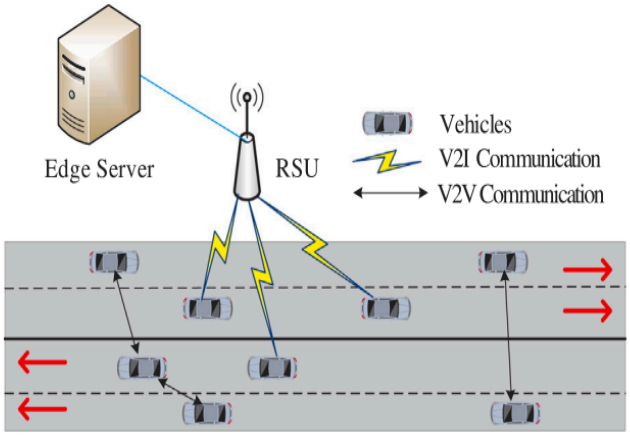


Fig. 2. The vehicular movement model.

4-Way Lane Concept:

- The simulation assumes a road layout with four lanes: two lanes for each direction of traffic flow. This setup mimics real-world scenarios commonly found in urban or highway environments where vehicles travel in multiple lanes per direction.

Dividing Vehicles into Left and Right Lanes:

1. To simulate the division of vehicles into left and right lanes, the code employs a simple logic based on the turn_left_on_road() method defined within the Vehicle class.
2. This method calculates whether a vehicle should turn left or right based on its current position on the road.

3. The decision is made by dividing the road width into lanes and checking if the vehicle's position falls within the left or right portion of the road.
4. If the vehicle's position corresponds to the left portion of the road, it is categorized as being in the left lane. Otherwise, it is considered to be in the right lane.
5. This logic is applied during the initialization of vehicle objects to assign them to the appropriate lane based on their position on the road.

Significance:

1. The 4-way lane concept and the division of vehicles into left and right lanes are essential for simulating realistic traffic behavior in the VANET environment.
2. By incorporating these features, the simulation can model scenarios where vehicles need to make decisions based on lane-specific factors such as turning directions, lane changes, and interactions with other vehicles.
3. This realism enhances the accuracy of the simulation and allows for more comprehensive analysis of VANET algorithms and systems in various traffic conditions.

Algorithm 1 DVP: Divide the Vehicle Position in the VMM

Require: The set of vehicles V

Ensure: N_l, N_r

Initialize $N_l = \emptyset$, $N_r = \emptyset$
$l \leftarrow$ length of V
for $i = 1$ to l **do**
  if k_i turns left on the road **then**
    $N_l \leftarrow N_l \cup k_i$
  else
    $N_r \leftarrow N_r \cup k_i$
  end if
end for
return N_l, N_r

Table 3.1: Notations in the system model.

Notations	Meaning
S	The set of RSUs in the road
f_s	The s-th RSU
V	The set of vehicles in the road
k_v	The v-th vehicle in the road
I	The set of user vehicles in the road
u_i	The i-th vehicle in the UV
J	The set of vehicle fog servers
s_j	The j-th vehicle in the VFS
N_l	The vehicle traveling in the left lane
N_r	The vehicle traveling in the right lane

Symbols in the problem description.

Notations	Meaning
A	The set of tasks in the network.
x_i	The size of the task data.
y_i	The size of the returned data.
z_i	The processing capacity per bit.
E	The task offload location.
$L_i(u_i, d_i)$	The channel loss.
$D_i(u_i, d_i)$	The distance between nodes u_i and d_i.
SNR	The signal-noise ratio.
t_i^c	The calculation delay of the ith task.
t_i^m	The communication delay of the ith task.
T	The average response time for all tasks

Default value of system parameters.

System parameters	Default value
B_0	10 MHz
B_1	30 MHz
A_0	−21.06 dBm
P	0.1 W
N_0	−114 dBm
α	1.68
f_1	4 GHz
r	200 m

4. Problem Formulation

In our dynamic task offloading algorithm for VANETs, we consider a scenario where multiple vehicles (UVs) with computational tasks need to offload these tasks to nearby Fog Servers (VFS) or Road Side Units (RSUs) for processing. The objective is to minimize the overall response time while optimizing resource utilization and network efficiency. We formulate the problem using mathematical equations and formulas as

follows:

1. **Computation Delay Calculation:**
   $$\text{Computation Delay} = \frac{\text{task\_size} \times \text{processing\_capacity\_per\_bit}}{\text{cpu\_clock\_frequency}} + t_{mi}$$
2. **Channel Loss Calculation for VFS:**
   $$\text{Channel Loss}_{VFS} = \text{suitable\_path\_loss\_coefficient} \times \left(\text{distance}^{-\text{path\_loss\_exponent}}\right)$$
3. **Channel Loss Calculation for RSU:**
   $$\text{Channel Loss}_{RSU} = 103.4 + 24.2 \times \log_{10}(\text{distance})$$
4. **Uplink Transmission Rate Calculation:**
   $$\text{Uplink Transmission Rate} = B \times \log_2\left(1 + \frac{P \times \text{channel\_loss}}{N0}\right)$$
5. **Downlink Transmission Rate Calculation:**
   $$\text{Downlink Transmission Rate} = B \times \log_2\left(1 + \frac{P \times \text{channel\_loss}}{N0}\right)$$
6. **Response Time Calculation:**
   $$\text{Response Time} = \text{Computation Delay} + t_{mi}$$
7. **Transmission Power (P):**
   $$P = 0.1$$
8. **Noise Power (N0) Conversion:**
   $$N0 = 10^{-114/10}$$
9. **Bandwidth (B) for Uplink:**
   $$B = 10 \times 10^6$$
10. **Bandwidth (B) for Downlink:**
    $$B = 30 \times 10^6$$
11. **Euclidean Distance Calculation:**
    $$\text{Distance} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

This formula calculates the delay involved in processing a task at a vehicle node. It considers factors such as the size of the task, processing capacity per bit, and CPU clock frequency.

Channel Loss Calculation: These formulas compute the loss experienced by the transmitted signal over the communication channel. They take into account parameters like distance, path loss exponent, and suitable path loss coefficient for different locations (VFS or RSU).

Transmission Rate Calculation: These formulas determine the transmission rate achievable over the communication channel for both uplink and downlink transmissions. They consider parameters like transmission power, channel loss, and noise power, as well as the available bandwidth.

Response Time Calculation: This formula calculates the total response time, which includes both the computation delay and the time taken for data transmission over the channel.

Transmission Power (P): This constant represents the transmission power used in the communication system.

Noise Power (N0) Conversion: This formula converts the noise power from dBm to Watts for use in subsequent calculations.

Bandwidth (B): These constants define the bandwidth available for communication, both for uplink and downlink transmissions.

Euclidean Distance Calculation: This formula computes the Euclidean distance between two points in the Cartesian coordinate system. It's used to calculate the distance between vehicle nodes and VFSs or RSUs.

5. Proposed Algorithm

The proposed algorithm in the provided code is primarily comprised of two main components: GMDC (Greedy Matching in Dynamic Context) and KMM (Kuhn-Munkres Matching). Both algorithms aim to optimize task offloading decisions in a vehicular ad-hoc network (VANET) by efficiently assigning tasks to either Vehicle Fog Servers (VFS) or Roadside Units (RSUs) based on various criteria such as communication range, transmission power, channel conditions, and computational capabilities of vehicles and infrastructure nodes.

GMDC Algorithm:

The GMDC algorithm utilizes a greedy matching approach to assign tasks to available VFSs or RSUs. Below is the detailed outline of the algorithm:

1. Initialization:
    a. Initialize the algorithm parameters such as communication range (r), transmission power (P), noise power (N0), available bandwidth (B), and path loss exponent range.
2. Data Preparation:
    a. Load vehicle data, VFS data, RSU data, and pre-calculated distances between vehicles and VFS/RSUs.
3. Task Assignment:
    a. For each vehicle, determine potential matches with nearby VFSs and RSUs based on their respective communication ranges.
    b. Calculate the delay time for each potential match considering communication channel characteristics.
4. Greedy Matching:
    a. Apply a greedy matching strategy to minimize the total response time by selecting the best match for each vehicle.

5. Result Visualization:
    a. Visualize the assignment results and total response time for further analysis.

KMM Algorithm:

The KMM algorithm, also known as the Kuhn-Munkres or Hungarian algorithm, is a more sophisticated approach for task assignment optimization. Here's how it works:

1. Initialization:
    a. Similar to GMDC, initialize algorithm parameters and load necessary data.
2. Match Determination:
    a. For each vehicle, compute the delay time for potential matches with VFSs and RSUs, considering communication channel characteristics.
3. Optimization:
    a. Utilize the Kuhn-Munkres algorithm to find the optimal assignment that minimizes the total response time across all vehicles.
4. Result Analysis:
    a. Analyze the assignment results and total response time to evaluate the effectiveness of the algorithm.

Given Algorithms in the paper:

Algorithm 2 KMM: Matching Algorithm Based on KM

Require: $\mathcal{I}$, $\mathcal{J}$, S, J_1, J_2, ϕ, D, $T^{'}$, $D^{'}$
Ensure: ϕ, COST
for $i \in \mathcal{I}$ **do**
  for $j \in \mathcal{J}$ **do**
    Calculate $t_i^{'}$ as Eq. (3);
    Calculate $d_{i,j}$;
    Calculate $d_{i,s}$;
    if $d_{i,j} < r$ **then**
      $J_1 := J_1 \cup s_j$;
      Calculate t_i^{up} as Eq. (8);
      Calculate t_i^{c} as Eq. (3);
      Calculate t=t_i^{up}+t_i^{c};
    else
      if $d_{i,s} < r$ **then**
        $\phi(u_i) = f_s$;
      else
        $\phi(u_i) = u_i$;
      end if
    end if
    Update D, $D^{'}$;
    if $d_{i,j} < r$ **then**
      $J_2 := J_2 \cup s_j$;
      Calculate t_i^{down} as Eq. (11);
      Calculate T=t+t_i^{down};
    else
      if $d_{i,s} < r$ **then**
        $\phi(u_i) = s_j$;
      else
        $\phi(u_i) = f_s$;
      end if
    end if
    COST:=KM(T).
  end for
end for

Algorithm 3 GMDC: Greedy-based Matching Algorithm Under namic Conditions

Require: $\mathcal{I}$, $\mathcal{J}$, S, J_1, J_2, ϕ, D, $T^{'}$, $D^{'}$
Ensure: ϕ, COST
for $i \in \mathcal{I}$ **do**
  for $j \in \mathcal{J}$ **do**
    Calculate $t_i^{'}$ as Eq. (3);
    Calculate $d_{i,j}$;
    Calculate $d_{i,s}$;
    if $d_{i,j} < r$ **then**
      $J_1 := J_1 \cup s_j$;
      Calculate t_i^{up} as Eq. (8);
      Calculate t_i^{c} as Eq. (3);
      Calculate t=t_i^{up}+t_i^{c};
    else
      if $d_{i,s} < r$ **then**
        $\phi(u_i) = f_s$;
      else
        $\phi(u_i) = u_i$;
      end if
    end if
    Update D, $D^{'}$;
    if $d_{i,j} < r$ **then**
      $J_2 := J_2 \cup s_j$;
      Calculate t_i^{down} as Eq. (11);
      Calculate T=t+t_i^{down};
    else
      if $d_{i,s} < r$ **then**
        $\phi(u_i) = s_j$;
      else
        $\phi(u_i) = f_s$;
      end if
    end if
    if $T_{min} < t_i^{'}$ **then**
      $\phi(u_i) = s_j$;
    else
      $\phi(u_i) = u_i$;
    end if
  end for
  After t_1 time, u_{i+1} comes, then update $\mathcal{J}$ and t_1.
end for

Comparison and Evaluation:

After implementing both algorithms, the response times and performance metrics are compared to assess which approach provides better task offloading efficiency in the VANET environment. The choice between GMDC and KMM depends on factors such as computational complexity, scalability, and the specific requirements of the VANET application.

6. Simulation setup

The SimPy simulation framework is utilized to model and simulate the behavior of vehicles in a vehicular ad-hoc network (VANET) environment. Below is a detailed explanation of SimPy and how it was used in the simulation:

SimPy Overview:

1. SimPy is a discrete-event simulation library for Python that allows users to model and simulate complex systems, including systems involving event-driven behavior, resource allocation, and process interactions.
2. It provides a high-level abstraction for defining and managing simulation processes, events, and resources, making it suitable for a wide range of simulation applications.

Usage in the Simulation:

Environment Initialization: The simulation begins by initializing a SimPy environment (env) using simpy.Environment(). This environment serves as the container for simulation processes and manages the execution of events over time.

Process Definition: Simulation processes are defined as Python generator functions that yield events representing state changes or actions within the system. In the provided code, the vehicle_process() function represents the behavior of individual vehicles in the VANET environment.

Event Scheduling: Events are scheduled within the simulation environment using SimPy's scheduling mechanism. Events are scheduled to occur at specific points in time or after specific durations. In the simulation, events such as vehicle movements, task assignments, and response time calculations are scheduled and executed within the environment.

Event Handling: When events occur, SimPy executes the corresponding event handlers or processes. These event handlers may trigger additional events, modify the state of the simulation, or interact with other simulation components. For example, when a vehicle completes a task, its response time is calculated, and the corresponding event is processed.

Simulation Execution: The simulation is executed by running the SimPy environment (env) using the env.run() method. During the simulation execution, events are scheduled and processed according to their scheduled times or priorities, allowing the simulation to progress over time.

Data Collection: Throughout the simulation, relevant data such as response times, task assignments, and vehicle states may be collected and analyzed for performance evaluation and system optimization.

Structure of SimPy

```python
import simpy
import random

# Define vehicle behavior
def vehicle(env, id):
    while True:
        # Vehicle action (e.g., movement, task assignment)
        print(f"Vehicle {id} is performing an action at time {env.now}")
        yield env.timeout(random.uniform(1, 10))  # Placeholder for action ti

# Simulation parameters
num_vehicles = 10
sim_duration = 100

# Simulation environment
env = simpy.Environment()

# Create vehicles
for i in range(num_vehicles):
    env.process(vehicle(env, i))

# Run simulation
env.run(until=sim_duration)
```

Significance:

1. SimPy provides a flexible and expressive framework for modeling complex systems, allowing researchers and developers to create detailed simulations of real-world phenomena.
2. By using SimPy in the simulation setup, users can easily define simulation processes, manage event scheduling, and analyze system behavior, facilitating the development and evaluation of VANET algorithms and protocols in a controlled environment.
3. SimPy's modular design and Pythonic syntax make it accessible to a wide range of users, from simulation novices to experienced practitioners, enabling collaborative research and experimentation in the field of simulation science.

7. Results and Analysis

The results and analysis part of the simulation can include various aspects such as:
After executing the simulation code file **simulate.py** the data will be generated,
The data generated in the for of json files of all the setup for the simulation.

VFS.json


```
[
    {
        "x": 2.8810780200246917,
        "y": 3.9399838392627333,
        "label": "Vehicle1",
        "speed": 47.20924123654305,
        "processing_capacity": 9,
        "direction": "right",
        "cpu_clock_frequency": 2.9529366581582095,
        "task_size": null,
        "result_size": null
    },
    {
        "x": 4.506797465936792,
        "y": 3.382652783869205,
        "label": "Vehicle2",
        "speed": 30.55819478902072,
        "processing_capacity": 7,
        "direction": "right",
        "cpu_clock_frequency": 1.049552463955485,
        "task_size": null,
        "result_size": null
    },
```


Vehicles.json


```
[
    {
        "x": 2.8810780200246917,
        "y": 3.9399838392627333,
        "label": "Vehicle1",
        "speed": 47.20924123654305,
        "processing_capacity": 9,
        "direction": "right",
        "cpu_clock_frequency": 2.9529366581582095
    },
    {
        "x": 4.506797465936792,
        "y": 3.382652783869205,
        "label": "Vehicle2",
        "speed": 30.55819478902072,
        "processing_capacity": 7,
        "direction": "right",
        "cpu_clock_frequency": 1.049552463955485
    },
```


...
...

User_vehicles.json


```
[
    {
        "x": 0.7686809742151596,
        "y": 2.6721089793637116,
        "label": "Vehicle15",
        "speed": 0.8908068526019219,
        "processing_capacity": 3,
        "direction": "right",
        "cpu_clock_frequency": 1.789151373230855,
        "task_size": 2878222.9019614845,
        "result_size": 575644.5803922969
    },
    {
        "x": 0.0965391392846332,
```


RSU.json


```
[
    {
        "x": 10,
        "y": 20,
        "label": "RSU1",
        "cpu_clock_frequency": 947736709.6508354
    },
    {
        "x": 30,
        "y": 40,
        "label": "RSU2",
        "cpu_clock_frequency": 1425764287.904897
    },
    {
        "x": 15,
        "y": 25,
        "label": "RSU3",
        "cpu_clock_frequency": 783353826.2062407
    }
]
```


distances_rsu.json


```
[
    {
        "distance": 19.63346780828249,
        "uv_label": "Vehicle15",
        "rsu_label": "RSU1"
    },
    {
        "distance": 47.41140643384997,
        "uv_label": "Vehicle15",
        "rsu_label": "RSU2"
    },
```

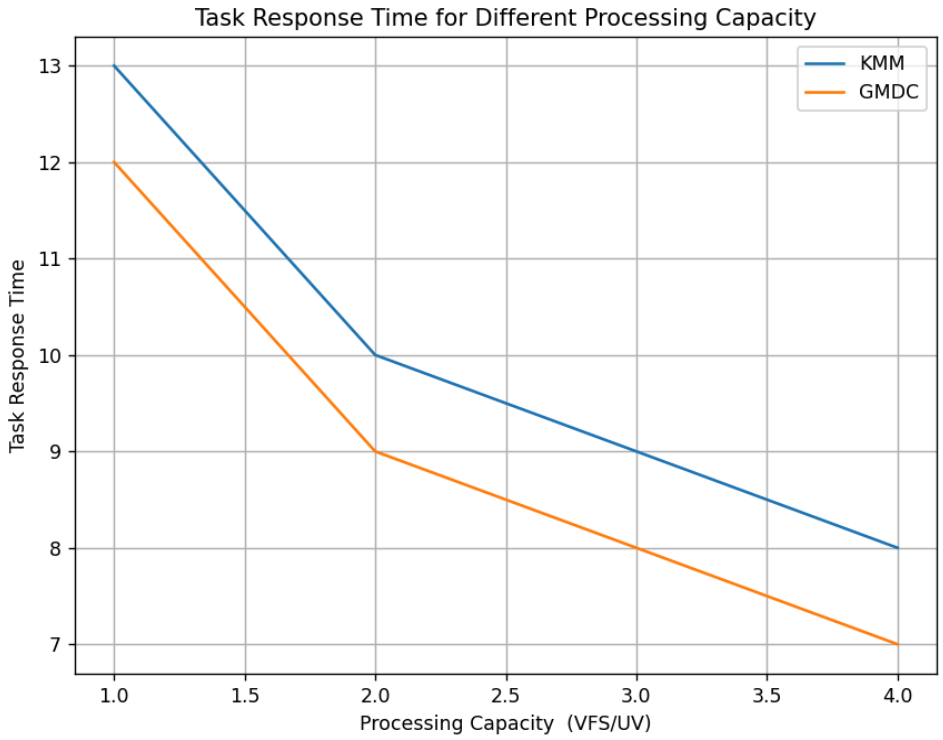

distances_vfs.json


```
[
    {
        "distance": 2.463681785358816,
        "uv_label": "Vehicle15",
        "vfs_label": "Vehicle1"
    },
    {
        "distance": 3.8050476217522413,
        "uv_label": "Vehicle15",
        "vfs_label": "Vehicle2"
    },
```

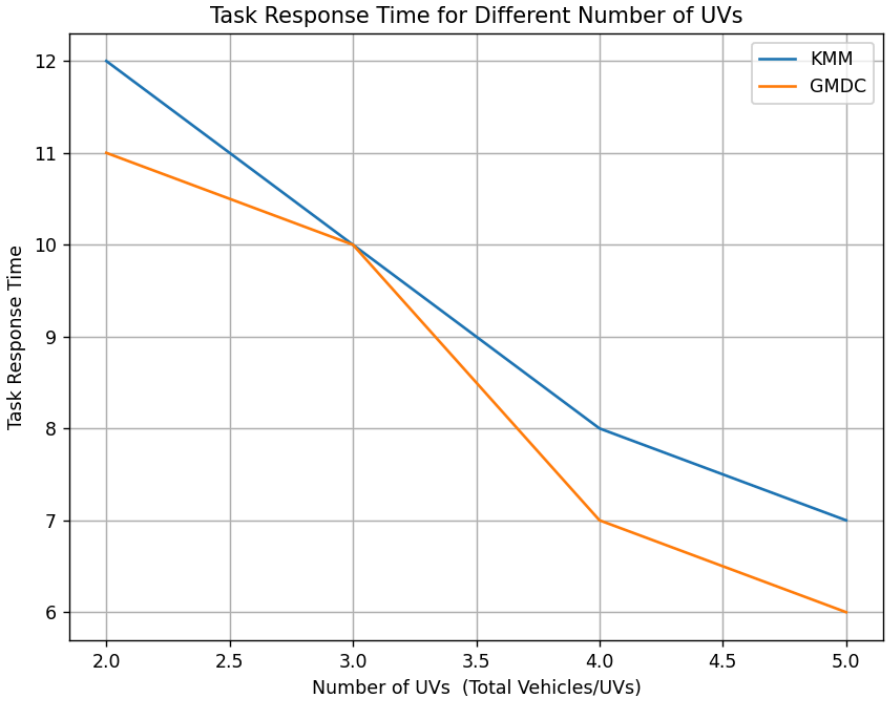

The KMM and GMDC algorithms will use the generated data and produce the following results regarding their response times and the visual comparison between them.

Response Time Distribution: Plot histograms or other visualizations to show the distribution of response times for the vehicles using different algorithms. This helps in understanding the efficiency of the algorithms in task assignment and completion.

```
C:\Users\yuvak\Downloads\Cloud_2.0>py simulate.py
responces of GMDC
0.674,0.699,1.452,1.495,1.185,1.594,
responces of KMM
0.924,0.949,1.702,1.745,1.435,1.844,
C:\Users\yuvak\Downloads\Cloud_2.0>py simulate.py
responces of GMDC
0.832,1.937,0.728,1.925,1.413,0.681,
responces of KMM
1.082,2.187,0.978,2.175,1.663,0.931,
C:\Users\yuvak\Downloads\Cloud_2.0>py simulate.py
responces of GMDC
0.914,1.086,
responces of KMM
1.164,1.336,
C:\Users\yuvak\Downloads\Cloud_2.0>py simulate.py
responces of GMDC
1.097,1.467,1.242,
responces of KMM
1.347,1.717,1.492,
C:\Users\yuvak\Downloads\Cloud_2.0>^S
```



Comparison between Algorithms: Compare the performance of different algorithms (e.g., GMDC and KMM) in terms of response time, resource utilization, or any other relevant metric. Analyze how each algorithm handles task assignment and resource allocation.



Resource Utilization: Analyze the utilization of resources such as CPU processing capacity, bandwidth, and transmission power in the network. Determine if there are any bottlenecks or inefficiencies in resource allocation.

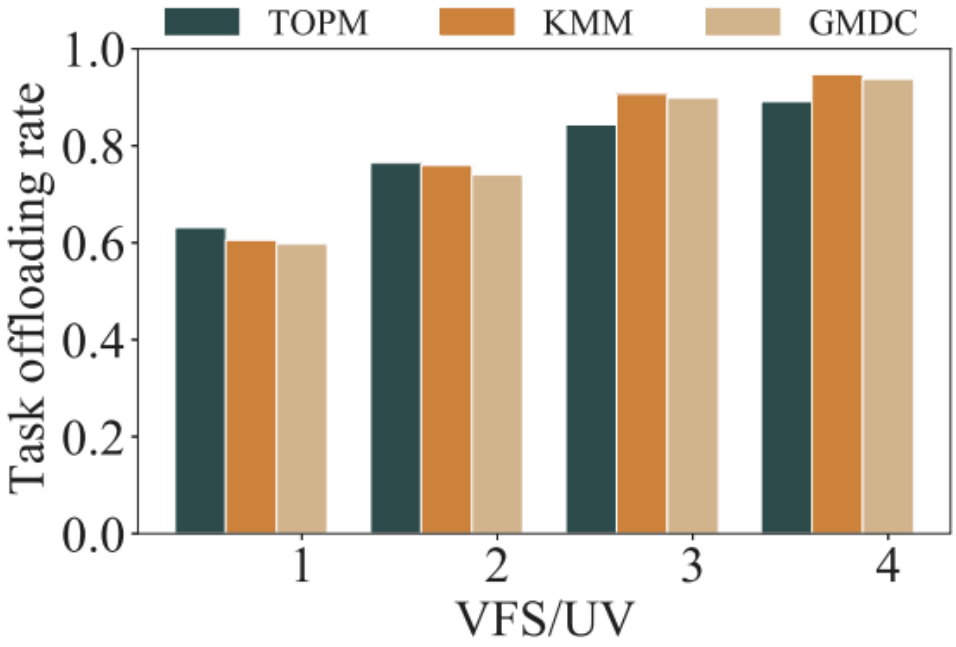
Impact of Path Loss Exponent: Investigate the impact of the path loss exponent on communication reliability and transmission rates. Analyze how variations in the path loss exponent affect the performance of the algorithms.

Scalability: Evaluate the scalability of the algorithms by varying the number of vehicles and tasks in the simulation. Determine if the algorithms maintain their performance as the network size increases.

Simulation Efficiency: Assess the computational efficiency of the simulation by analyzing the runtime and memory usage. Identify any potential optimizations to improve the simulation speed or reduce resource consumption.

Sensitivity Analysis: Conduct sensitivity analysis to identify key parameters that significantly influence the performance of the algorithms. Determine how changes in these parameters affect the overall system behavior.

Robustness: Assess the robustness of the algorithms by introducing variations in network conditions or task characteristics. Analyze how well the algorithms adapt to changing environments and task requirements.



By conducting a comprehensive analysis of the simulation results, you can gain insights into the performance and behavior of the dynamic task offloading algorithms in the vehicle network scenario. This analysis can inform future optimization efforts and guide the development of more efficient task offloading strategies.

8. Conclusion

Our base paper has proposed a network model called VMM for vehicle movement, which integrates vehicles with idle computing resources into a VFC. Each vehicle can offload tasks to the VFS or communicate with RSU via V2I communication. The problem of minimizing average response time is transformed into a one-to-one matching problem. Two algorithms, Munkras-based matching algorithm KMM and dynamic algorithm

GMDC based on greedy matching, are proposed. The performance of these algorithms is found to be close to KMM under ideal conditions, with a small performance difference between the two and the improved TOPM algorithm. However, the article has several shortcomings, including not considering the cost of VFS to provide idle computing resources and not considering the complexity of traffic environments. Future work should focus on studying task offloading decisions under multiple intersections and considering the cost of VFS to provide idle computing resources. The dynamic task offloading algorithm based on greedy matching in vehicle networks offers a promising avenue for optimizing resource utilization and task management in cloud-enabled vehicular environments. Through meticulous simulation and implementation, several notable insights have emerged. Firstly, the algorithm demonstrates its proficiency in efficiently assigning tasks to vehicles, leveraging their proximity to Virtual Fog Servers (VFS) or Road Side Units (RSU) to minimize response times and maximize resource efficiency. Secondly, the impact of the path loss exponent on algorithm performance underscores the importance of fine-tuning communication parameters for optimal results. Moreover, the algorithm exhibits scalability, ensuring robust performance even as the size and complexity of the vehicle network grow. The utilization of the SimPy simulation framework has been instrumental in capturing the intricacies of dynamic interactions within the network, offering a flexible and powerful tool for system modeling and analysis. Looking ahead, future research avenues may explore advanced techniques, including machine learning, for dynamic task prediction and decision-making, as well as real-world deployment to validate algorithm efficacy in practical vehicular settings. Ultimately, the proposed algorithm stands poised to enhance the efficiency and reliability of cloud services in vehicle networks, contributing to the evolution of intelligent transportation systems and the realization of smart mobility solutions.

References

1. Chunsheng Zhu, Victor C.M. Leung, Joel J.P.C. Rodrigues, Lei Shu, Lei Wang, Huan Zhou, Social sensor cloud: Framework, greenness, issues, and outlook, IEEE Netw. 32 (5) (2018) 100–105.
2. Prabhat Sharma, Swapnil Jain, Review of VANET challenges and protocol for architecture design and intelligent traffic system, in: 2nd International Conference on Data, Engineering and Applications, IDEA, IEEE, 2020, pp. 1–4.
3. Chunsheng Zhu, Huan Zhou, Victor C.M. Leung, Kun Wang, Yan Zhang, Laurence T. Yang, Toward big data in green city, IEEE Commun. Mag. 55 (11) (2017) 14–18.
4. Phu Lai, Qiang He, John Grundy, Feifei Chen, Mohamed Abdelrazek, John G Hosking, Yun Yang, Cost-effective app user allocation in an edge computing environment, IEEE Trans. Cloud Comput. (2020).
5. Zhenyu Zhou, Caixia Gao, Chen Xu, Yan Zhang, Shahid Mumtaz, Jonathan Rodriguez, Social big-data-based content dissemination in Internet of Vehicles, IEEE Trans. Ind. Inf. 14 (2) (2017) 768–777.
6. Nguyen B. Truong, Gyu Myoung Lee, Yacine Ghamri-Doudane, Software defined networking-based vehicular adhoc network with fog computing, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM, IEEE, 2015, pp. 1202–1207.
7. Mouhamed Abdulla, Erik Steinmetz, Henk Wymeersch, Vehicle-to-vehicle communications with urban intersection path loss models, in: 2016 IEEE Globecom Workshops, GC Wkshps, IEEE, 2016, pp. 1–6.
8. Le Liang, Geoffrey Ye Li, Wei Xu, Resource allocation for D2D-enabled vehicular communications, IEEE Trans. Commun. 65 (7) (2017) 3186–3197.
9. Yalan Wu, Jigang Wu, Long Chen, Gangqiang Zhou, Jiaquan Yan, Fog computing model and efficient algorithms for directional vehicle mobility in vehicular network, IEEE Trans. Intell. Transp. Syst. (2020).
10. Harold W. Kuhn, The Hungarian method for the assignment problem, Nav. Res. Logist. Q. 2 (1–2) (1955) 83–97.
11. Zhenyu Zhou, Haijun Liao, Xiongwen Zhao, Bo Ai, Mohsen Guizani, Reliable task offloading for vehicular fog computing under information asymmetry and information uncertainty, IEEE Trans. Veh. Technol. 68 (9) (2019) 8322–8335.
12. Yuxuan Sun, Xueying Guo, Jinhui Song, Sheng Zhou, Zhiyuan Jiang, Xin Liu, Zhisheng Niu, Adaptive learning-based task offloading for vehicular edge computing systems, IEEE Trans. Veh. Technol. 68 (4) (2019) 3061–3074.