

A Greedy Dynamic Task Offloading and Service Management in V2V Networks

Hanyang Xing^a

*Institute of Communication Engineering, University of Electronic Science and Technology of China,
2006 Xiyuan Avenue, High-tech Zone (West), Chengdu, China*

Keywords: Vehicle-to-Vehicle Communication, Knapsack Algorithm, Game Theory, Decision Tree.


Abstract: This paper introduces an advanced framework for optimizing task offloading and service caching in Vehicle-to-Vehicle (V2V) communication networks. The proposed approach leverages a greedy algorithm to address key challenges such as offloading latency, energy consumption, and system overhead. By incorporating practical factors such as task size, server storage capacity, and task popularity, the framework efficiently allocates tasks, thereby reducing computational delays and enhancing network performance. The effectiveness of the algorithm is validated through comprehensive simulations that demonstrate significant improvements in both time efficiency and resource utilization compared to existing methodologies. The results underscore the potential for future advancements in V2V networks, particularly in enhancing network stability under high-speed conditions and developing robust communication systems that maximize the use of roadside computational resources.

1 INTRODUCTION

The computational demands for information transmission are rapidly increasing with the proliferation of the Internet of Things (IoT). While the existing cloud server infrastructure remains an efficient and cost-effective solution for handling most user requests, the growing number of local devices with stringent computational needs—including augmented reality, virtual reality, autonomous vehicles, and vehicle communication—has created new challenges. These applications require ultra-reliable and low-latency communications (URLLC) to ensure low latency, reduced power consumption, and adequate computational resources without compromising performance (Zeng, Zhang, Wang, Liu, Wang, 2020). In this context, Mobile Edge Computing (MEC) is becoming a critical component of modern networks, serving as a distributed computational infrastructure embedded within hosts (He, Li, Chen, Wang, 2019). Tasks can be offloaded to these edge servers, a process known as Mobile Edge Computing Offloading (MECO) (He et al, 2019), allowing highly demanding tasks to be processed and subsequently downloaded to local devices.

However, current URLLC devices and services face several significant challenges. Firstly, many communication networks are burdened with a substantial number of computation-intensive tasks, which strains the computational capacity of nodes or hosts. This strain, coupled with the network's intermittent availability, can result in inaccurate communication processes (Zeng et al, 2020). Secondly, the network often operates near full capacity, making immediate transmission and processing of data nearly impossible due to congestion (Zeng et al, 2020; He et al, 2019). Thirdly, although the existing capacity allocation algorithms within MEC networks are sophisticated, they fall short in accounting for long-term effectiveness, leading to suboptimal resource allocation over extended periods (He et al, 2019).

Vehicle-to-vehicle (V2V) communication, which is crucial for the advancement of driverless vehicles and the improvement of traffic conditions, is particularly affected by these challenges. V2V communication requires stable connections between vehicles, which function as both hosts and nodes while in motion (Liu, Wang Chen, Bensaou 2021). Current vehicular ad hoc networks rely on cellular

^a <https://orcid.org/0009-0008-5534-0930>

networks for direct data transfer. However, this approach is limited, often leading to inefficiencies and wasted computational resources within the network (Pachet, Chen, Chen, 2020).

To address these challenges and integrate mobile edge networks into vehicular systems, our research proposes a robust framework for task offloading and service caching within both edge and cloud networks. This framework is designed to reduce latency and energy consumption, thereby minimizing time delays and resource demands during information transmission. Based on computational capabilities, data packets will dynamically choose the most optimal processing route—whether locally, at the edge server, or within the cloud—correlating computational power with transmission time. The proposed framework incorporates a non-cooperative game theory-based algorithm to efficiently distribute packets across various endpoints. Additionally, considering the limited energy resources at the edge servers, the framework leverages a 0-1 knapsack algorithm to refine the initial selection process, implementing dynamic service caching based on task popularity (i.e., frequency of service requests) to ensure that the endpoint can handle the required data volume. The final offloading decision is derived from a comprehensive analysis of both service caching and the original selection.

1.1 Accessibility

Vehicle-to-Vehicle (V2V) communication is a form of Mobile Edge Computing (MEC) that leverages resources from vehicles and roadside infrastructure to collect and disseminate traffic information, thereby supporting self-driving vehicles and enhancing traffic conditions. Currently, V2V communication demands low latency and energy-efficient transmission, but existing algorithms struggle to fully meet these requirements. However, through the optimization and refinement of these algorithms, V2V communication can be made faster and more energy-efficient.

To improve the accessibility of V2V communication, task offloading should be prioritized based on the popularity or importance of the tasks, ensuring a more rational processing by the algorithm. Additionally, the algorithm must operate within reasonable computational requirements to reduce processing time and further minimize latency.

1.2 Inserting V2V Concepts

V2V communication, short for Vehicle-to-Vehicle communication, is a subset of Mobile Edge

Computing (MEC) that offers an alternative to Mobile Cloud Communication (MCC). An MEC system comprises not only a cloud center but also various edge devices, such as base stations and mobile phones. Tasks requiring computation are partitioned, with some processed locally while others, particularly those that are computation-intensive, are offloaded to MEC servers or cloud servers to utilize higher computational power. Key factors in determining the optimal task partitioning include the offloading ratio, CPU-cycle frequency, and transmission power. This approach results in lower latency, reduced energy consumption, and shorter transmission distances. Consequently, the system is transformed into an information-centric architecture capable of partitioning and offloading tasks more efficiently, rather than rigidly transferring all tasks from one end to another.

2 RELATED WORK

Vehicle-to-Vehicle (V2V) communication addressing specific traffic issues has been extensively studied in various contexts. Dey et al. explored wireless communication within a heterogeneous network (Het-Net) environment, utilizing Wi-Fi, DSRC, and LTE for the transmission of accident information between vehicles. Their research demonstrated that this system effectively reduces the dependency on infrastructure communication and establishes a stable connection between rapidly moving vehicles (Dey, Ding, Zheng, 2016). Navas et al. developed a device equipped with an adaptive cruise control (ACC) system that can be easily installed on vehicles to mitigate poor traffic conditions (Navas, Milanés, 2019). Notably, this device remains effective even when preceding vehicles are not equipped with it, enhancing its acceptability and widespread adoption (Navas et al, 2019).

In 2017, Perfecto et al. proposed a framework for beam alignment in millimeter-wave V2V networks, which improves millimeter-wave communication by addressing issues related to directionality, blockage, and alignment delay (Perfecto, Del Ser, Bennis 2017). This framework has proven effective under complex conditions in high-density, multi-lane highway scenarios (Perfecto et al, 2017). Ahmad et al. recommended a validation method for congestion control and performance in V2V systems through vehicle-level testing, which enables congestion detection within a 1.5-meter range and achieves a transmission latency of 600ms (Ahmad,

Bakhshizadeh, Yilmaz, 2019). Bian et al. introduced a linear feedback controller designed to reduce time headway in V2V communication under various conditions. Their approach includes a novel definition of the inter-vehicle distance, which helps prevent undesirable intermittent fluctuations in distance detection (Bian, Wu, Zheng 2019). By increasing the number of predecessors, their method ensures internal asymptotic stability and meets string stability requirements (Bian et al, 2019). Gao et al. proposed an enhanced GPSR-based wireless routing scheme that incorporates newly introduced parameters, such as a prediction mechanism and computed weights, to improve wireless communication stability (Gao, Zhao, Yin, 2021). Their approach demonstrated superior performance in terms of packet delivery ratio, wireless hops, and time delay (Gao et al, 2021). Bazzi et al. conducted a comparison between the IEEE 802.11 standard and the proposed V2V communication network, revealing that the proposed network outperforms IEEE 802.11 in terms of maximum awareness range and vehicle density, given identical inputs and external conditions (Bazzi, Wong, 2017).

3 ALGORITHMS

In this part, it shows the part of original code of the purposed algorithm. The principle of this part of code illustrates how the decision is made based on the task size and capacity requirement. In Q1, it is a 2-dimensional array which contains size and capacity requirement information about the task.

In this section, we present a segment of the original code implementing the proposed algorithm. This code illustrates the core decision-making mechanism, which is based on the task size and the corresponding capacity requirements. The algorithm operates on a two-dimensional array, denoted as Q1, which encapsulates the size and capacity requirements of the tasks. The iterative nature of the algorithm allows for dynamic adjustments to be made as tasks are processed.

To evaluate the proposed framework, we consider a V2V communication system comprising vehicles and roadside edge servers. Vehicles are assumed to move at varying speeds with densities ranging from sparse rural areas to dense urban environments. In this model, each vehicle generates tasks with specific computational requirements, including CPU cycles and storage needs. Edge servers are equipped limited storage and processing capacities which influencing task offloading decisions. Moreover, we built a traffic

model simulates real-world scenarios where vehicles dynamically interact with edge servers, enabling comprehensive analysis of system performance.

Our proposed algorithm incorporates three main modules: Task Offloading, Dynamic Service Caching, and Adjustment of Task Offloading. These modules work together to optimize computational resource utilization. The algorithm begins by evaluating task size and capacity requirements through a two-dimensional array. Probabilities are calculated to determine task selection, and a random selection mechanism ensures balanced resource allocation.

As Algorithm 1 shown, this algorithm begins by resetting the two-dimensional array Q1 through the Restart function, ensuring that all task-related data is cleared before processing. The main decision-making process occurs in the UpdateAction1 function, where tasks are evaluated based on their size and capacity requirements stored in Q1.

Probabilities are calculated to determine which tasks are likely to be selected for execution, and a random selection mechanism is employed to choose the next task, ensuring a balanced distribution of computational resources. This approach enables the algorithm to adapt dynamically to varying task demands, optimizing performance in a V2V communication environment.

```

Data: task size and capacity requirements
Result: how to select the next task;
while not iterate over all the tasks do
    read the task index;
    if the loaded task is not current task then
        calculate the resource ratio;
        compute the probability for task selection
        update the probability list
    else
        select a new task
    end
end

```

Algorithm 1: How to select a new task

4 METHODOLOGY

This section outlines the methodology of the proposed framework, including the algorithm's processing flow and structure, which encompasses:

1. Task Offloading: Using non-cooperative game theory to minimize latency.
2. Dynamic Service Caching: Employing the 0-1 knapsack algorithm to cache tasks based on popularity and storage capacity.

3. Adjustment of Task Offloading: Refining initial decisions to optimize resource utilization.

4.1 Algorithm Process

The principles and procedures of the framework are briefly introduced below with corresponding diagrams: The process begins with tasks that are candidates for offloading to an edge server. The IoT device first identifies and confirms the edge server before initiating task transmission. Upon receiving the notification, the edge server evaluates the service request based on its frequency of occurrence. A higher frequency increases the likelihood that the task will be offloaded to that particular edge server. The edge server then downloads services from the cloud server to cache them, considering both the popularity of the services and the storage limitations of the edge server. If the requested service for a task is already cached in the edge server, the task can be offloaded to it. Otherwise, the task will need to be offloaded to another endpoint.

4.2 Algorithm Structure

The algorithm first identifies the optimal terminal for task offloading and then caches the task accordingly. The objective is to minimize energy consumption while maximizing task popularity, using the 0-1 knapsack algorithm and incorporating game theory. Based on these factors, the algorithm provides the most effective strategy for task offloading.

4.2.1 Algorithm Explanation

In this algorithm, five IoT devices are assumed to handle local processing. Additionally, both edge servers and cloud servers are available for task offloading. The values 0, 1, and -1 represent local processing, edge server processing, and cloud server processing, respectively, for each task.

The algorithm is composed of three main modules: Task Offloading, Dynamic Service Caching, and Adjustment of Task Offloading. The Task Offloading module employs a non-cooperative game theory approach to make initial task offloading decisions, aiming to minimize system overhead. The Dynamic Service Caching module uses the 0-1 knapsack algorithm to maximize the popularity of task-requested services by implementing dynamic service caching in edge servers. Finally, the Adjustment of Task Offloading module fine-tunes the initial

offloading decisions based on the caching outcomes at the edge server.

The following algorithms are used for comparative experiments:

- The TOCS algorithm, which comprehensively addresses task offloading and service caching within a mobile edge computing network without relying on cloud server assistance.
- The TO algorithm, which focuses solely on the offloading problem within a mobile edge computing network.
- The LP algorithm, where all tasks are processed on local IoT devices.

5 EXPERIMENT

This section presents two experiments designed to validate the algorithm under different conditions and scenarios. It begins with an introduction to the parameters and key components of the algorithm and the experimental setup. Following this, the specific steps of the experiments are detailed, along with an analysis of the results.

Two experiments were conducted to evaluate the proposed framework under different conditions. Key parameters include CPU capacity, task size, and server storage. The experiments simulate task generation and offloading in both sparse and dense traffic scenarios.

5.1 Experiment Introduction

This subsection describes two experiments conducted under varying conditions, along with the primary results. Figure 1 illustrates the algorithm's flowchart and provides an overview of the experimental procedure. Initially, devices and vehicles request services from nearby edge servers, such as base stations. These tasks are subsequently aggregated

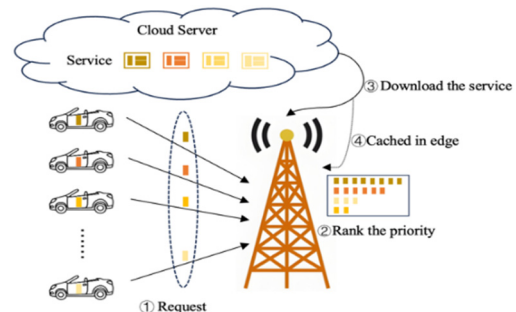


Figure 1: Flowchart of the Algorithm.

at the base station, where they are ranked based on their popularity, size, and the storage capacity of the base station. The algorithm subsequently generates a decision based on this ranking. The appropriate services are then downloaded from the cloud server and processed on different hosts, ensuring that tasks are executed where energy consumption and processing time are minimized.

The parameters used in the experiments are listed in Table 1 below. Each parameter plays a critical role in influencing task offloading latency. For example, the Task CPU parameter affects the processing time and directly impacts the task offloading decision, especially since the edge server may not provide sufficient computational capacity for certain tasks.

The Task Size parameter indicates the storage requirement for hosting the task. Task Popularity reflects the importance of each task, determining which tasks should be prioritized in the scheduling process.

Table 1: Factors to select the best server of task.

Factor Name	Definition
Task CPU	Required CPU capacity for task
Task Size	Required server size for storing
Task Popularity	Importance of task
Server CPU	CPU capacity offered by server
Server Storage	Storage size offered by server

5.2 The Capacity Experiment

In the first experiment, the algorithm's performance is evaluated by varying the CPU capacity for task requests. By adjusting the CPU value, different task offloading decisions are observed. Here, 3 types of extant method, that are introduced in chapter 4.2.1, are imported to compare with the proposed algorithm.

Initially, the algorithm is run to determine a selection that minimizes system overhead. In this phase, the capacities of the cloud server, edge server, and local devices are predefined. Here, "Task CPU" refers to the required processing capacity, while "Task Size" indicates the storage requirement for each task. The algorithm employs a non-cooperative game theory approach, making decisions based on single-oriented connections. To assess the impact of CPU requirements, the fifth task is treated as a variable, with its required CPU capacity adjusted from 0.1GHz to 1GHz.

After iterating the selection process three times, the resulting task allocation is 0-0-1-0-0, indicating that the third task is offloaded to the edge server while the others are processed locally.

Following the initial result, the algorithm considers task popularity and the storage capacity of edge servers in the Dynamic Services Caching phase. The outcome of this phase is 0-1-1-0-0, meaning that tasks 2 and 3 cache their services on the edge server, while the others do not.

Finally, combining the initial result with the caching outcome, the final task offloading decision is made, yielding the selection 0-0-1-0-0, where only the third task is offloaded to the edge server, and the remaining tasks are processed locally. As depicted in Figure 2, when the CPU requirement is not excessively high, the algorithm achieves the lowest task offloading cost compared to other methods. However, when the CPU requirement exceeds a certain threshold, the cost becomes constrained by the limited storage capacity of the edge server, leading to no further cost reduction.

Table 2: Factors to select the best server of task.

Method Storage	Propose	TOSC	TO	LP
0.1	3.75602	3.75602	5.0203	6.48231
0.2	4.04102	4.04102	5.3053	6.76731
0.3	4.32602	4.32602	5.5903	7.05231
0.4	4.38102	4.61102	5.8753	7.33731
0.5	4.38102	4.89602	6.1603	7.62231
0.6	4.38102	5.18102	6.4453	7.90731
0.7	4.38102	5.46602	6.7303	8.19231
0.8	4.38102	5.75102	7.0153	8.47731
0.9	4.38102	6.03602	7.3003	8.76231
1.0	4.38102	6.32102	7.5853	9.04731

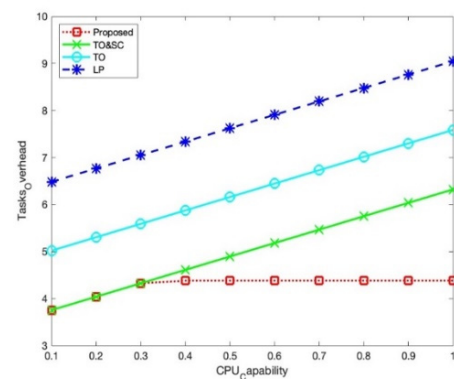


Figure 2: The cost of the task offloading with the change of CPU requirement.

5.3 The Storage Experiment

In the second experiment, the performance of the algorithm is assessed by varying the storage capacity of the edge servers. The process for determining the final task offloading decision follows the same steps as in the first experiment, but with a key difference: the storage capacity of the edge servers is altered in each iteration. This adjustment is crucial, as the storage capacity directly impacts the number of tasks that can be offloaded, as well as the popularity of those tasks within the network.

For this experiment, the storage unit is set to 3, meaning that each edge server has the capacity to accommodate three tasks. Each task has a uniform size of 1, while their respective values are assigned as 1, 2, 0, 0, and 1. These values represent the importance or priority of the tasks in the system, which influences the offloading decisions.

The experiment begins with the execution of the Dynamic Service Caching algorithm. This step determines which tasks should be cached on the edge servers based on their popularity and the available storage space. The outcome of this step guides the subsequent task offloading process by ensuring that high-priority tasks are positioned where they can be processed most efficiently.

Following the caching decisions, the Adjustment of Task Offloading phase is conducted. In this phase, the algorithm refines the initial offloading decisions, taking into account the caching results and the storage constraints of the edge servers. The final offloading decisions, under these conditions, are determined to be 0-0-0-1-1. This result indicates that the last two tasks will be offloaded to the edge server, while the first three tasks will remain for local processing.

The experiment also explores how different storage conditions, combined with a fixed CPU capacity of 0.3, influence the overall cost associated with each offloading decision. The analysis reveals that as the storage capacity and task values vary, the cost of offloading decisions fluctuates accordingly. This relationship underscores the importance of optimizing both storage and processing capacities to minimize costs and maximize efficiency in the V2V communication system.

This experiment highlights the critical role of edge server storage in the task offloading process. By adjusting storage capacity and analyzing its impact on task allocation, the algorithm demonstrates its ability to adapt to varying network conditions, thereby ensuring efficient resource utilization and optimal performance across different scenarios.

Table 3: Factors to select the best server of task.

Storage Methods \	1	2	3	4
Proposed	7.68642	7.59442	7.40793	7.40793
TOSC	8.8587	7.59442	7.40793	7.40793
TO	8.86992	8.8587	8.6722	7.40793
LP	10.3207	10.3207	10.3207	10.3207

5.4 Analysis

In the first experiment, the results, as illustrated in Figure 2, demonstrate the impact of varying CPU capacity in the edge server using different data transfer methods. The proposed algorithm consistently achieves the lowest transmission cost compared to a singular processing method. Furthermore, the cost of task offloading peaks when the CPU capacity is sufficiently high, as the CPU can easily handle the tasks without requiring additional resources.

In the second experiment, the results, presented in Figure 3, focus on the influence and effectiveness of server storage. Under identical storage conditions, the proposed algorithm outperforms other algorithms by minimizing the cost of task offloading. Similar to the first experiment, when the storage capacity is sufficiently large, the cost stabilizes because the tasks no longer fully occupy the available server space.

These experiments collectively demonstrate the validity and effectiveness of the proposed algorithm in optimizing task offloading in V2V communication systems. The algorithm's ability to minimize costs while effectively utilizing CPU capacity and server storage underscores its potential for practical applications in edge computing environments.

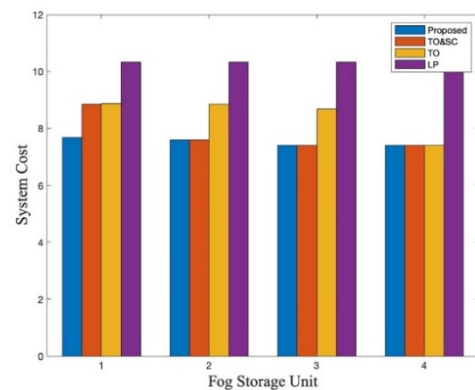


Figure 3: The cost of the task offloading with the change of server storage.

However, sufficient edge computing capacity is currently a challenge and the relative movement between vehicles and vehicles causes the challenge of stable information connection and further influences the entire communication network of V2V.

6 CONCLUSIONS

This paper presents an effective algorithm and framework for task offloading and service downloading, inspired by multi-predecessor investigations and research. The algorithm considers various factors, including task size, server storage, and task popularity, to optimize task offloading decisions. These considerations allow for intelligent service downloading decisions based on the specific conditions of the network. As demonstrated through experimental simulation and validation, the proposed algorithm can effectively minimize both time delays and energy consumption by making informed, strategic decisions.

Looking ahead, further improvements could be made to enhance the stability of V2V networks, particularly in scenarios where vehicles are moving at high speeds. Additionally, the development of vehicle-to-infrastructure communication holds promise for fully leveraging the computational resources of roadside infrastructures.

REFERENCES

- Zeng, J., Zhang, H., Wang, H., Liu, Y., & Wang, W. (2020). *Mobile edge communications, computing, and caching (MEC3) technology in the maritime communication network*. *China Communications*, 17(5), 223-234.
- He, Y., Li, Y., Chen, Y., & Wang, H. (2019). *D2D communications meet mobile edge computing for enhanced computation capacity in cellular networks*. *IEEE Transactions on Wireless Communications*, 18(3), 1750-1763.
- Liu, L., Wang, T., Chen, Y., & Bensaou, B. (2021). *Vehicular edge computing and networking: A survey*. *Mobile Networks and Applications*, 26(3), 1145-1168.
- Pachatz, J., Chen, Y., & Chen, Y. (2020). *Index coding in vehicle to vehicle communication*. *IEEE Transactions on Vehicular Technology*, 69(10), 11926-11936.
- Dey, K. C., Ding, Y., & Zheng, Y. (2016). *Vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication in a heterogeneous wireless network – Performance evaluation*. *Transportation Research Part C: Emerging Technologies*, 68, 168-184.
- Navas, F., & Milanés, V. (2019). *Mixing V2V- and non-V2V-equipped vehicles in car following*. *Transportation Research Part C: Emerging Technologies*, 108, 167-181.
- Perfecto, C., Del Ser, J., & Bennis, M. (2017). *Millimeter-wave V2V communications: Distributed association and beam alignment*. *IEEE Journal on Selected Areas in Communications*, 35(9), 2148-2162.
- Ahmad, S. A., Bakhshizadeh, A., & Yilmaz, M. (2019). *V2V system congestion control validation and performance*. *IEEE Transactions on Vehicular Technology*, 68(3), 2102-2110.
- Bian, Y., Wu, C., & Zheng, Y. (2019). *Reducing time headway for platooning of connected vehicles via V2V communication*. *Transportation Research Part C: Emerging Technologies*, 102, 87-105.
- Gao, H., Zhao, J., & Yin, Y. (2021). *V2VR: Reliable hybrid-network-oriented V2V data transmission and routing considering RSUs and connectivity probability*. *IEEE Transactions on Intelligent Transportation Systems*, 22(6), 3533-3546.
- Bazzi, A., & Wong, K. W. (2017). *On the performance of IEEE 802.11p and LTE-V2V for the cooperative awareness of connected vehicles*. *IEEE Transactions on Vehicular Technology*, 66(11), 10419-10432.