VECMAN: A Framework for Energy-Aware Resource Management in Vehicular Edge Computing Systems

Tayebeh Bahreini, *Student Member, IEEE*, Marco Brocanelli, *Member, IEEE* and Daniel Grosu, *Senior Member, IEEE*

Abstract—In Vehicular Edge Computing (VEC) systems, the computing resources of connected Electric Vehicles (EV) are used to fulfill the low-latency computation requirements of vehicles. However, local execution of heavy workloads may drain a considerable amount of energy in EVs. One promising way to improve the energy efficiency is to share and coordinate computing resources among connected EVs. However, the uncertainties in the future location of vehicles make it hard to decide which vehicles participate in resource sharing and how long they share their resources so that all participants benefit from resource sharing. In this paper, we propose VECMAN, a framework for energy-aware resource management in VEC systems composed of two algorithms: (i) a resource selector algorithm that determines the participating vehicles and the duration of resource sharing period; and (ii) an energy manager algorithm that manages computing resources of the participating vehicles with the aim of minimizing the computational energy consumption. We evaluate the proposed algorithms and show that they considerably reduce the vehicles' computational energy consumption compared to the state-of-the-art baselines. Specifically, our algorithms achieve between 7% and 18% energy savings compared to a baseline that executes workload locally and an average of 13% energy savings compared to a baseline that offloads vehicles' workloads to RSUs.

Index Terms—Vehicular edge computing, Resource management, Chance constrained optimization.

1 Introduction

Electric Connected Autonomous Vehicles (eCAVs), the future of our transportation system, have two main requirements: processing massive amount of data with minimum latency and having long driving ranges. To address the first requirement computational nodes must be placed closer to the eCAVs at the *edge* of the cloud [1]. Thus, in these so called *Vehicular Edge Computing (VEC)* systems, computational nodes are placed in Road-Side Units (RSUs) and exploit the Dedicated Short Range Communication (DSRC) technology [2] for vehicle-to-vehicle and vehicle-to-RSU communications. However, the scalability of these systems may be hindered by the amount of vehicles and workloads in RSU coverage areas. Indeed, due to the limited capacity of RSUs, some computational tasks may experience poor Quality of Service (QoS) or even failure. In order to improve reliability, computing resources (e.g., Nvidia Drive Px 2) are installed on each vehicle for local workload execution and minimized latency. On the other hand, having local computing resources might affect the eCAV's driving range. For example, a preliminary study by Lin et al. [3] shows that a computing node (including computing, storage, and cooling hardware) can reduce the driving range of a Chevy Bolt by up to 11.5%. However, the authors only considered three executing tasks on various computing configurations and did not consider the effect of the eCAV speed on the driving range. To conduct a more general analysis, we exploit the driving range vs speed data available for the Tesla Model S [4], which can be used to easily estimate the total power consumption of the eCAV at different speeds given the battery pack size. Figure 1 shows the results for the addition of a computing system of power consumption varying from 0W to 3kW. We observe that the impact of the computing power on the driving range varies from 10% to 40% for a 2kW computing power. This is computed for the eCAV traveling at average speeds from 60mph to 20mph, respectively. Considering that the rush-hour (i.e., 4-5pm) speed in 20 US cities is 33mph on average [5], a 2kW computing system can cause a 25% reduction in driving range on modern electric vehicles. As a result, because most eCAVs on the road during weekdays travel at rush hours and because during this period eCAVs spend a considerable amount of time near each other due to traffic, it is desired to design new strategies that allow to reduce the energy consumption of eCAVs computing systems for longer driving ranges.

In order to achieve high QoS, reliability, and increased driving range, eCAVs in a VEC system could share their resources. Rather than relying only on the limited capacity of RSU nodes, the computing resources of the vehicles can be coordinated to achieve energy savings. The key-intuition for achieving energy savings is to 1) exploit the *computational slack* caused by discrete power/QoS settings of computing resources (e.g., CPUs and GPUs), and 2) exploit the non-linear relationship between these settings and the computational power consumption [6]. First, CPUs commonly have a fixed number of selectable configurations for voltage-frequency levels and number of cores to trade-off power

T. Bahreini, M. Brocanelli, and D. Grosu are with the Department of Computer Science, Wayne State University, Detroit, MI 48202.
E-mail: tayebeh.bahreini@wayne.edu, brok@wayne.edu, dgrosu@wayne.edu

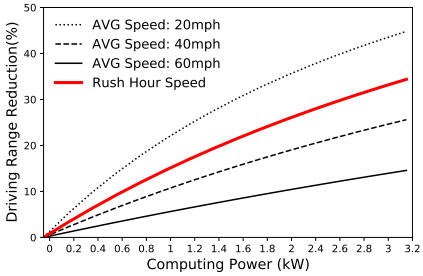


Fig. 1. Effect of computing power on eCAV's driving range.

consumption and QoS. Each configuration leads to a maximum number of instructions that can be executed within a certain time period. If the local workload exceeds that maximum, the system must select a new configuration that increases the QoS at the cost of a higher power consumption, e.g., activate more cores or increase the voltage-frequency level. However, this selected *default* configuration may not be fully utilized by the local workload, i.e., there exist a slack of computational capacity that can be used to execute extra workload at the same default configuration. Second, there exist a non-linear relationship between the computational power consumption and the frequency settings [6]. Thus, a taskset executing on an eCAV operating at a high CPU frequency may consume much more power than the same taskset partially executing on an eCAV operating at a lower CPU frequency with similar overall performance. Thus, the system can achieve energy savings by offloading part of *requester* vehicle workload (currently operating at a high frequency) to *provider* vehicles (currently operating at a low frequency) that can exploit the providers' computational slack without changing their default configuration, which limits the provider vehicles power overhead. Then, at a later time, providers can become requesters to achieve higher energy savings.

Enabling resource sharing among eCAVs in a VEC system requires the dynamic determination of two important parameters: (a) the set of participating vehicles in resource sharing, and (b) the time duration of resource sharing. However, the uncertainties in the future location of vehicles make it hard to decide these parameters so that all participants benefit from resource sharing. In this paper, we overcome these challenges by proposing VECMAN, a framework for energy-aware resource management, which consists of two algorithms: (i) a resource selector algorithm that runs periodically on the local RSU and determines the set of participating vehicles as well as the duration of resource sharing; and (ii) an energy manager algorithm that runs periodically at a finer time grain within each resource sharing time period and determines the state of each participating vehicle, i.e., requester or provider, the number of replicas for the requesters' workloads, and the amount of requester's workload to offload so that energy consumption is minimized for all the participating vehicles.

Main Contributions. This paper is an extended version of our previous work on energy-aware resource management in VEC systems [7], where we have assumed to already have determined the set of participating vehicles and the length of the sharing period for the energy manager. In addition, the system has only been tested on a randomly-generated vehicle motion trace. Different from our previous work and other related work extensively described in Section 2, in this paper we provide a comprehensive energy-aware resource management framework for eCAVs in VEC systems. Specifically, our main contributions are as follows:

- In Section 3 we formulate the new resource selection problem and generalize the energy manager formulation for a complete integration in the VECMAN framework. The objective of the resource selector is to dynamically determine the set of participating vehicles and the duration of the resource sharing period for maximized computing resources and reliability. The VECMAN framework allows partial offloading of vehicles' workload.
- VECMAN is robust to uncertainties in the future location of vehicles. However, the optimization problem solved by VECMAN is a chance-constrained problem, and thus, obtaining the optimal solution in a reasonable amount of time is not freasible. Therefore, in Section 4, VECMAN is designed based on an iterative algorithm called I-Selector to solve the resource selection problem, and a greedy algorithm called G-ERMP to solve the energy-aware resource management problem.
- In Section 5, we test VECMAN using a real-world dataset of vehicular mobility collected in the city of Cologne, Germany [8]. The results show that VECMAN achieves between 7% and 18% energy savings compared to a baseline that executes workload locally, and an average of 13% energy savings compared to a baseline that offloads workloads only to RSUs.

2 Related Work

A highly efficient offloading mechanism for VEC systems has to overcome several challenges that do not exist or are less significant in the cloud-based systems. Due to the dynamic nature of VEC systems that mainly stems from the mobility of vehicles, an optimal offloading for the current setting of the network might turn to the least efficient offloading in few seconds/minutes. Furthermore, the variability of the available resources over time affects two important performance metrics, i.e., the reliability of the network and the QoS.

High QoS. Some of the existing challenges of resource management in VEC systems have been addressed from different perspectives and using different approaches. Several studies focused on ensuring high QoS. Zheng et al. [9] addressed the variability of the resources in a system in which clouds, parked vehicles, and mobile vehicles provide computational services. In their system, the goal is to maximize the amount of power saving of vehicles while the latency and transfer costs are minimized. Yu et al. [1] proposed a hierarchical VM migration mechanism for mobile vehicles by integrating computing resources in data centers, RSUs, and vehicles. They presented a layered structure that allows vehicles to select their services resiliently. Yang et al. [10] proposed a task offloading scheme for cooperative edge servers scenario with the aim of minimizing the cost of offloading while guaranteeing the QoS. However, these studies considered integrated edge resources when making

offloading decisions and did not consider the distributed and mobile nature of servers.

High Reliability. Some researchers have also considered speculative execution as a technique to ensure high QoS and low risk of failure. Zhiyuan et al. [11] studied the task replication problem to minimize the probability of deadline violations. Zhu et al. [12] proposed an online algorithm for task replication in vehicular fog computing with the aim of minimizing the maximum service latency while minimizing the total quality loss of tasks. Sun et al. [13] proposed a learning based algorithm for task replication in VEC systems. Their objective is to minimize the average offloading delay of tasks. Zhou et al. [14] considered a cooperative vehicle infrastructure system and addressed the execution time minimization problem. Bahreini et al. [15] developed a speculative execution framework for VEC systems, where the number of replicas for each requests is predetermined and is given by an energy manager. The replica manager decides how to allocate copies of the vehicles' workload on different nodes to ensure high reliability and low latency. Hou et al. [16] developed a computation offloading mechanism for latency-sensitive applications. To ensure high reliability, they jointly optimized task allocation and the reprocessing of failed subtask executions. However, these studies do not balance the number of replicas with the energy consumption: having a high number of replicas may lead to a small improvement in robustness to failure while causing energy waste on vehicles.

QoS vs. Energy. Several solutions have been proposed to trade off between latency and energy consumption in mobile edge computing systems [17], [18], [19], [20], [21], [22], [23]. These studies investigate how to account for the interference of multiple service requesters sharing service providers with consideration of latency and energy consumption. In particular, Viswanathan et al. [22] collected statistics on availability of connections between service requesters and service providers to improve the QoS. Workload is then migrated across service providers to handle unexpected scenarios (e.g., loss of connectivity or empty battery). However, all the above solutions have a single point of failure, i.e., if the selected service provider fails, the workload has to be migrated or offloaded again, which increases the latency. Note that some of the studies mentioned above (e.g., [1], [12], [13]) use task replication to reduce the risk of failure but without consideration of the impact of the number of replicas on the energy consumption.

To the best of our knowledge, different from our solution, none of the above studies consider at the same time the problems of (a) determining the duration of resource sharing, (b) coordinating the computing resources among moving vehicles, and (c) determining the number of replicas for vehicular workloads to minimize the energy consumption of vehicles without violating the desired QoS levels.

3 VECMAN Problem Formulation

VECMAN is characterized by two main components, the *resource selector* and the *energy manager*. The resource selector takes the vehicles' information (i.e., location and computing resources) into consideration periodically (at a coarse time scale) and, based on the history of the traffic data in the coverage area and the movement behavior of vehicles, (a) it selects the set of vehicles that can participate in resource sharing, (b) it determines the length of the period so that each selected vehicle remains in the coverage area with a high probability during the resource selection period. We assume that the real-time location information of a vehicle is obtained by the on-board global position system (GPS). The resource selector, based on the history of traffic data and the current location of a vehicle, determines the probability that the vehicle stays in the coverage area to decide whether or not to engage that vehicle in the resource sharing. At a finer time scale than the resource selector, the energy manager, decides the vehicles' state (i.e., requester or provider), the number of replicas for each requester's workload, and the replicas' allocation on provider vehicles. The resource selector and the energy manager run at different time scales, and thus the problems they solve can be treated as separate.

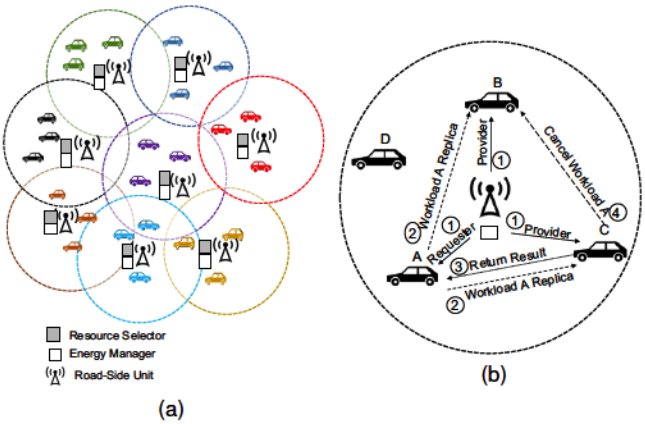


Fig. 2. A VEC system: (a) An area with several vehicles and RSUs, (b) Each RSU runs the resource selector and the energy manager locally.

3.1 Illustrative Example

Figure 2a shows the distributed architecture of a VEC system implementing VECMAN. Each vehicle is assigned to the nearest RSU. Each RSU runs the resource selector and the energy manager locally. Figure 2b shows the coverage area of an RSU, with four assigned vehicles. In this example, the resource selector selects vehicles A, B, and C to participate in resource sharing for T units of time. Due to the high probability of leaving the coverage area, vehicle D is not selected to participate in resource sharing. At a finer grain, the energy manager is executed on the selected vehicles for T/T_{em} periods, where T_{em} is the length of each energy manager period. In the first period, the energy manager (1) selects vehicle A as a service requester and vehicles B and C as service providers. It also determines two replicas for the request of vehicle A to be assigned on vehicles B and C. (2) The energy manager coordinates the replicas deployment. (3) When one of the providers returns the computation result to A (e.g., vehicle C), (4) the other providers stop the computation and wait for other workloads. In the following energy manager periods, vehicle A will host the workloads of vehicles B and C to help them lower their energy consumption. Thus, all the vehicles can save energy over the resource selector period T.

In the following, we formulate the resource selection problem (RSP) and the energy-aware resource management problem (ERMP) in VEC systems.

3.2 RSP Formulation

The inputs of the resource selector are a fixed set V of vehicles that are currently located in the coverage area, their available computing resources, and the historical information about locations of vehicles until the current time. Based on these parameters, the resource selector determines (a) the length of resource selection period T, and (b) the subset of vehicles $\mathcal{V}$ that with a risk factor less than α stay in the coverage area for T units of time. In fact, the goal of the resource selector is to engage as many computing resources of vehicles as possible but at the same time guarantee that each vehicle, by participating in sharing computing resources, can save some energy before the next invocation of the resource selector algorithm. At that point, a new period length is determined based on the current vehicle motion information.

We consider the number of instructions that can be executed by the CPU in a resource selection period as the amount of computing resources of a vehicle. Thus, increasing the length of the resource selection period increases the amount of computing resources of each vehicle. However, increasing the length of the period may decrease the probability that the vehicle remains in the RSU coverage area, which leads to a lower expected value of the total computing resources. Thus, to maximize the amount of computing resources, the resource selector should consider a trade off between the length of the period and the probability that the vehicles stay in the coverage area during the resource selector period. Thus, the *objective of the resource selector* is to find a subset of vehicles $\mathcal{V}$ over all possible subsets of vehicles Λ so that the total amount of computing resources for $T_{\mathcal{V}}$ units of time is maximized:

$$\max_{\mathcal{V}\in\Lambda} \sum_{i\in\mathcal{V}} M_i \cdot T_{\mathcal{V}} \tag{1}$$

where M_i is the Millions Instructions Per Second (MIPS) for the CPU of vehicle i, and $T_{\mathcal{V}}$ is the maximum number of units of time that vehicles in $\mathcal{V}$ stay in the coverage area with a risk factor less than α. In other words, $T_{\mathcal{V}}$ is the maximum number of units of time that satisfies the following risk factor constraint for all vehicles in $\mathcal{V}$:

$$p\left\{\bigcap_{t=1}^{T_{\mathcal{V}}}(\ell_i^t \le \rho)\right\} \ge 1-\alpha, \;\; \forall i \in \mathcal{V} \tag{2}$$

where ρ is the coverage range of the RSU and ℓ_i^t is a probabilistic parameter indicating the distance of vehicle i from the RSU at time unit t. The probability distribution of ℓ_i^t is obtained based on the possible scenarios for the location of the vehicle in time unit t. Note that, to increase the readability, in the rest of the paper, we use T instead of $T_{\mathcal{V}}$ when we refer to the currently selected time duration of the resource selector.

3.3 ERMP Formulation

The inputs of the energy manager are T, the length of resource selector period, $\mathcal{V}$, a fixed set of vehicles that with a risk factor less than α stay in the coverage area, the history of vehicle locations until the current time, and the workload characteristics. We assume that the RSU runs the energy manager for F_e periods within T units of time. The length of each period is fixed and is denoted by T_{em}, which can be set to a value that satisfies the Quality of Service (QoS) for the workloads execution, i.e., every workload should complete its execution within T_{em} units of time. Thus, the energy manager, at each one of the F_e invocations, decides the vehicle's state and the number of replicas for each selected requester so that, after F_e periods, the vehicles' energy consumption is minimized without violating the QoS requirements. In other words, we want to ensure that each vehicle, by participating in sharing computing resources, can save some energy before the end of the current resource selector period. Next, we identify two important constraints, i.e., limited capacity and risk factor, define the energy consumption model of the vehicles, and finally, formulate the ERMP.

Capacity Constraints. We characterize the capacity of vehicle i by $C_i = \{C_{1i}, \ldots, C_{Qi}\}$, where Q is the number of resource types. We consider three resource types (i.e., $Q = 3$) indexed by h: CPU ($h = 1$), memory ($h = 2$), and storage ($h = 3$). Each vehicle i has a limited capacity C_{hi} for each resource type h. In addition, we characterize the workload of vehicle i by $r_i = \{r_{1i}, \ldots, r_{Qi}\}$, where r_{hi} is the amount of resource of type h needed to complete the execution of the workload. Thus, r_{1i} is the number of CPU instructions (in millions), r_{2i} is the amount of memory, and r_{3i} is the amount of storage needed by the workload of vehicle i. For each vehicle i selected as a provider, the total amount of resources requested cannot exceed its available capacity:

$$\sum_{j\in S_i} {r'}_{hj} \le C_{hi}, \quad \forall i \in P, \forall h \tag{3}$$

where ${r'}_{hj}$ is the amount of resource of type h of vehicle j partially offloaded to provider i (${r'}_{hj} \le r_{hj}$), P is the set of providers, S_i is the set of vehicles that have a replica of their workload assigned to vehicle i, and C_{hi} is the available capacity of vehicle i for resource type h during each T_{em} units of time.

A challenge to overcome is how to calculate the CPU capacity C_{1i} in Constraint (3). We define the CPU capacity based on the Millions of Instructions (M_i) that can be executed within an energy manager period minus the number of CPU instructions required for the local workload:

$$C_{1i} = M_i \cdot T_{em} - r_{1i} \tag{4}$$

For a simple single-core CPU, the value of M_i is approximated by a function of the CPU frequency as follows:

$$M_i = \vartheta_i \cdot f_i + \theta_i \tag{5}$$

where ϑ_i and θ_i are estimated parameters, and f_i is the CPU frequency of each CPU. In order to ensure a good QoS, the required time to run the local workload r_{1i} must be shorter than the energy manager period duration T_{em}, which can be set as desired:

$$\frac{r_{1i}}{\vartheta_i \cdot f_i + \theta_i} \le T_{em} \tag{6}$$

Given the discrete frequency levels available in every CPU, each vehicle i, to achieve the required QoS at minimum energy consumption, must set (as default) the minimum frequency level f_i that satisfies the following constraint:

$$f_i \geq \frac{r_{1i}}{\vartheta_i \cdot T_{em}} - \frac{\theta_i}{\vartheta_i} \tag{7}$$

As a result, the total CPU capacity C_{1i} in Equation (3) can be calculated as follows:

$$C_{1i} = (\vartheta_i \cdot f_i + \theta_i) \cdot T_{em} - r_{1i} \tag{8}$$

As a result, the capacity constraints in Equation (3) enable the energy manager to place extra workload on provider vehicles without affecting their default CPU frequency.

Risk Factor Constraint. Despite the resource selector efforts to provide a fixed set of vehicles within each resource selector period, it may happen that some vehicles change their location in any of the F_e energy manager periods. Thus, because the future vehicle locations can only be predicted, we need to ensure that, with some level of confidence defined by a risk factor, each requester has a good connection with at least one provider during each period. To formulate this constraint, we first need to find the minimum distance between each requester and providers. In particular, for every selected requester, we want to have at least one provider within a reliable distance $\delta > 0$. On the other hand, the location of vehicles is non-deterministic and thus it may be affected by estimation errors. As a result, we must make sure that the probability of having at least one provider within a reliable distance is greater than a satisfaction factor $(1 - \beta)$. This constraint can be expressed as follows:

$$p\left\{\min_{j \in P | i \in S_j} l_{ij} \leq \delta\right\} \geq 1 - \beta, \quad \forall i \in \mathcal{V} \setminus P \tag{9}$$

where l_{ij} is the average distance between vehicle i and vehicle j in the current period, and P is the set of providers.

Energy Consumption Model. The computing system energy consumption for vehicle i is mainly characterized by two components, i.e., the computational and the transmission energy consumption. The computational energy consumption includes dynamic energy consumption and the idle energy consumption. The idle energy consumption is the basic power consumption in T_{em} units of time. The dynamic energy consumption is the power consumption for executing the requests which is proportional to the execution time of the requests and the third power of the CPU frequency [24]. According to Equation (6) and assuming that that all instructions are running at the same frequency, the execution time of request r'_j on provider i is $\frac{r'_{1j}}{\vartheta_i \cdot f_i + \theta_i}$. Thus, the extra energy consumption by executing request r'_j on provider i is $\lambda_i \cdot \frac{r'_{1j}}{\vartheta_i \cdot f_i + \theta_i} \cdot f_i^3$, where λ_i is an estimated parameter. Thus, the total extra computational energy consumption on provider i is:

$$E_i^{extra} = \lambda_i \cdot f_i^3 \cdot \frac{\sum_{j \in S_i} r'_{1j}}{\vartheta_i \cdot f_i + \theta_i}, \quad \forall i \in P \tag{10}$$

On the other hand, if vehicle i is selected as a requester, by offloading r'_i, the default frequency of vehicle i may change from f_i to f'_i, with $f_i \geq f'_i$. Thus, the energy saving of the vehicle is equivalent to:

$$E_i^{save} = \lambda_i \cdot f_i^3 \cdot \frac{r_{1i}}{\vartheta_j \cdot f_i + \theta_i} - \lambda_i \cdot f'^3_i \cdot \frac{r_{1i} - r'_{1i}}{\vartheta_i \cdot f'_i + \theta_i}, \quad \forall i \in \mathcal{V} \setminus P \tag{11}$$

The transmission energy consumption is proportional to the transmission latency which depends on the ratio between the request size and the data rate between the requester and the provider [25]. On the other hand, the data rate is proportional to the bandwidth between the requester and the provider. Thus, the transmission energy consumption is proportional to the ratio of the request size and the bandwidth between the requester and the provider. Thus, the transmission energy consumption of vehicle i to receive a request r'_j from vehicle j is calculated as the ratio of the request size d_j and the average bandwidth b_{ij}, i.e., $\omega_i \cdot \frac{d_j}{b_{ij}}$. The parameter ω_i is the energy consumption of vehicle i to receive one unit of data. Therefore, the total energy consumption of provider i to receive requests from other vehicles is:

$$E_i^{rec} = \sum_{j \in S_i} \omega_i \cdot \frac{d_j}{b_{ij}}, \quad \forall i \in P \tag{12}$$

Similarly, the energy consumption of vehicle i to send its request to other vehicles is:

$$E_i^{send} = \sum_{j | i \in S_j} \psi_i \cdot \frac{d_i}{b_{ij}}, \quad \forall i \in \mathcal{V} \setminus P \tag{13}$$

where ψ_i is the energy consumption of vehicle i to send one unit of data to the network.

In order to keep track of the total energy saved and the extra energy spent by each vehicle when selected as requesters or providers, respectively, we define the *energy balance*. In each energy manager period, the energy balance of vehicle i, E_i^{blnc}, is calculated based on the energy balance $E_i^{blnc'}$ obtained from the previous periods, the transmission energy, and the energy savings/extra energy consumption in the current period. In practice, *a negative energy balance means energy savings compared to the case of always executing the workload locally*. Given the above models,

TABLE 1
Notation

Notation	Description
T	Duration of the resource selection period
F_e	Number of energy manager periods
T_{em}	Duration of the energy manager period
V	Set of vehicles in the coverage area
$\mathcal{V}$	Set of participating vehicles.
α, β	Risk factor
r'_{hi}	Amount of type h resource requested by vehicle i
l_{ij}	Average distance between vehicle i and j
C_{hi}	Available capacity of resource of type h
$E_i^{blnc'}$	Energy balance of vehicle i from previous period
E_i^{idle}	Idle energy consumption of vehicle i in an energy manager period
f_i	Default CPU frequency of vehicle i
d_i	Size of the request of vehicle i
b_{ij}	Average bandwidth between vehicle i and vehicle i
ψ_i	Transmission energy to send one unit of data
ω_i	Transmission energy to receive one unit of data
$\lambda_i, \theta_i, \gamma_i, \vartheta_i$	Estimated parameters

Algorithm 1 VECMAN Framework

Input: ID of the RSU executing VECMAN: RSU_ID
1: **while** true **do**
2: Update set of vehicles allocated to RSU_ID and their history: V, H
3: Calculate the probability vector p based on V and H
4: $\mathcal{V}, T \leftarrow$I-Selector$(V, p)$
5: **for** each $k \in [1, \frac{T}{T_{em}}]$ **do**
6: Update current locations of vehicles $\mathcal{V}$ and generate scenarios ξ based on H and T
7: $S, P \leftarrow$G-ERMP$(\mathcal{V}, \xi)$
8: Communicate S and P to vehicles

the energy balance of a provider in the current energy manager period is calculated as follows:

$$E_i^{blnc} = E_i^{blnc'} + \sum_{j \in S_i} \omega_i \cdot \frac{d_j}{b_{ij}} + \lambda_i \cdot f_i^3 \cdot \frac{\sum_{j \in S_i} r'_{1j}}{\vartheta_i \cdot f_i + \theta_i}, \quad \forall j \in P \tag{14}$$

Similarly, the energy balance of a requester in the current energy manager period is calculated as follows:

$$E_i^{blnc} = E_i^{blnc'} + \sum_{j | i \in S_j} \psi_i \cdot \frac{d_i}{b_{ij}} - \Bigg(\lambda_i \cdot f_i^3 \cdot \frac{r_{1i}}{\vartheta_i \cdot f_i + \theta_i} - \lambda_i \cdot {f'}_i^3 \cdot \frac{r_{1i} - r'_{1i}}{\vartheta_i \cdot {f'}_i + \theta_i} \Bigg), \quad \forall i \in \mathcal{V} \setminus P \tag{15}$$

Formulation. The objective of the energy manager is to find a set of providers P over all possible set of providers Π and the set of replicas S_i assigned to each provider $i \in P$ over all possible replica assignments Γ_i so that the maximum energy balance over all vehicles is minimized:

$$\min_{P \in \Pi, S_i \in \Gamma_i | i \in P} \max_{j \in \mathcal{V}} \left\{ E_j^{blnc} \right\} \tag{16}$$

where E_j^{blnc} is obtained based on Equation (14) and Equation (15), subject to the capacity and risk factor constraints in Equations (3) and (9), respectively. Table 1 summarizes the notation that we use in the paper.

4 VECMAN Algorithms

Because of Constraints (2) and Constraints (9) RSP and ERMP are both chance-constrained optimization problems. As a result, they are robust to location uncertainties of the vehicles. However, solving chance-constrained optimization problems optimally usually requires computationally expensive algorithms due to the large number of scenarios on the movement of the vehicles. To tackle this complexity, we only consider a sample of scenarios, where each scenario represents a potential sequence of vehicles location over consecutive energy manager periods. Then, we develop an iterative algorithm called I-Selector to find a solution for RSP and a greedy algorithm called G-ERMP to find a solution for ERMP in polynomial time.

An algorithmic description of the proposed VECMAN framework is given in Algorithm 1. The VECMAN framework running on each RSU (each one uniquely identified by an RSU_ID) starts each vehicle management loop (Lines 2-8) by updating the set of vehicles V currently within the RSU coverage area and by getting their location history H (Line 2). The RSU coverage area is divided into multiple cells. VECMAN calculates the vector p that gives, for each vehicle, the probability of leaving the RSU coverage area from each one of these cells for various candidate lengths of resource selection periods (Line 3). We refer the reader to Section 5.1 for more details on how p is obtained from a real dataset. Based on V and p, the I-Selector determines the length of the next resource selection period T and, at the same time, the subset of vehicles $\mathcal{V}$ that with a high probability remain in the RSU coverage area over that period (Line 4). Then, VECMAN starts executing the energy manager (Lines 5-8).

At the beginning of each one of the F_e energy manager periods, with $F_e = T/T_{em}$, the G-ERMP algorithm updates the current locations of the participating vehicles and generates a set of possible scenarios to predict where each vehicle may be by the end of the current energy manager period (Line 6). We refer the reader to Section 5.1 for more details on how the scenarios are obtained using a real dataset. Based on these scenarios, the G-ERMP algorithm executes to decide the vehicles' state, the number of workload replicas, and their allocations to the selected providers, for minimized energy consumption while ensuring a low risk of failure and good QoS (Line 7). Then, the computed allocations for the current energy manager period are communicated to the vehicles (Line 8). These steps (Lines 6-8) are repeated every T_{em} units of time for F_e times, after which VECMAN starts a new vehicle management loop (Lines 2-8).

Algorithm 2 I-Selector

Input: Set of vehicles: V
Vector of probabilities of leaving the coverage area: p
1: $T \leftarrow T_{em}$
2: $R \leftarrow 0$
3: $\mathcal{V} \leftarrow \emptyset$
4: $stop \leftarrow$ false
5: **while not** $stop$ **do**
6: $\mathcal{V}_{new} \leftarrow$find-satisfied-set$(V, p^T)$
7: $R_{new} = \sum_{i \in \mathcal{V}_{new}} M_i \cdot T$
8: **if** $R_{new} \geq R$ **then**
9: $R \leftarrow R_{new}$
10: $\mathcal{V} \leftarrow \mathcal{V}_{new}$
11: $T \leftarrow T + T_{em}$
12: **else**
13: $stop \leftarrow$ true
14: $T \leftarrow T - T_{em}$
Output: $\mathcal{V}, T$

4.1 The I-Selector Algorithm

The I-Selector algorithm is given in Algorithm 2. The inputs of the algorithm are the set of vehicles V with their initial location in the coverage area, and the vector p calculated as described in the previous section. The output of the algorithm consists of the set of participating vehicles $\mathcal{V}$ and the length of the resource selection period T.

I-Selector starts with a small value of T corresponding to the minimum energy manager period T_{em} (Line 1). It increases this value iteratively until it obtains a length of

period that maximizes the amount of available computing resources (Lines 5-14). Since the objective function of RSP (i.e., Equation 1) is a bitonic function of T (the total available capacity first increases by the increase of T, and then decreases) we can guarantee that I-Selector finds the optimal length for resource selection period for the given sample of scenarios. In each iteration, for the current value of time period T, I-Selector calls find-satisfied-set procedure to obtain the set of vehicles $\mathcal{V}_{new}$ that with a risk factor less than α stay in the coverage area for T units of time (Line 6). The input of find-satisfied-set procedure is the set of vehicles V with their initial locations and the vector $p^T = \{p_a^T\}$ that gives the probability of leaving the coverage area from each cell a within T units of time. As the output, the procedure returns the set of vehicles $\mathcal{V}_{new}$, where for each vehicle in $\mathcal{V}_{new}$ with an initial location a, the probability of leaving the coverage area is less than α (i.e., $p_a^T < \alpha$). Then, I-Selector computes R_{new}, the computing resources of vehicles in $\mathcal{V}$ (Line 7). If there is no improvement in the amount of available resources, the algorithm stops; otherwise, it increases the value of T and continues this procedure as long as the amount of available resources for the new time period T is higher than that for the previous time period.

Complexity Analysis. The time complexity of I-Selector is $O(V \cdot F_e \cdot |\xi^T|)$. The main part of I-Selector consists of the loop in Lines 5-14, which executes $\frac{T}{T_{em}} = F_e$ times. In each iteration, we call satisfied-set which takes $O(|\xi^T| \cdot V)$ time. Therefore, the total time complexity of I-Selector is $O(V \cdot F_e \cdot |\xi^T|)$.

4.2 The Energy Manager Algorithm (G-ERMP)

G-ERMP operates in two phases using various *scenarios* to handle the chance constraint. Each scenario assumes a deterministic (i.e., known) location for vehicles. In the first phase, the algorithm picks a random sample of scenarios generated based on a set of probable locations for the vehicles. Therefore, the algorithm solves the deterministic version of ERMP to obtain a solution for each scenario. Because the location within a scenario is known, this solution provides a single replica for each requester vehicle. Then, in the second phase, based on the assignments obtained for each scenario, the algorithm determines the number of replicas for each vehicle as well as their replica assignment so that Constraint (9) is satisfied with a probability higher than $(1 - \beta)$. However, the problem solved in the first phase of G-ERMP belongs to the class of packing problems, which are known to be NP-hard. Therefore, it is not solvable in polynomial time, unless P=NP. Thus, we first develop a greedy algorithm called GD-ERMP, that solves the problem associated with the first phase of G-ERMP for a selected scenario. Then, we describe the complete G-ERMP algorithm, which examines the solutions provided by GD-ERMP to finalize the selection of the providers and the number of replicas.

4.2.1 The GD-ERMP Algorithm

In order to minimize the maximum energy balance of vehicles, GD-ERMP analyzes the energy balance of each vehicle at the beginning of each period: vehicles with a low energy balance are more likely to be selected as the providers for the current period. However, if the decision is made only based on the energy balance, we might obtain a solution in which providers are distributed very irregularly. Thus, the decision maker needs to consider a high number of providers in the system so that all the selected requesters are covered within a reliable distance. In other words, making decisions only based on the energy balance may increase the number of providers. Thus, the leftover CPU capacity may not be used efficiently. To solve this problem, our algorithm considers both the location of the vehicles and their energy balance.

Algorithm 3 GD-ERMP

Input: Set of vehicles: $\mathcal{V} = \{(C_i, r_i, E_i^{blnc'})\}$
Scenario for the location of vehicles: ε

1: $i \leftarrow \text{argmin}_{j \in \mathcal{V}} E_j^{blnc'}$
2: $P \leftarrow \{i\}$
3: $stop \leftarrow$ false
4: **while not** $stop$ **do**
5: $stop \leftarrow$ true
6: $S_i \leftarrow \emptyset \quad \forall i \in P$
7: **for** $j \in \mathcal{V} - P$ **do**
8: $i \leftarrow$ find-provider(j, P, ε)
9: **if** $i > 0$ **then**
10: $S_i \leftarrow S_i \cup \{j\}$
11: **else**
12: $i \leftarrow \text{argmax}_{j \in \mathcal{V}-P}(\frac{l(j,P,\varepsilon)}{l} - \frac{E_j^{blnc'}}{\bar{E}})$
13: $P \leftarrow P \cup \{i\}$
14: $stop \leftarrow$ false
15: **break**

Output: S, P

The algorithm defines a set of providers P, which is updated in an iterative manner. The algorithm starts with a minimum possible number of providers, i.e., $|P| = 1$. This value increases in the next iteration if the current providers are not able to provide a good connectivity for all the requesters or they do not have enough resources to process the workloads. GD-ERMP stops when all requesters are allocated within a reliable distance.

Algorithm 3 shows the pseudo-code of GD-ERMP. It considers both the energy balance and the location of vehicles to decide the set of providers P and the assignment of the replicas. The inputs are the vector of vehicles with their request type r_i, capacity C_i, and their current energy balance $E_i^{blnc'}$, and a scenario ε for the location of vehicles in the current energy manager period. The outputs are the set of providers P and the set of replicas $\{S_i\}$ allocated to each provider i.

In order to determine the providers, GD-ERMP first picks a vehicle with the minimum energy balance, and puts it in the set of providers P (Lines 1-2). Then, in an iterative manner, other providers are added to P. For each vehicle j that is not selected as a provider, the algorithm calls find-provider to find the nearest provider i that (i) has *enough capacity* to serve the vehicle; (ii) is within a reliable distance, i.e., $l_{ij} \leq \delta$; (iii) works at a lower frequency; and (iv) by offloading a part of workload of vehicle j to it the system achieves energy savings (Line 8). To determine the amount of offloading for vehicle j, find-provider considers the maximum possible amount of workload r'_j that can be processed on the remaining capacity of provider i. Then, based on Equations (10-13), it obtains the amount of energy

saving for the system by this offloading. The positive energy saving means that provider i can serve the request of vehicle j; otherwise, the algorithm considers the possibility of offloading to other providers. If this procedure does not find such a provider, it returns a negative value; otherwise it returns the index of the provider. Thus, if find-provider finds such a provider, the requester is assigned to that provider and the replica set of that provider is updated (Lines 9-10). If the algorithm cannot allocate a request within a reliable distance, the current set of providers is not enough to satisfy all the requests. Hence, the algorithm needs to add another vehicle to the set of providers. The next provider is chosen so that it has a relatively low energy balance and it is far away from the already selected providers, which helps covering more requesters with a minimum number of providers. This strategy is implemented in Lines 12-13, where the algorithm picks a vehicle that has the maximum value of $\left(\frac{l(j,P,\varepsilon)}{\bar{l}} - \frac{E_j^{blnc'}}{\bar{E}}\right)$, where $l(j,P,\varepsilon)$ is the minimum distance of vehicle j from the set of selected providers under the scenario ε. $\bar{l}$ and $\bar{E}$ are the average distance over vehicles and the average energy balance over vehicles, respectively. The above procedure is repeated until all requests are allocated to providers that are within a reliable distance.

Complexity Analysis. The time complexity of GD-ERMP is $O(|\mathcal{V}|^3)$. The main part of GD-ERMP consists of the loop in Lines 4-14, which executes $|P|$ times. In each iteration, for each non-provider vehicle, finding the nearest provider among j providers will take $O(j)$ time. Therefore, the total time complexity of GD-ERMP is $\sum_{j=1}^{|P|}(|\mathcal{V}|-j)\cdot j = O(|\mathcal{V}|^3)$

Algorithm 4 G-ERMP

Input: Set of vehicles: $\mathcal{V} = \{(C_i, r_i)\}$
Set of scenarios :ξ

1: $P \leftarrow \emptyset$
2: $S_i \leftarrow \emptyset \quad \forall i \in \mathcal{V}$
3: $\sigma_i \leftarrow 0 \quad \forall i \in \mathcal{V}$
4: $\Gamma \leftarrow \emptyset$
5: **for** each $\varepsilon \in \xi$ **do**
6: $(\tilde{S}, \tilde{P}) \leftarrow$GD-ERMP($\mathcal{V}$,$\varepsilon$)
7: **for** each $i \in \tilde{P}$ **do**
8: **for** each $j \in \tilde{S}_i$ **do**
9: $w_{ji} \leftarrow w_{ji} + 1$
10: $indeg_i \leftarrow indeg_j + 1$
11: **while** $|\Gamma| < \mathcal{V}$ **do**
12: $j \leftarrow \text{argmax}_{i \in \mathcal{V} \setminus P}\, indeg_i$
13: $P \leftarrow P \cup \{j\}$
14: $\Gamma \leftarrow \Gamma \cup \{j\}$
15: **for** each $i \in P$ **do**
16: **if** $j \in S_i$ **then**
17: $S_i \leftarrow S_i - \{j\}$
18: **for** each $k \in \mathcal{V} \setminus \Gamma$ **do**
19: **if** $w_{ki} > 0$ **and** available(k,i,S) **then**
20: $S_i \leftarrow S_i \cup \{k\}$
21: $\sigma_k \leftarrow \sigma_k + w_{ki}$
22: **if** $\sigma_k > (1-\beta)\cdot|\xi|$ **then**
23: $\Gamma \leftarrow \Gamma \cup \{k\}$
24: **for** each $g \in \mathcal{V} \setminus \Gamma$ **do**
25: **if** $w_{gj} > 0$ **and** available(g,j,S) **then**
26: $S_j \leftarrow S_j \cup \{g\}$
27: $\sigma_g \leftarrow \sigma_g + w_{gj}$
28: **if** $\sigma_g > (1-\beta)\cdot|\xi|$ **then**
29: $\Gamma \leftarrow \Gamma \cup \{g\}$

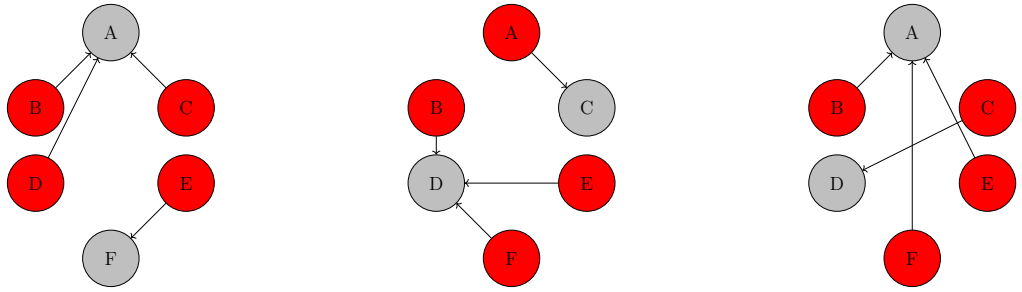
Output: S, P

4.2.2 The G-ERMP Algorithm

Algorithm 4 shows the pseudo-code of G-ERMP. The algorithm has as input the set of scenarios, ξ, the vector of vehicles with their request size, r_i, and their capacity, C_i. The output consists of the set of providers P and the set of replica's assignments S for the current energy manager period. The main idea of G-ERMP is to create a graph based on the replica allocations obtained for each scenario by the GD-ERMP algorithm. Each vertex of this graph represents a vehicle; each edge of the graph indicates a requester j to provider i assignment, weighted by the number of scenarios in which a request of j has been allocated to i by the GD-ERMP algorithm. Then, the algorithm partitions this graph into the set of providers and the set of requesters, and determines the number of replicas for each requester. This partitioning is done so that, for each vehicle, Constraint (9) is satisfied for more than $(1-\beta)\cdot|\xi|$ scenarios.

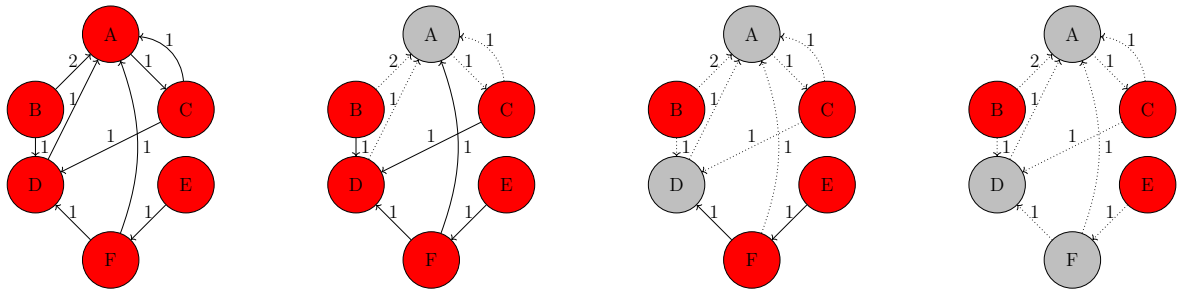
G-ERMP starts with an empty set of providers P and empty set of replicas' assignments (Lines 1-2). In each iteration of the algorithm, these sets will be updated. Also, we define vector $\sigma = \{\sigma_i\}$ to store the number of scenarios in which Constraints (9) are satisfied for each vehicle i (Line 3). We define Γ as a set of vehicles for which Constraints (9) are satisfied. G-ERMP initializes Γ with the empty set (Line 4). Sets $\tilde{P}$ and $\tilde{S} = \{\tilde{S}_i\}$ are used to save the set of providers and the set of replicas obtained for each scenario by GD-ERMP (Lines 5-6). G-ERMP creates a graph with $|\mathcal{V}|$ vertices. Each vertex represents a vehicle and each edge indicates a request-provider assignment. The weight of an edge from vertex j to vertex i is denoted by w_{ji} and is defined as the number of scenarios in which vehicle j is assigned to vehicle i. The indegree of vertex i, i.e., the total weight of edges adjacent to vertex i, is stored in vector $indeg = \{indeg_i\}$ (Lines 7-10).

In order to find the minimum number of providers, in each iteration, G-ERMP selects the vehicle as the provider that has received the maximum number of requests from the various scenarios. Therefore, it choses the vertex with the maximum indegree as a provider (Line 12). Then, it updates the set of providers (Line 13). When a vehicle is selected as a provider, it runs its requests locally, which means that, for this vehicle, Constraint (9) is automatically satisfied. Thus, it adds the current provider to the set Γ (Line 14). The algorithm then updates the replica assignment of vehicles in two steps. In the first step, since vehicle j is selected as a provider, the algorithm removes all the previous assignments from vehicle j on any provider. In fact, the algorithm checks if a request from vehicle j has been assigned to a provider i, it removes that assignment (Lines 16-17). Furthermore, since the remaining capacity of vehicle i is increased, it might be able to serve more requests. Thus, for each vehicle $k \in V - \Gamma$ willing to be assigned to vehicle i, the algorithm updates the assignment if vehicle i has enough capacity. It also updates σ for vehicle k. If σ_k is greater than $(1-\beta)\cdot|\xi|$, the algorithm adds vehicle k to Γ. Therefore, the algorithm will not generate any further replica for that vehicle (Lines 18-23). In the second step, the algorithm assigns requests from each vehicle g willing to be assigned to vehicle j. It updates the assignment if vehicle i



(a) Scenario 1 (b) Scenario 2 (c) Scenario 3

Fig. 3. Example: A problem instance with six vehicles and three scenarios.



(a) (b) (c) (d)

Fig. 4. Example: Replica placement obtained by G-ERMP

has enough capacity. It also updates σ for vehicle g. If σ_g is greater than $(1-\beta)\cdot|\xi|$, the algorithm adds vehicle g to Γ (Lines 24-29). The algorithm continues this procedure until all the vehicles are added to set Γ.

Complexity Analysis. To investigate the time complexity of G-ERMP, we analyze the time complexity of the two main parts of the algorithm. In the first part, G-ERMP calls GD-ERMP for each scenario. Therefore, as analyzed in the previous section, the time complexity of the first part is $O(|\xi|\cdot|\mathcal{V}|^3)$. In the second part, G-ERMP builds a graph based on the solution obtained in the first part. The time complexity of the second part mainly depends on the loop in Lines 11-29, which executes $O(|\mathcal{V}|)$ times. The main part of the loop consists of the loop in Lines 15-23 which executes $O(|P|\cdot(|\mathcal{V}\setminus\Gamma|))$ times. Therefore, the time complexity of the second part is $O(|\mathcal{V}|^3)$. As a result, the total time complexity of G-ERMP is $O(|\xi|\cdot|\mathcal{V}|^3+|\mathcal{V}|^3)=O(|\xi|\cdot|\mathcal{V}|^3)$

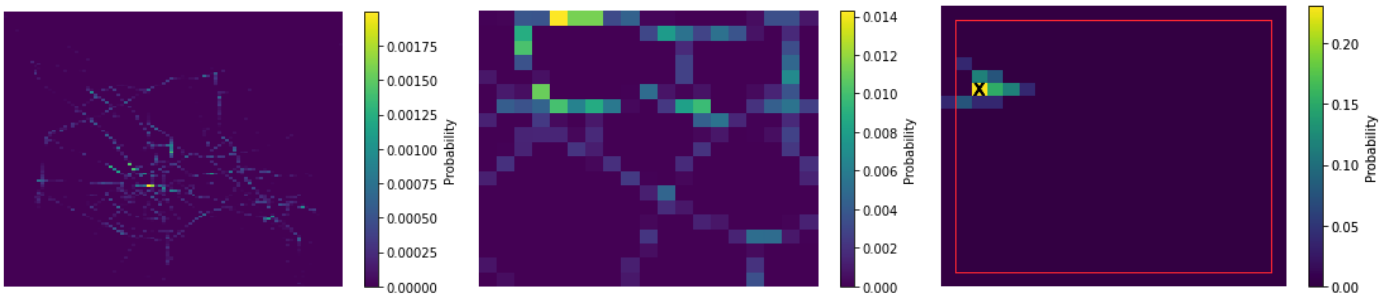
4.3 G-ERMP Execution: Example

Here, we provide an example to show how the G-ERMP algorithm works. We consider an area with six vehicles. We assume that there are three scenarios for the predicted locations of the vehicles. In this example, we set $\beta=0$. Therefore, a feasible solution is obtained when for each vehicle Constrain (9) is satisfied for all scenarios. In Figure 3, we show the three solutions obtained by GD-ERMP for each scenario. In the first scenario (Figure 3a), the request from vehicles B, C, and D are assigned to A and the request from E is assigned to F. In the second scenario (Figure 3b), requests from B, E, and F are assigned to D and the request from A is assigned to C. In the third scenario (Figure 3c), requests from B, F, and E are assigned to A and request from C is assigned to D. Now, we show how G-ERMP determines the set of requesters, the set of providers, and the replica assignment based on these solutions. Figure 4a shows the graph obtained by the solutions for each scenario. The weight on an edge from i to j indicates the number of scenarios that recommend an assignment from vehicle i to j. For example, the weight on edge (B,A) is two because in two scenarios (Figures 3a and 3c), the request from vehicle A is assigned to B.

Figure 4b shows the first iteration of the algorithm. In this iteration, vehicle A, is selected as a provider because it has the maximum indegree. Replicas from B, D, and C are respectively assigned to A. The dotted arrows in the figures are used to highlight the edges that are not used anymore to compute the nodes' indegree. Note that due to the limited capacity, replicas from F are not assigned to A. Figure 4c shows the next iteration of the algorithm. In this iteration, vehicle D, which has the second-highest indegree, is selected as a provider. Thus, a replica from B and C is allocated to D. Again, due to the the limited capacity, vehicle F is not assigned to D. However, since D is a provider now, its previous assignment on A is removed and the capacity of A is updated. Now, there is enough capacity for A to serve replicas from F. Therefore, its corresponding edges are marked with dotted arrows. In the last iteration (Figure 4d), the last unsatissfied request from E is served by F, which is marked as provider since its indegree is higher than that of E.

5 Experimental Analysis

In this section, we describe our experimental setup and then analyze the experimental results.



(a) (b) (c)

Fig. 5. Scenario generation model: (a) Distribution of vehicles over a region of Cologne city, (b) Distribution of vehicles over the most congested area (2 km by 2 km), (c) An example of vehicles' mobility model.

5.1 Experimental Setup

Computing Setup. Table 2 shows the parameters used to generate instances in our analysis. $U[x, y]$ indicates the uniform distribution within the interval $[x, y]$, and $N(x, y)$ indicates the normal distribution with mean x and variance y. We have tested VECMAN for various values of T_{em} but observed little variation on its general behavior. Thus, due to limited space, in the rest of the section we show the result only for $T_{em} = 10$ seconds. We assume that for each type of resources, the capacity of vehicles is in the same range and does not vary significantly. Therefore, we use the normal distribution for the memory and storage capacity of vehicles. According to Equation (8), the CPU capacity of a vehicle depends on its frequency and the size of the local workload. We use as an example CPU ARM Cortex A57 that has 13 frequency levels from 700 MHz to $1,900$ MHz. We profile the Cortex A57 in terms of MIPS and power consumption for each frequency level to get the model parameters in Equations (8) and (10). We define three types of instances with light, moderate, and heavy workloads. For light workloads, the default frequency varies from 700 to 1000, while for moderate and heavy cases, it varies from 1100 to 1400 and 1500 to 1900, respectively. For all instances the required time to execute workloads is uniformly drawn from $[1, 10000]$ milliseconds. Thus, by using Equation (6), the number of CPU instructions for each instance is $r_{1i} = (\vartheta_i \cdot f_i + \theta_i) \cdot \frac{U[1,10000]}{1000}$ and the CPU capacity is $C_{1i} = (\vartheta_i \cdot f_i + \theta_i) \cdot T_{em} - (\vartheta_i \cdot f_i + \theta_i) \cdot \frac{U[1,10000]}{1000}$. According to this equation and since the execution time of all types of workloads is in the same range, the expected CPU capacity of vehicles with heavy workloads is higher than that of lighter workloads.

We estimate the transmission energy parameters ω_i and ϕ_i based on the analysis provided in [26]. We also set the value of μ to 27Mb/s (equivalent to 3.375MB/s) which is the maximum data transmission in DSRC networks [27]. Since the length of energy manager period is 10 seconds, the maximum size of data that can be transmitted within an energy manager period in a unit of distance is 33.75MB. Considering the computational time needed to execute workload on vehicles within each T_{em}, to guarantee QoS, we define three problem instances with light, moderate, and heavy amount of data. In these instances the size of data varies from 0.5MB to 9MB.

TABLE 2
Distribution of parameters

Parameter	Distribution/Value	Parameter	Value
f_i	light: $U[700, 1000]$	r_{2i}	$U[100, 1000]$
	moderate: $U[1100, 1400]$	r_{3i}	$U[100, 1000]$
	heavy: $U[1500, 1900]$	μ	3375
α, β	0.1	λ_i	0.00125
ω_i	0.2	ϕ_i	0.2
d_i	light: $U[500, 1000]$	θ_i	-4558.52
	moderate: $U[1000, 5000]$	ϑ_i	7.683
	heavy: $U[5000, 9000]$	γ_i	-0.741625
C_{2i}	$N[5000, 500]$	T_{em}	10
C_{3i}	$N[5000, 500]$		

RSU Coverage Area Setup. We use the dataset of vehicular mobility in the city of Cologne, Germany [8]. The dataset contains the traces of vehicles over a region of 400 square kilometers during the 24 hours of a working day with a granularity of one second. Due to the large amount of data in the dataset, we generate vehicle mobility scenarios using only data during the rush hour (i.e., 7:30am-8:30am) to stress-test VECMAN when the maximum number of vehicles are in traffic.

We assume that the coverage range of the RSU is 1 kilometer, which is similar to the radius of the DSRC [2]. Thus, we consider a 2 kilometers by 2 kilometers area of Cologne that has the heaviest traffic during the selected rush hour and assume the RSU is at the center of this area. For simplicity, we consider the area as a two-dimensional grid of 20×20 cells in which the size of each cell is 100 meters by 100 meters. Figure 5a shows the traffic over the city of Cologne during rush hour and Figure 5b shows the traffic of the most congested area selected for our tests. We observe that on average, in every second, there are 720 vehicles in the RSU coverage area. Thus, we set the number of vehicles V that are initially in the coverage area to 720. For each cell of the RSU coverage area, we obtain the average number of vehicles that are located in that cell. We use this number as a probabilistic parameter to initialize the distribution of vehicles across the coverage area.

I-Selector Probability Vector. According to the I-Selector Algorithm, we need to consider various lengths of resource selection periods to determine the length that maximizes the objective function (i.e., Equation 1). For each candidate length T, for each cell in the RSU coverage area, and given

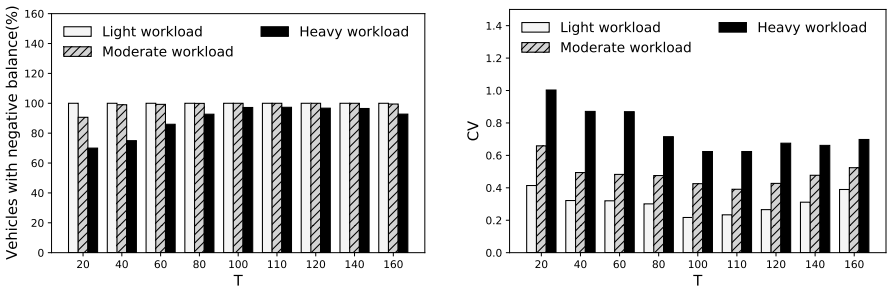


Fig. 6. Performance for various lengths of the resource selection period (a) Percentage of vehicles with a negative energy balance, (b) Coefficient of Variation (CV) of energy balance.

the vehicles initial locations when the I-Selector is invoked, we obtain the probability for each vehicle to leave the RSU coverage area before T seconds. For this purpose, we consider the vehicles' location records for each T seconds time intervals in the dataset. Based on these records, we obtain the total number of vehicles that are initially located in each cell, but they leave the coverage area by time slot T. For each cell, we then divide this number to the total number of vehicles that are initially located in the cell, which is the probability for each vehicle in each cell of leaving the RSU coverage area by the end of T.

Scenario Generation for G-ERMP. We generate scenarios for the future locations of vehicles based on the current location of vehicles and the movement probabilities of vehicles between the cells of the grid area. To determine the probability that a vehicle moves from one cell A to cell B in each energy manager period, we consider the number of movements from cell A to cell B divided by the total number of departures from cell A. Figure 5c shows the movement probability from the cell marked with X to other cells as a heat map. The red square shows the RSU coverage area. In this example there is a low probability a vehicle will leave the RSU coverage area. These probabilities are then used to generate the various possible scenarios.

Performance Metrics. The performance of VECMAN is evaluated by computing the percentage of total energy savings, which is defined as the ratio between the total energy savings of vehicles and the total baseline energy consumption (i.e., vehicles run their requests locally).

$$\text{ES}(\%) = 100 \cdot \frac{\sum_{i \in \mathcal{V} \setminus P} (\vartheta_i \cdot f_i + \theta_i) \cdot n_i}{\sum_{i \in \mathcal{V}} (\vartheta_i \cdot f_i + \theta_i) \cdot n_i} \tag{17}$$

To evaluate the fairness of VECMAN, we determine the Coefficient of Variation (CV) over energy balance of the vehicles. A lower value of CV means a more fair distribution of requests. CV is defined as the ratio of the standard deviation of E_i^{blnc} over the average energy balance across vehicles $\bar{E}$,

$$CV = \frac{\sqrt{\frac{1}{\mathcal{V}} \sum_{i=1}^{V} (E_i^{blnc} - \bar{E})^2}}{\bar{E}} \tag{18}$$

G-ERMP and I-Selector are implemented in C++ and executed on an Intel 1.6GHz Core i5 with 8 GB RAM.

5.2 Experimental Results

In this section, we first investigate the impact of the length of resource selection period on the fairness and the energy balance of vehicles. We show that the length obtained by I-Selector algorithm yields a fair distribution of workloads and a low energy balance among the vehicles compared to other possible lengths of resource selection period. Then, we investigate the performance of the VECMAN compared to the baseline that executes workload of each vehicle locally for the light, moderate, and heavy workload instances. Next, we investigate the performance of VECMAN compared to the baseline for instances with light, moderate, and heavy data transmission sizes. Finally, we investigate the scalability of VECMAN compared to a baseline that only offloads vehicles' workload to the local RSU while changing the number of vehicles .

5.2.1 Impact of the Resource Selector

We run the I-Selector algorithm on the set of vehicles that are initially located in the coverage area. To investigate the efficiency of the resource selection, we compare the fairness and energy balance of the participating vehicles obtained for various lengths of resource selection period. We vary the value of T from 20 seconds to 160 seconds. For each length T, we obtain the participating vehicles who for the next T units of time stay in the area with probability not less than $1 - \alpha$ and periodically execute our G-ERMP algorithm using the selected $T_{em} = 10$ seconds. For fairness of comparison, we run all the experiments over a fixed time interval (200 seconds) and then, for each baseline, average the results across resource-selection intervals.

Figures 6a and 6b show the average percentage of vehicles with negative energy balance and average coefficient of variation at the end of each resource selection period, respectively. As the figures show, the optimal resource selection period that ensures both fairness and high energy savings for most participating vehicles is $T = 110$ seconds, which is also the period selected by the I-Selector algorithm of VECMAN. The corresponding number of vehicles selected for participation is 374. A higher energy unbalance across participating vehicles is observed for $T < 110$ and $T > 110$, specially for heavy workload type. Specifically, when $T < 110$ the energy manager runs fewer times during each resource selection period, therefore giving no time to some participating vehicles to become both requester and

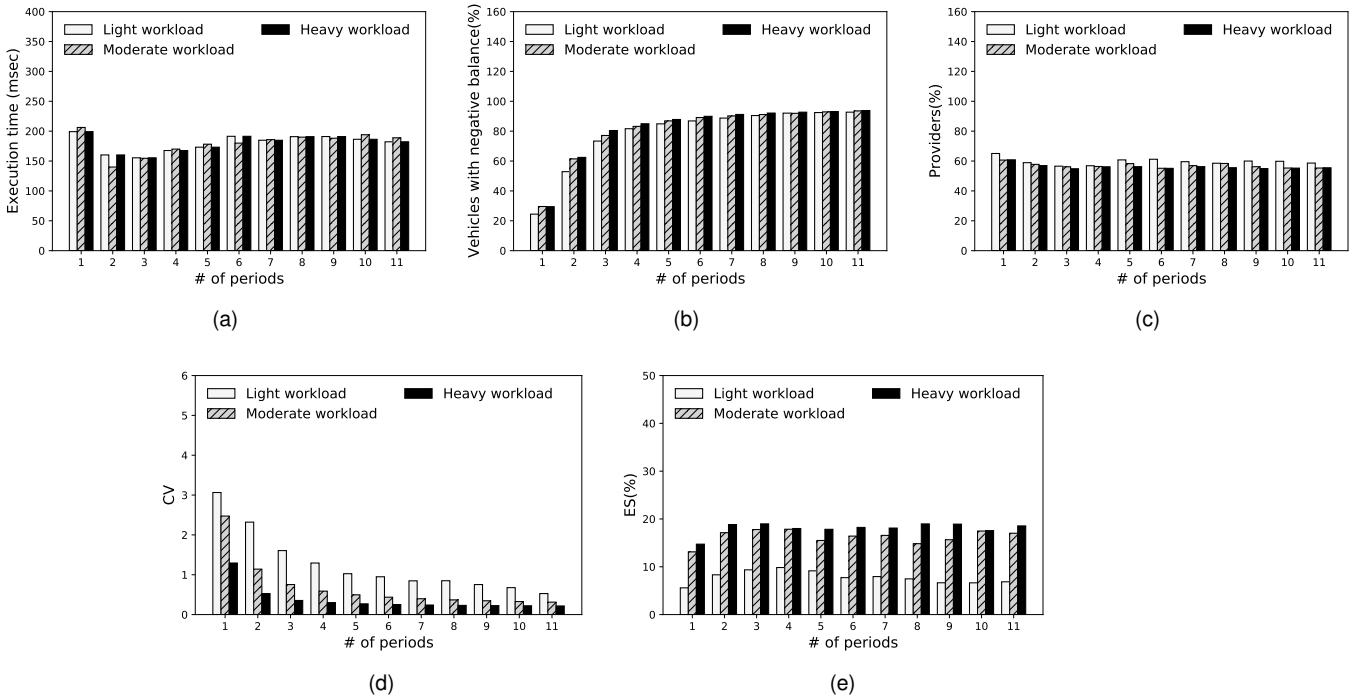


Fig. 7. Performance with respect to the workload types: (a) Execution time of G-ERMP, (b) Percentage of vehicles with negative balance, (c) Percentage of vehicles selected as providers, (d) Coefficient of Variation (CV) of energy balance, and (e) % of energy savings.

provider before the period expiration. As a result, during each resource selection interval some vehicles may have been only in requester mode, thus enjoying energy savings, while some others may have been only in provider mode, which leads to no energy savings, i.e., unfairness. On the other hand, when $T > 110$ the number of vehicles that are likely to remain in the RSU coverage area and are therefore selected for participation decreases, which leads to a lower amount of shared resources during each resource selection period. This low resource availability leads to a higher imbalance in energy savings across vehicles because a fewer number of workloads can be placed on the lower number of available providers.

These experiments show that the I-Selector algorithm helps G-ERMP provide fair energy savings among the participating vehicles.

5.2.2 Performance vs. Workload Types

As we discussed in previous section, the resource selector chooses 374 vehicles to participate in resource sharing with optimal period $T = 110$ seconds. Here we analyze the performance of VECMAN by considering these vehicles and run the G-ERMP in 11 energy-manager periods. We consider three sets of workload instances: light, moderate, and heavy. In these instances, the size of transmission data of vehicles is moderate. Figure 7a shows that the average execution time of G-ERMP for each problem instance is less than 0.2 second, which is negligible compared to the execution period of the requests ($T_{em} = 10$ seconds). The execution times for the three types of workloads are almost the same. For some of the periods, the execution time of G-ERMP for problem instances with moderate workload is slightly higher than that for problem instances with heavy workload. The reason is that for some periods, in the case of moderate workload instances, the percentage of providers that do not execute only their local workload is slightly higher than in the case of high workload instances. Thus, the graph generated by G-ERMP for problem instances with moderate workload is slightly more complex compared to the graph generated for instances with heavy workload. As a result, the time needed to partition the graph is slightly higher in the case of instances with moderate workloads.

As Figure 7b shows, the percentage of vehicles with a negative balance increases over the periods. For all problem instances, after all periods, more than 92% of vehicles obtain energy savings. However, for problem instances with heavy workload, we observe a higher percentage of vehicles achieving energy savings compared to the moderate and light workloads. The reason is that in average, the CPU capacity of instances with heavy workload is higher than that with lighter workloads (see section 5.1). Thus, as Figure 7c shows, the percentage of providers decreases compared to the light and moderate workloads. Consequently, more energy savings are achieved for these instances, as Figure 7e shows; and a higher percentage of vehicles achieve energy savings. Furthermore, for problem instances with heavy workload, as Figure 7d shows, the CV value is generally less than that of the lighter workloads because fewer vehicles have to process their requests locally. This leads to having higher vehicles achieving energy savings. On the other hand, as Figure 7e shows, even for the light-workload case the vehicles can achieve about 7% more energy savings compared to the baseline, while the moderate and heavy workloads achieve 16% and 18% energy savings, respectively.

Note that some vehicles, e.g., 8% in the above experiments, may not achieve energy savings in the current resource sharing period. However, they may achieve energy

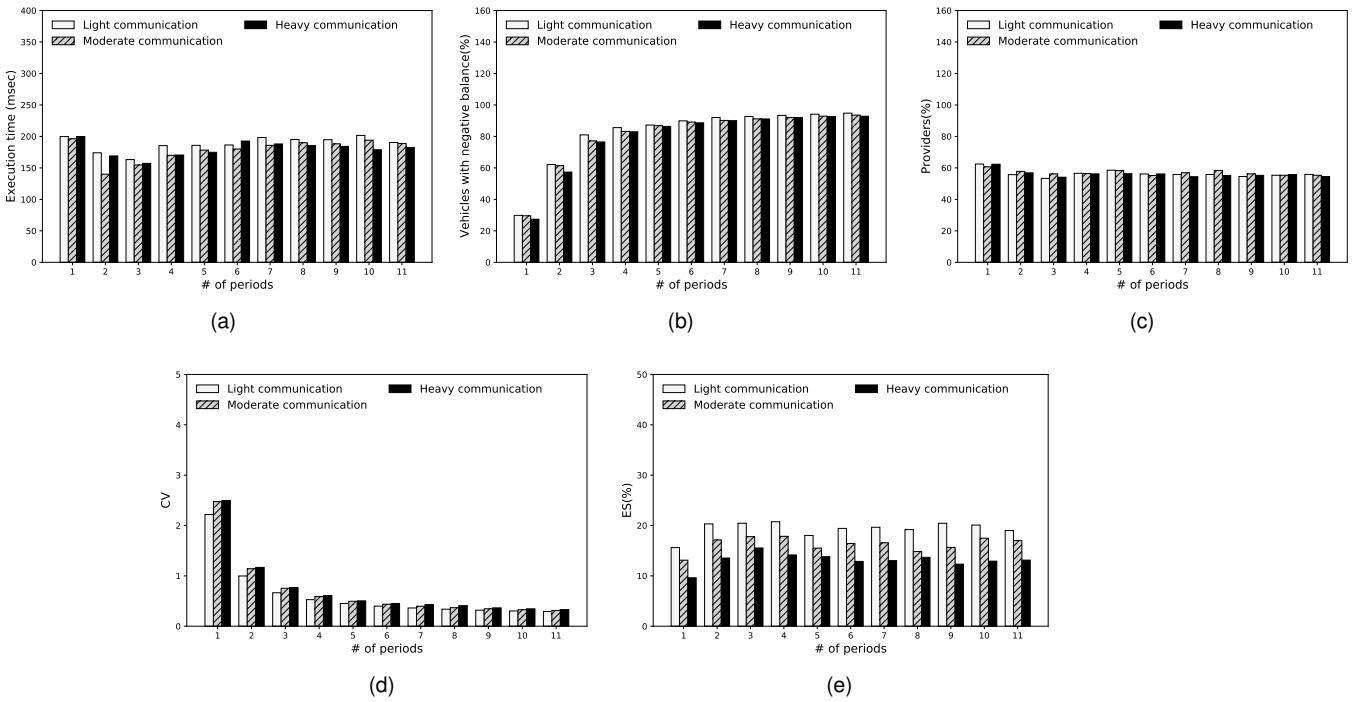


Fig. 8. The effect of data transmission on (a) Execution time of G-ERMP, (b) Percentage of vehicles with negative balance, (c) Percentage of vehicles selected as providers, (d) Coefficient of Variation (CV) of energy balance, and (e) % of energy savings.

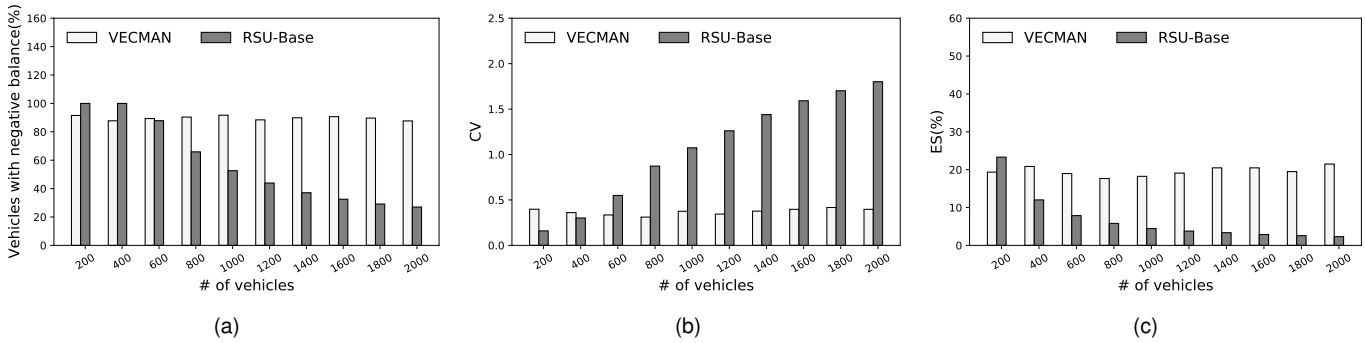


Fig. 9. The effect of number of vehicles on (a) % of vehicles with negative balance, (b) Coefficient of Variation (CV) of energy balance, and (c) % of energy savings.

savings in the next sharing periods. It is in our future work to design an algorithm that considers the initial energy balance of each vehicle to ensure that all vehicles can achieve energy savings over multiple resource selection periods.

These experiments show that VECMAN enables vehicles to achieve energy savings for various workload instances.

5.2.3 Performance vs. Data Size

In this experiment, we investigate the effect of the size of transmitted data on the performance of VECMAN. We consider three types of problem instances, light, moderate, and heavy communication. We assume that the workload of the vehicles is moderate.

Figure 8a shows the execution time of G-ERMP for the three types of instances. As we observe, the size of data does not affect the execution time of the algorithm. For all instances the execution time of the algorithm is less than 0.2 seconds. Figure 8b shows the percentage of vehicles with negative balance. We observe that for problem instances with light communication more vehicles achieve energy savings compared to the moderate and heavy communication. Figure 8c shows the percentage of vehicles that are selected as providers over the energy manager periods. As the figure shows, the percentage of providers does not change much with the increase in the data size. The reason is that the number of providers does not depend on the size of the data transmitted by the vehicles, but it depends on how the transmitted data size varies among the vehicles. Since in all types of problem instances, the size of the data transmitted follows the uniform distribution, the percentage of providers will not change. Figure 8d shows the CV of the energy balance over energy manager periods. Due to the fact that, in the case of heavy communication vehicles achieve energy savings in a slower manner, the CV value is higher than the CV value for the light and moderate communication instances. However, as Figure 8e shows, even for the heavy communication the vehicles achieve an average of 13% energy savings compared to the baseline.

These experiments show that VECMAN enables vehicles to achieve energy savings for various instances with different data sizes.

5.2.4 Performance vs. Number of Vehicles

Here, we investigate the scalability of VECMAN with respect to the number of vehicles and compare it to a baseline called RSU-Base, which uses the local RSU to run the requests of the vehicles. RSU-Base orders the vehicle requests in descending order based on the vehicles' energy balance and then, starting from the one with highest balance, it allocates as many requests as possible on the RSU's resources.

As discussed in Section 5.1, the estimated number of vehicles that are initially in the coverage area is 720. But, here we vary the number of vehicles from 200 to 2000 to investigate the scalability of the algorithm. We use the scenario generation model and resource selection setup described in Section 5.1 to generate problem instances. The vehicles' workload and the size of data is moderate. As Figure 9a shows, VECMAN enables about 90% of the vehicles to obtain energy savings while RSU-Base, because of its limited resources, cannot achieve energy savings for more than 50% of the vehicles when there are more than 1000 vehicles. As a result, as Figure 9b shows, VECMAN provides a fair distribution of the energy savings (decreasing CV value) while RSU-Base has an unbalanced savings distribution (increasing CV value) due to the limited number of vehicles that can offload their workload. This behavior, as Figure 9c shows, translates in a stable 19% energy savings with VECMAN and a decreasing amount of savings for RSU-Base with an increasing number of vehicles.

These experiments show that VECMAN enables vehicles to achieve energy savings independently from the number of vehicles it manages.

6 Conclusions and Future Work

In this paper, we proposed VECMAN, an energy-aware resource management framework for VEC systems with the aim of minimizing the energy consumption of the participant vehicles. We evaluated VECMAN by performing an extensive experimental analysis on several problem instances. The results showed that the proposed framework allows vehicles to achieve between 7% and 18% computational energy savings compared to a baseline that executes workload locally and 13% energy savings compared to a baseline that offloads vehicles' workloads only to RSUs. In our future research, we plan to improve VECMAN to (i) theoretically guarantee the minimum computational energy consumption for any data transmission size, (ii) explore the possibility of collaboration between RSUs, and (iii) allow provider vehicles to temporarily increase their default computing frequency level.

Acknowledgments

This paper is a revised and extended version of [7] presented at The IEEE International Conference on Cloud Engineering (IC2E 2020). The research was supported in part by the US National Science Foundation under grant no. IIS-1724227 and grant no. CNS-1948365.

References

[1] R. Yu, Y. Zhang, S. Gjessing, W. Xia, and K. Yang, "Toward cloud-based vehicular networks with efficient resource management," *IEEE Network*, vol. 27, no. 5, pp. 48–55, September 2013.

[2] "DSRC Technology," https://www.auto-talks.com/technology/dsrc-technology.

[3] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *ACM SIGPLAN Notices*, vol. 53, no. 2. ACM, 2018, pp. 751–766.

[4] Tesla, "Autopilot," https://www.tesla.com/autopilot, 2018.

[5] Geotab, "Gridlocked Cities - U.S. traffic congestion maps," https://www.geotab.com/gridlocked-cities/.

[6] Y. Choi, S. Park, and H. Cha, "Graphics-aware power governing for mobile devices," in *Proc. 17th Annual Int. Conf. on Mobile Systems, Applications, and Services*, ser. MobiSys '19. ACM, 2019, pp. 469–481.

[7] T. Bahreini, M. Brocanelli, and D. Grosu, "Energy-aware resource management in vehicular edge computing systems," in *Proc. of the IEEE Int. Conf. on Cloud Engineering (IC2E)*, 2020, pp. 49–58.

[8] S. Uppoor, O. Trullols-Cruces, M. Fiore, and J. M. Barcelo-Ordinas, "Generation and analysis of a large-scale urban vehicular mobility dataset," *IEEE Trans. Mobile Comp.*, vol. 13, no. 5, pp. 1061–1075, 2013.

[9] K. Zheng, H. Meng, P. Chatzimisios, L. Lei, and X. Shen, "An smdp-based resource allocation in vehicular cloud computing systems," *IEEE Transactions on Industrial Electronics*, vol. 62, no. 12, pp. 7920–7928, 2015.

[10] C. Yang, Y. Liu, X. Chen, W. Zhong, and S. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26 652–26 664, 2019.

[11] Z. Jiang, S. Zhou, X. Guo, and Z. Niu, "Task replication for deadline-constrained vehicular cloud computing: Optimal policy, performance analysis, and implications on road traffic," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 93–107, 2018.

[12] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski, "Fog following me: Latency and quality balanced task allocation in vehicular fog computing," in *IEEE International Conference on Sensing, Communication and Networking*, 2018.

[13] Y. Sun, J. Song, S. Zhou, X. Guo, and Z. Niu, "Task replication for vehicular edge computing: A combinatorial multi-armed bandit based approach," *arXiv preprint arXiv:1807.05718*, 2018.

[14] J. Zhou, D. Tian, Y. Wang, Z. Sheng, X. Duan, and V. C. Leung, "Reliability-optimal cooperative communication and computing in connected vehicle systems," *IEEE Transactions on Mobile Computing*, vol. 19, no. 5, pp. 1216–1232, 2019.

[15] T. Bahreini, M. Brocanelli, and D. Grosu, "Energy-aware speculative execution in vehicular edge computing systems," in *Proc. 2nd Int. Workshop on Edge Systems, Analytics and Networking*, 2019, pp. 18–23.

[16] X. Hou, Z. Ren, J. Wang, W. Cheng, Y. Ren, K.-C. Chen, and H. Zhang, "Reliable computation offloading for edge-computing-enabled software-defined iov," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7097–7111, 2020.

[17] L. Li, X. Zhang, K. Liu, F. Jiang, , and J. Peng, "An energy-aware task offloading mechanism in multiuser mobile-edge cloud computing," *Mobile Information Systems*, vol. 2018, pp. 3–15, April 2018.

[18] O. Muñoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 10, pp. 4738–4755, 2015.

[19] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks*, vol. 1, no. 2, pp. 89–103, 2015.

[20] H. Trinh, D. Chemodanov, S. Yao, Q. Lei, B. Zhang, F. Gao, P. Calyam, and K. Palaniappan, "Energy-aware mobile edge computing for low-latency visual data processing," in *The 5th International Conference on Future Internet of Things and Cloud*, 2017.

[21] Y. Jang, J. Na, S. Jeong, and J. Kang, "Energy-efficient task offloading for vehicular edge computing: Joint optimization of offloading and bit allocation," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*. IEEE, 2020, pp. 1–5.

[22] H. Viswanathan, E. K. Lee, I. Rodero, and D. Pompili, "Uncertainty-aware autonomic resource provisioning for mobile

cloud computing," *IEEE Trans. on Parallel and Distributed Syst.*, vol. 26, no. 8, pp. 2363–2372, 2015.
[23] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.
[24] S. Wang, Z. Qian, J. Yuan, and I. You, "A dvfs based energy-efficient tasks scheduling in a data center," *IEEE Access*, vol. 5, pp. 13 090–13 102, 2017.
[25] T. Q. Dinh, J. Tang, Q. D. La, and T. Q. Quek, "Offloading in mobile edge computing: Task allocation and computational frequency scaling," *IEEE Transactions on Communications*, 2017.
[26] E. Björnson and E. G. Larsson, "How energy-efficient can a wireless communication system become?" in *Asilomar Conference on Signals, Systems and Computers*, 2018.
[27] Z. Xu, X. Li, X. Zhao, M. H. Zhang, and Z. Wang, "Dsrc versus 4g-lte for connected vehicle applications: A study on field experiments of vehicular communication performance," *Journal of Advanced Transportation*, vol. 2017, 2017.

BIOGRAPHIES

Tayebeh Bahreini is currently a Ph.D. candidate in Computer Science at Wayne State University. She received her M.Sc. degree in Computer Engineering, from Shahed University, Iran in 2014, and her B.Sc. degree in Computer Science from University of Isfahan, Iran, in 2010. Her main research interests are edge and cloud computing, distributed systems, parallel computing, and combinatorial optimization. She is the recipient of the 2019 National Center for Women & Information Technology (NCWIT) Collegiate National Award. She was selected as one of the 2019 Top Ten Women in Edge and a Finalist for the Edge Woman of the Year 2019 Award by the Edge Computing World organization for her contribution to research in edge computing. She is a student member of ACM, IEEE, IEEE Computer Society, and INFORMS.

Marco Brocanelli is an Assistant Professor in the Computer Science Department of Wayne State University and the director of the Energy-aware Autonomous Systems Lab (EAS-Lab). He received the B.E. and M.E. degrees in Control Systems from the University of Rome Tor Vergata (Italy). In August 2018, he received the Ph.D. degree in the Electrical and Computer Engineering program of The Ohio State University. His research interests are in the area of cyber-physical systems, energy-aware systems, Internet of Things (IoT), edge computing, embedded and real-time systems.

Daniel Grosu received the Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iaşi, Romania, in 1994 and the MSc and PhD degrees in computer science from the University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is an associate professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include parallel and distributed computing, approximation algorithms, and topics at the border of computer science, game theory and economics. He has published more than one hundred peer-reviewed papers in the above areas. He has served on the program and steering committees of several international meetings in parallel and distributed computing such as ICDCS, CLOUD, ICPP and NetEcon. He is a senior member of the ACM, the IEEE, and the IEEE Computer Society.