A Context-Oriented Framework for Computation Offloading in Vehicular Edge Computing using WAVE and 5G Networks

Alisson Barbosa de Souza, Paulo Antonio Leal Rego, Tiago Carneiro, Paulo Henrique Gonçalves Rocha, José Neuman de Souza

A Context-Oriented Framework for Computation Offloading in Vehicular Edge Computing using WAVE and 5G Networks

Alisson Barbosa de Souza[a,*], Paulo Antonio Leal Rego[a], Tiago Carneiro[bc], Paulo Henrique Gonçalves Rocha[a], José Neuman de Souza[a]

[a] *GREat - Federal University of Ceará, Fortaleza, Brazil*
[b] *University of Luxembourg, Belval, Luxembourg*
[c] *INRIA Lille - Nord Europe, Lille, France*

Abstract

Despite technological advances, vehicles are still unable to meet the demands of some applications for massive computational resources in a feasible time. One way to deal with this situation is to integrate the computation offloading technique into a vehicular edge computing system. This integration allows application tasks to be executed on neighboring vehicles or edge servers coupled to base stations. However, the dynamic nature of vehicular networks, allied to overloaded servers, can lead to failures and reduce the effectiveness of the offloading technique. Therefore, we propose a context-oriented framework for computation offloading to reduce the application execution time and maintain high reliability in vehicular edge computing. The framework modules perform computational resources discovery, contextual data gathering, computation tasks distribution, and failure recovery. Its main part is a task assignment algorithm that seeks the best possible server to execute each application task, using contextual information and WAVE and 5G networks. The results of extensive experiments in different vehicular environments show that our framework reduces up to 70.3% of total execution time compared to totally local execution and up to 42.9%

*Corresponding author
Email addresses: alisson@ufc.br (Alisson Barbosa de Souza), pauloalr@ufc.br (Paulo Antonio Leal Rego), tiago.carneiro-pessoa@inria.fr (Tiago Carneiro[b]), paulorocha@great.ufc.br (Paulo Henrique Gonçalves Rocha), neuman@ufc.br (José Neuman de Souza)

compared to other literature approaches. Concerning reliability, our framework achieves to offload up to 89.4% of all tasks and needs to recover only 0.8% of them. Thus, our solution outperforms the totally local execution of the application and other existing computation offloading solutions.

Keywords: Vehicular edge computing, Computation offloading, Task offloading, WAVE, 5G, Task assignment

1. Introduction

Vehicular technologies are increasingly advancing in terms of communication and intelligence. For example, Vehicular Ad Hoc Networks (VANETs) provide Vehicle to Vehicle (V2V) and Vehicle to Infrastructure (V2I) communications [1, 2] using technologies such as Wireless Access in Vehicular Environments (WAVE)/IEEE 802.11p [3] and 5G/mmWave (millimeter Wave) [4, 5]. Vehicles have also evolved in intelligence through computing capabilities, cameras, embedded systems, sensors, and satellite navigation systems [6].

However, the advent of autonomous vehicles and new and popular applications such as augmented reality, automatic object recognition, and real-time video surveillance demand massive computing resources to deal with complicated data processing and critical latency requirements [7, 8]. Thus, despite technological advances, vehicles do not yet have sufficient on-board computing resources to handle all of these requirements in a feasible time. Even if more powerful processors were installed in the vehicles, this could compromise their energy and displacement efficiency [6].

One manner to assist vehicles with latency and processing requirements is the vehicular edge computing system. In this system, computational processing can be done on Vehicular Clouds (VCs) or edge servers in an integrated or isolated way, as can be seen in Figure 1. In turn, vehicular clouds are a pool of computational resources of two or more vehicles, stationary or in motion, which can be dynamically coordinated to offer services on demand, through V2V connections and on-board units (OBUs), as in the cloud computing model. Although

VCs have less latency in communication and operate in scenarios without infrastructure, they do not present great computational power. Another option is to use edge servers coupled with Roadside Units (RSUs) or base stations (BSs) through V2I connections. These servers are deployed close to streets and roads by service providers. Even though they have a little more communication latency and can be quite requested by network devices, these servers generally have greater computational capacity than vehicular clouds [9].

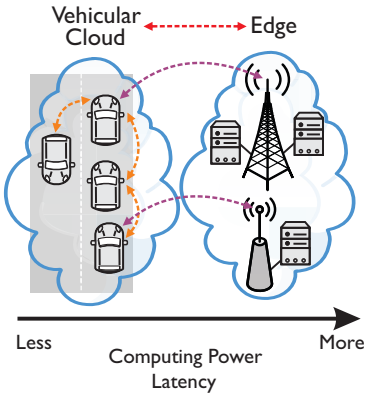


Figure 1: Vehicular edge computing (red line) system showing the integration between the vehicular cloud and the edge and aspects of latency and computing power. Orange lines represent V2V connections. Purple lines represent V2I connections.

One way to take advantage of these available computing resources is to apply the computation offloading technique, also called task offloading. This technique offloads smaller parts or tasks of an application to remote devices or servers, which are vehicles or edge servers in vehicular edge computing systems. Then, these servers process the tasks and return the result to the client. Computation offloading is used to improve applications' execution time and decrease the overload of processing [10, 11]. In fact, one of the main objectives of the computation offloading technique is to allow an application to execute in parallel in less time than execution on the client only [7].

Nevertheless, computation offloading is not always worthwhile due to communication and processing delays of remote servers [7]. Moreover, the dynamic

nature of vehicular networks can reduce the effectiveness of the offloading technique. This is because the network nodes have rapid movements, leading to frequent disconnections and offloading failures. Additionally, the servers may already be overloaded, and there is no support from a central coordination point [2]. In such cases, contextual information helps to deal with these challenges. Context refers to information that characterizes entities, including the situation of network devices or vehicles. Although contextual information changes dynamically, it assists in decision or adaptation processes and improves the performance of offloading systems [11, 12]. For example, the information about the complete trajectory or route of a vehicle, if available, enables more accurate vehicle positioning predictions and helps to deliver data between network nodes [13].

Therefore, we propose a context-oriented framework for computation offloading in vehicular edge computing systems using WAVE and 5G networks. The objective is to ensure the reduction of application execution time and the reduction of offloading failures. The framework modules help the vehicle to discover computational resources and gather contextual data from devices within its communication range. Such information is used by the algorithm of decision and task assignment called Greedy Task by Task (GTT), the main part of the framework. GTT decides whether offloading is worthwhile and, if so, decides where to send the tasks. This decision is also the most challenging part of the framework because finding the best way to distribute tasks to minimize application execution time is an NP-hard problem [9]. For this decision, we consider contextual information such as speed, location, direction, CPU capacity and availability, data rates and communication ranges WAVE and 5G, link lifetimes, tasks characteristics, distances between devices, transmission and processing times, connectivity, known routes of vehicles, and estimated arrival time to the destination. After the decision, the framework distributes the computation tasks. Besides, if any task is lost or not executed, the framework has mechanisms to recover from failures.

1.1. Challenging Issues

Thus, the main challenges related to this work are summarized below:

- Adaptation of computation offloading to the fast movement of nodes in vehicular networks. This behavior of nodes is linked to frequent disconnections, causing offloading failures [2].
- Gathering of contextual information. In dynamic environments, this information needs to be gathered and treated in real-time to be valid [12].
- NP-hardness of the task assignment decision problem to minimize application execution time. In this way, there is no exact polynomial-time solution for this problem. Furthermore, this decision needs to consider different contextual parameters [9].

1.2. Contributions

We describe the main contributions of this work as follows:

- A context-oriented framework for computation offloading in vehicular edge computing with descriptions of the conceptual architecture and offloading and failure recovery processes.
- GTT, a task assignment algorithm that seeks the best possible servers for each application task, aiming to minimize applications' execution time and process failures. For this, various parameters of contextual information are taken into account.
- Use of a special contextual information about known routes of vehicles, helping to predict vehicle positioning more accurately and avoid offloading failures.
- Simultaneous use of WAVE and 5G technologies, combining their advantages, increasing capacities, and decreasing task transmission delays.

- Extensive simulations of the proposed solution and literature algorithms, making it possible to evaluate and validate them in different vehicular environments in terms of execution time and reliability.

1.3. Organization

The remainder of the manuscript is organized as follows. Section 2 presents related works. An overview of the system model and problem formulation is given in Section 3. Section 4 describes the proposed framework, detailing its conceptual architecture and processes and its task assignment algorithm. The details of the experiments are presented in Section 5. Section 6 discusses the results. Finally, Section 7 presents the conclusions of this work.

Moreover, we have provided a list of acronyms definitions in Table 1 to make the paper easier to read.

Table 1: Acronyms definitions.

ALPR	Automatic License Plate Recognition	TL	Task executed locally
BS	Base Station	TR	Task executed with recovery
FIFO	First In, First Out	TS	Task successfully executed remotely
GCF	Greedy for CPU Free	V2I	Vehicle to Infrastructure
GTT	Greedy Task by Task	V2V	Vehicle to Vehicle
HVC	Hybrid Vehicular edge Cloud	VANET	Vehicular Ad Hoc Network
MDO	Multi-Decision based Offloading	VC	Vehicular Cloud
mmWave	Millimeter Wave	WAVE	Wireless Access in Vehicular Environments
RSU	Roadside Unit		

2. Related Works

Several works have been developed in the area of computation offloading. Among these works, some consider that the clients (devices with applications that need computing offloading) are people's smartphones or virtual/augmented reality devices that participate in non-vehicular environments [14, 15, 16]. Thus, the solutions proposed in these works do not consider important aspects that should be considered when the client is a vehicle, such as fast mobility, frequent

network topology changes, and variations in wireless communication channels [17]. In [18], clients are smartphones of people inside vehicles that send tasks to be processed on traditional cloud servers. However, in this computation offloading process, the distinctive aspects of vehicular networks are also not considered, and the objectives do not include reducing the execution time of vehicular applications. In addition, clients have energy constraints that vehicles do not have, and communicating with traditional cloud servers can add high access latency, making applications with ultra-low latency requirements unfeasible [9].

Therefore, although there are different aspects considered in the offloading works, this section focuses specifically on the principal works of computation offloading in vehicular edge computing systems that have two essential characteristics. The first is that clients are only vehicles. The second is that the main objective of the proposed solutions is to reduce vehicular applications' execution time. These two characteristics are also essential in our work.

In these vehicular edge computing systems, edge servers coupled to base stations or vehicles can act as servers and execute third parties' tasks. These servers can receive tasks by different communication technologies such as WAVE, 4G, or 5G. In [19, 20], authors presented solutions to allow vehicles to execute computation tasks on selected neighboring vehicles over 5G/V2V connections. Nevertheless, using only one communication technology limits the capabilities to transmit more network packets and prevents connections to other types of networks [21, 22, 23]. Another interesting work in [24] proposed a computation offloading framework for 5G networks. The framework allows tasks to be executed only on edge servers coupled to base stations or RSUs. However, these proposed solutions allow third party tasks to be executed only by vehicles or only by edge servers. In either case, there is a waste of computational resources from unutilized servers.

Some works allow third-party tasks to be executed by vehicles and edge servers to avoid such a waste of resources. Besides, to increase the network's transmission capacities, these works use WAVE and 4G technologies. In [25, 26, 27, 28], proposed schemes and algorithms enable a vehicle to transfer its

computations tasks to neighboring vehicles (via WAVE/V2V) or nearby edge servers (via 4G/V2I). Then, the latter can process the tasks and return their results to the requesting vehicle. Nonetheless, while it is an interesting approach, 4G networks are already starting to become outdated. They have deficiencies in relation to 5G networks, such as lower data rates, weaker for high mobility, fewer frequency ranges, and higher latencies. Also, algorithms used in 4G networks may not be well adapted to 5G networks because of differences in network architecture, modulation and multiplexing techniques, wireless signal behavior, limited range, and need for a line of sight between devices [29, 30, 31, 4].

In [32], the authors proposed a framework to take advantage of several computational resources available locally, in nearby vehicles (via V2V), and on edge servers (via V2I). WAVE, 4G, and 5G technologies can also be used to increase network capacity. Even so, there is a prioritization of sending tasks via 5G. In scenarios where only base stations and few vehicles use 5G, the sending of tasks will end up being, mostly, only via V2I and to the edge servers. Thus, this prioritization by 5G can overload the edge servers and underutilize the vehicles' computational resources. Moreover, sending via 5G can result in failures due to inherent technology challenges, such as limited range and the need for devices to be in the same line of sight. Also, if 5G fails, the algorithm in [32] will need to seek for resources via WAVE, which ends up generating more delays in the offloading process.

Even using different technologies, a computation offloading solution can work well in one scenario and not work in another. For example, scenarios with many intersections of streets (urban) or a single road (highway) influence network connectivity and interfere with the proposed solutions' performance. This also occurs with different vehicular densities [9]. Accordingly, it is important to submit the proposed solutions to different scenarios and densities, providing credibility to them, and evaluating the impact of specific situations. Despite this, as summarized in Table 2, none of these previous works used both the highway and urban scenarios in the experiments. Furthermore, most did not use all the main types of vehicular densities.

Table 2: Comparison of different aspects used in related works.

Reference	Technology		Server		Scenario		Vehicular Density			Contextual Information			Objective
	WAVE	Cellular	VC	Edge	Urban	Highway	High	Medium	Low	Known Routes	CPU Capacity	CPU Availability	+ Reliability
Ref. [19]		5G	√		√						√	√	√
Ref. [20]		5G	√			√			√		√		
Ref. [24]	√	5G		√	√				√		√	√	
Ref. [25]	√	4G	√	√		√			√		√		
Ref. [26]	√	4G	√	√		√					√		
Ref. [27]	√	4G	√	√		√			√		√		
Ref. [28]	√	4G	√	√		√		√	√		√	√	
Ref. [32]	√	4G/5G	√	√	√				√		√	√	
Ref. [33]	√	5G	√	√		√	√	√	√			√	√
Our proposal	√	5G	√	√	√	√	√	√	√	√	√	√	√

In order to help deal with different scenarios, contextual information provides data that assists computation offloading systems [11]. One such information is the known routes of vehicles, a set of geographical coordinates that vehicles will travel. This information helps estimate vehicles' positions within a given time more accurately, avoiding sending tasks to those who will lose connectivity [13]. In our previous work [33], we proposed a scheme that applies computation offloading to edge servers, via 5G/V2I, and vehicles, via WAVE/V2V. However, the scheme does not use information about CPU capacity and known routes of vehicles. In fact, although the works mentioned in this section use various contextual information, none of them use the known routes of vehicles. Moreover, most of the works in Table 2 does not aim to increase reliability by counting and recovering eventual failures, even knowing that a failure can lead to increased latency, incomplete information, and application crashes.

Our proposed framework aims to fill the gaps left by the mentioned works. In terms of technology and servers, we allow tasks to be sent to edge servers and vehicles simultaneously through 5G/V2I and WAVE/V2V connections. With this, we increase the network's transmission capacities and took advantage of all available computing resources. In addition, we use several contextual in-

formation, such as CPU capacity and availability, distances between devices, WAVE and 5G ranges, known routes of vehicles, and estimated arrival time to the destination. Then, we use an algorithm to assign tasks so that the best available servers execute them, aiming to minimize the applications' execution time and the number of offloading failures. Finally, we used different scenario configurations to analyze our proposal.

For more details about other works, we have done an extensive review and classification of several solutions of computation offloading in VANETs [9].

3. System Model and Problem Formulation

This section presents an overview of the system model and problem formulation. First, we describe the network general structure. Then, we present the models for communication and computation. Finally, we present the problem formulation. In turn, Table 3 lists the definitions of the main symbols used.

Table 3: Symbols definitions.

τ	Task	r	Transmission rate
B	Bandwidth	S	Set of chosen servers
C	Computational capacity	s	Data size
c	CPU cycles required to process a task	$server$	Chosen server for the *client*
$client$	Client vehicle	T	Workload or set of tasks
d	Distance	t	Time
F	Set of feasible servers	u	Velocity
n_τ	Total number of tasks in a workload	W	Number of tasks to be executed by *client*
P	Transmission power	X	Number of tasks to be executed by servers
p	Position	Y	Backup of T
R	Communication range		

3.1. Network General Structure

The network topology considered in this work has different types of nodes. For example, it has a set of vehicles $V = \{v_1, v_2, v_3, ..., v_{n_v}\}$, where n_v is the total number of vehicles on the network. The topology also has a set of edge servers $E = \{e_1, e_2, e_3, ..., e_{n_e}\}$, where n_e is the total number of edge servers

present on the network. Since each edge server is connected to a different 5G base station, n_e also represents the total number of 5G base stations on the network. The 5G base stations are installed on the shoulders of streets or roads. The edge servers are connected via optical fiber to the 5G base stations. With a battery power supply, vehicles can simultaneously process, store, transmit data, and use sensors. Besides, vehicles periodically generate beacon messages and distribute them in a one-hop broadcast to all nodes within their communication range.

We consider *client* as a vehicle that offloads tasks to other vehicles (via V2V communication) or to edge servers (via V2I communication). Offloading decisions are made based on information from other devices. However, the decision of whether and how to offload is made only by the *client*. This decision can lead *client* to take the following actions: execute all tasks locally, execute all tasks remotely (on one or more servers), or execute some tasks locally and some more remotely.

A *server* is any vehicle or edge server that can receive and process one or more tasks sent by the *client*. All vehicles and edge servers in our topology can act as a server. Such possible servers continuously run offloading services in background that can 1) advertise the computational resources available on the server to other devices and 2) enable them to receive and execute tasks from other devices.

The *client* has two network interfaces in our network topology: WAVE/IEEE 802.11p and 5G/mmWave. All other vehicles have only WAVE/IEEE 802.11p interfaces. All edge servers have an only connection to base stations (which also have 5G/mmWave interfaces). All vehicles present the same communication range. All base stations connected to the edge servers also have the same communication range. In the WAVE/IEEE 802.11p interface, we only consider one-hop communications. On the 5G/mmWave interface, if the *client* wants to transmit something to the edge server, it must first transmit to the 5G base station. The latter then forwards the data to the edge server via a wired link. If the edge server wants to transmit something to a vehicle, the reverse path is

taken.

3.2. Communication Model

Below, we present the communication models related to the link lifetime, position prediction based on known routes of vehicles, and data transmission rate.

3.2.1. Link Lifetime

It is possible to estimate how long two neighboring network nodes remain connected within each other's communication range. The estimate is made through kinematic calculations since parameters such as speed, direction, and distance between nodes do not vary significantly [34]. The work in [35] proposed a way to calculate this estimate. For this, we use four parameters: p_x position of the node on the x-axis, p_y position of the node on the y-axis, u_x vector velocity of the node on the x-axis, and u_y vector velocity of the node on the y-axis. We assume that the nodes follow a linear path in a short period. Thus, Equation 1 describes the position of the node i as a function of time t:

$$p_i(t) = \begin{bmatrix} p_{x_i} + u_{x_i} \cdot t \\ p_{y_i} + u_{y_i} \cdot t \end{bmatrix}, \tag{1}$$

where $p_i(t)$ is the position of node i at time t.

We consider that a node j is a neighbor of the node i. Thus, the Equation 2 shows how to estimate the future distance between nodes i and j:

$$\begin{aligned} d_{i,j}^2 = d_{j,i}^2 = \|p_j(t) - p_i(t)\|^2 &= \left(\begin{bmatrix} p_{x_j} - p_{x_i} \\ p_{y_j} - p_{y_i} \end{bmatrix} + \begin{bmatrix} u_{x_j} - u_{x_i} \\ u_{y_j} - u_{y_i} \end{bmatrix} \cdot t \right)^2 \\ &= \alpha_{i,j} t^2 + \beta_{i,j} t + \gamma_{i,j}, \end{aligned} \tag{2}$$

where $\alpha_{i,j} \geq 0$, $\gamma_{i,j} \geq 0$. Then, the future relative distance between j and i is $d_{i,j}(t) = \sqrt{\alpha_{i,j} t^2 + \beta_{i,j} t + \gamma_{i,j}}$.

With this, it is possible to calculate the link lifetime. According to [35] and [36], the link lifetime of the two nodes (i and j) is the estimated time

$t_{link}^{i,j} = t_1 - t_0$, where t_1 is the time when the distance d between the two nodes becomes greater than their communication range R and t_0 is the initial time of the nodes. For the two nodes to be connected, d must be less than or equal to R.

For $t_{link}^{i,j}$ to be calculated, it is necessary to make $d_{i,j}(t) \leq R$. If we square both sides of the inequality and put R on the other side, we get the Equation 3, which gives the value of $t_{link}^{i,j}$:

$$d_{i,j}^2(t) - R^2 = 0 \tag{3}$$

Since we already have an equation that describes $d_{i,j}^2(t)$, we get the Equation 4 below:

$$\alpha_{i,j}t^2 + \beta_{i,j}t + \gamma_{i,j} - R^2 = 0, \tag{4}$$

where t represents $t_{link}^{i,j}$. Thus, to know the value of $t_{link}^{i,j}$, it is only necessary to solve the second degree equation presented in Equation 4.

However, if two nodes have very similar mobility (for example, if they are close to each other, with similar velocities and going in the same direction), t_{link} tends to be infinite. To adjust this question, [37] proposed an upper limit constant for very large values of t_{link}. We defined this constant as $t_{maxlifetime}$ with a value of 100 seconds. That way, if $t_{link} > 100s$, t_{link} becomes 100 seconds.

3.2.2. Position Prediction Based on Known Routes of Vehicles

Public or private vehicles can share information from their on-board navigation systems, such as their trajectories/routes and the estimated arrival time to the destination. With this, it is possible to estimate, with more precision, if two neighboring network nodes will still be connected after a given time $t_{\varkappa}$ [13].

The trajectory or route of a vehicle j is a set of points (geographic coordinates) that will be traveled by it as follows:
$[(p_{x_j}(t_0), p_{y_j}(t_0))\ ;\ (p_{x_j}(t_1), p_{y_j}(t_1))\ ;\ ...\ ;\ (p_{x_j}(t_n), p_{y_j}(t_n))]$,

where $(p_{x_j}(t_0), p_{y_j}(t_0))$ represents the latitude and longitude of the vehicle at the initial time t_0 and $(p_{x_j}(t_n), p_{y_j}(t_n))$ represents the latitude and longitude of the vehicle at the estimated arrival time t_n.

Thus, the distance to be travelled (d_{total}) is calculated as follows:

$$d_{total} = \sum_{\ell=1}^{n} \sqrt{(p_{x_j}(t_\ell) - p_{x_j}(t_{\ell-1}))^2 + (p_{y_j}(t_\ell) - p_{y_j}(t_{\ell-1}))^2}. \tag{5}$$

Therefore, the average vehicle speed is $\overline{u_j} = {}^{d_{total}}/_{(t_n - t_0)}$. Then, the estimated distance traveled by the vehicle j in a given time $t_\varkappa$ is: $d_{est} = \overline{u_j}(t_\varkappa - t_0)$.

For simplicity, we call the estimated position $p_j(t_\varkappa)$ of (φ_x, φ_y). If $t_\varkappa > t_n$, then (φ_x, φ_y) it is the final position of the trajectory, already known. If $t_\varkappa < t_n$, to find (φ_x, φ_y), add the distances between the points of the vehicle's trajectory, from the initial point as in Equation 5, until the sum of these distances is greater than d_{est}. When this happens, we know that (φ_x, φ_y) is approximately on the line between the last and the penultimate points added, respectively (ϖ_x, ϖ_y) and $(\varpi_{x-1}, \varpi_{y-1})$. With these two points, we calculate the general equation of the line as a function of (φ_x, φ_y) as follows:

$$(\varpi_y - \varpi_{y-1})\varphi_x + (\varpi_x - \varpi_{x-1})\varphi_y + \varpi_x\varpi_{y-1} - \varpi_{x-1} - \varpi_y = 0, \tag{6}$$

In addition, the distance between (φ_x, φ_y) and $(\varpi_{x-1}, \varpi_{y-1})$ is d_{est} minus the distance from the initial point of the vehicle's trajectory until $(\varpi_{x-1}, \varpi_{y-1})$, which we call d_ϱ. Thus, we can obtain another equation as a function of (φ_x, φ_y):

$$\sqrt{(\varphi_x - \varpi_{x-1})^2 + (\varphi_y - \varpi_{y-1})^2} = d_{est} - d_\varrho \ . \tag{7}$$

Combining the Equations 6 and 7, we were able to obtain the estimated position of the vehicle j in time $\varkappa$. The same procedures are used to calculate the position of a vehicle i in time $\varkappa$. After this, we calculate the distance between i and j in time $\varkappa$ and check if it is less than the communication range.

3.2.3. Data Transmission Rate

According to [28, 26, 38, 39], in WAVE/V2V communications, the data transmission rate between a node i and a node j at a given time t is given by:

$$r_{V2V}^{i,j}(t) = B_{V2V} \log_2(1 + \frac{P_t L_i^{V2V} |\phi^2|}{\omega}), \tag{8}$$

where B_{V2V} is the bandwidth between i and j, P_t is the transmission power of the node, L_i^{V2V} is the loss of system propagation, ϕ is the fading coefficient of the uplink channel, and ω is the power of the white Gaussian noise.

Thus, the average uplink rate between nodes i and j is given by:

$$\overline{r_{V2V}^{i,j}} = \frac{\int_0^{t_{link}^{i,j}} r_{V2V}^{i,j}(t)\ \mathrm{d}t}{t_{link}^{i,j}}. \tag{9}$$

As [40], for simplicity, we can neglect the time to access the control (CCH) and service (SCH) WAVE channels and the time spent on switching channels, such as the guard interval.

Thus, the time it takes to transmit data of size s from i to j via WAVE/V2V is given by:

$$t_{trans}^{i,j} = \frac{s}{\overline{r_{V2V}^{i,j}}} + \frac{d^{i,j}}{u_{prop}} + t_{other}, \tag{10}$$

where u_{prop} is the propagation speed over the wireless medium and t_{other} are estimates of possible network and queue delays.

For 5G/V2I communications, the authors in [41, 42, 43] show that the data transmission rate between a node i and a node j in a given time t is given by:

$$r_{V2I}^{i,j}(t) = B_{V2I} \log_2(1 + \frac{P_t d_{i,j}^{-\epsilon} |\phi^2|}{\omega}), \tag{11}$$

where B_{V2I} represents the bandwidth between i and j, $d_{i,j}$ is the distance between i and j and the factor ϵ is the exponent of propagation loss of the system. Due to the fast transmission rates on the wired link and the coexisting

deployment between the base station and the edge server, the transmission delay of the wired link is neglected in this work [43].

In this way, the average uplink rate between nodes i and j is given by:

$$\overline{r_{V2I}^{i,j}} = \frac{\int_0^{t_{link}^{i,j}} r_{V2I}^{i,j}(t)\ \mathrm{d}t}{t_{link}^{i,j}}. \tag{12}$$

Thus, the time it takes to transmit data of size s from i to j, via 5G/V2I, is given by:

$$t_{trans}^{i,j} = \frac{s}{\overline{r_{V2I}^{i,j}}} + \frac{d^{i,j}}{u_{prop}} + t_{other}. \tag{13}$$

3.3. Computation Model

The vehicles have different computational capacities and CPU availability over time. The same is true with edge servers. Each vehicle or edge server has a computational capacity C_v and C_e, respectively.

We consider a computationally intensive and real-time application that generates a workload or set $T = \{\tau_1, \tau_2, \tau_3, ..., \tau_{n_\tau}\}$ of tasks. Each task $\tau \in T$ can be processed locally by the *client* or remotely in an independent, asynchronous, and parallel way. Thus, each workload can have its tasks distributed for local or remote execution (on the edge servers or in other vehicles) or in both local and remote environments. In addition, each task $\tau \in T$ is a tuple composed of the following parameters $\{c_\tau, s_\tau^{up}, s_\tau^{down}\}$, where c_τ indicates the total number of CPU cycles required to execute the task τ, s_τ^{up} shows the data size for upload of τ (which includes input parameters, the code to be executed and information about the device that generated the task) and s_τ^{down} shows the size of the processing results of τ for download (which includes information on which device the results should be sent to). s^{up} and s^{down} are known for the history of the application analyzed by the system. Each task is executed with 100% of the CPU available for task executions. Each CPU available from any node on the network can only execute one task at a time. Tasks are placed in a queue and taken out in First In, First Out (FIFO) model to be executed.

Next, we can see the details of the computational modeling of the execution for each environment [28].

3.3.1. Local Computation

When the *client* chooses to execute a task locally, the local execution delay of the *client* is set to t_{client}. C_{client} is described as the computational capacity of the *client* node (in CPU cycles per second). t_{client} consists of two parts: 1) queue delay (there may be other tasks waiting to be executed or in execution) and 2) processing delay of the task on the CPU. The queue delay is given by:

$$t_{client}^{queue} = \sum_{w=1}^{g} \frac{c_w}{C_{client}} \ , \tag{14}$$

where w represents each task in the queue of the *client* node and g represents the number of tasks that were already waiting in the queue or executing.

The processing delay of a task τ on the *client* CPU is given by:

$$t_{client}^{proc} = \frac{c_\tau}{C_{client}} \ . \tag{15}$$

Thus, the local execution delay in the *client* node of a task $\tau \in T$ is given by:

$$t_{client} = t_{client}^{queue} + t_{client}^{proc} \ . \tag{16}$$

3.3.2. Remote Server Computation

A *client* vehicle generates a task and send it to execute on a remote server. As part of vehicular edge computing, the *server* is a vehicle or edge server. Such a *server* executes the task and returns the result of the execution to the *client* vehicle that generated the task.

Thus, the delay in executing the task on the *server* is divided into four parts. The first part is the upload of s_τ^{up} from the *client* vehicle to the *server*. If the upload is via WAVE/V2V, the delay is given by Equation 10. If the upload is via 5G/V2I, the delay is given by Equation 13.

The second part is the waiting time for the task in the *server* queue t_{server}^{queue} [44, 19]. To be processed, the task must wait for all tasks that were already waiting or executing on the server to finish executing. This time is given by:

$$t_{server}^{queue} = \sum_{x=1}^{q} \frac{c_x}{C_{server}} \ , \tag{17}$$

where x represents each task in the queue of the *server* node and q represents the number of tasks that were already waiting in the queue or in execution.

The third part consists of the time it takes to process a task τ:

$$t_{server}^{proc} = \frac{c_\tau}{C_{server}} \ . \tag{18}$$

The fourth part of the delay is the time it takes to transmit the processing result (s_τ^{down}) from the *server* to the *client*. If the transmission is via WAVE/V2V, the time is given by Equation 10. If it is via 5G/V2I, the time is given by Equation 13.

Thus, the delay for executing a task $\tau \in T$ on a remote server is given by:

$$t_{server} = t_{client}^{upload} + t_{server}^{queue} + t_{server}^{proc} + t_{client}^{download} \ . \tag{19}$$

3.3.3. Total Execution Time

As we consider that workload has a total of n_τ tasks, they are distributed to be executed locally and on remote servers (vehicles or edge servers).

So, we assume that tasks are distributed as follows:

- W tasks are distributed to the *client*;
- X_1 tasks are distributed to the $server_1$, X_2 to the $server_2$, X_3 to the $server_3$ and so on up to X_k to the $server_k$, so that $X_1+X_2+X_3+...+X_k = X$;

Thus, the total time required to execute tasks locally is given by:

$$t_{client}^{total} = \sum_{w=1}^{W} t_{client}^{w} \quad , \tag{20}$$

where w represents each task distributed to the *client* node, and W represents the total number of tasks distributed to the *client*.

In turn, the total time required to execute tasks on remote servers is given by:

$$t_{servers}^{total} = max\left\{ \sum_{x_1=1}^{X_1} t_{server_1}^{x_1}, \sum_{x_2=1}^{X_2} t_{server_2}^{x_2}, ..., \sum_{x_k=1}^{X_k} t_{server_k}^{x_k} \right\} , \tag{21}$$

where x_1 represents each task distributed to the $server_1$, x_2 each task to the $server_2$, x_k each task to the $server_k$; X_1 represents the total number of tasks distributed to the $server_1$, X_2 the total tasks to the $server_2$, and X_k the total tasks to the $server_k$.

Therefore, the total time to execute the workload is described as:

$$t^{total} = max\left\{ t_{client}^{total}, t_{servers}^{total} \right\} . \tag{22}$$

3.4. Problem Formulation

The objective of this work is to minimize the execution time of a vehicular application through the computation offloading process in vehicular edge computing systems, satisfying reliability restrictions. So the problem can be formulated as follows:

$$\begin{aligned} P1: \ & min \ t^{total} \\ & s.t. \ C1: W + X = n_\tau \ , \\ & \quad C2: \sum_{x_k=0}^{X_k} t_{server_k}^{x_k} < t_{link}^{client,server_k} \\ & \qquad \vee \ (\ d_{client,server_k}(t_{server_k}^{x_k}) < R \ ) \ . \end{aligned}$$

Thus, problem P1, identified as an NP-hard problem [9], involves finding a way to assign computation tasks to different servers in order to minimize t^{total}. The value of t^{total} is calculated according to the Equation 22. This equation

takes into account the total time required to execute tasks locally (t_{client}^{total}) and the total time required to execute tasks on remote servers ($t_{servers}^{total}$). The value of t_{client}^{total} depends on the tasks assigned to the client, the queue time, and the processing capacity of the client, as shown in Equation 20 and the equations in Section 3.3.1. The value of $t_{servers}^{total}$ depends on the time to execute tasks on each server. In turn, this time depends on the tasks assigned, the processing capacity of the servers, their queue times, and upload and download times, as presented in Equations 19 and 21. The upload and download times follow the equations in Section 3.2.3, which consider the network bandwidth and tasks size, as well as other parameters.

Regarding the constraints of problem P1, C1 ensures that the sum of the total tasks executed on the client (W) and the total tasks executed on remote servers (X) is equal to the number of tasks in the application workload (n_τ). The constraint C2 means that the time $t_{server_k}^{x_k}$ to execute a workload x_k on a given $server_k$ needs to be less than the link lifetime between *client* and $server_k$, according to Equation 4. Or, if the *client* and $server_k$ routes are known, the distance between their estimated positions in time $t_{server_k}^{x_k}$, following the Equations 6 and 7, must be less than the communication range R.

As mentioned before, the value of $t_{server_k}^{x_k}$ used in the C2 constraint is calculated according to the Equation 19. Such equation considers several parameters such as the characteristics of the tasks assigned, the bandwidth to transmit the tasks (see Section 3.2.3), the server's computational capacity, and its queue time (see Section 3.3.2). With the constraint C2, the problem formulation aims to contribute to greater reliability of the computation offloading process. This constraint makes the *client* aware of possible servers' mobility. It also prevents the *client* from choosing a possible $server_k$ that goes out of its communication range without completing the computation offloading process.

Although we consider several parameters in the problem formulation, we do not consider the storage capacity of devices in constraints for two reasons. The first is that the servers discard the tasks, freeing up storage space after proper processing and returning the results to the client. The second is that,

unlike data offloading (or caching), the storage space required in computation offloading is typically small [45].

4. Proposed Framework

In this section, we present the proposed framework for vehicular systems. It manages all stages of the computation offloading technique to minimize the execution time of vehicular applications reliably, according to the objective and restrictions of problem P1. The main part of this framework, the task assignment algorithm, is responsible for providing good and feasible solutions to the problem at hand.

Next, we describe in detail the architecture of the proposed framework and its managed computation offloading process.

4.1. Framework Architecture

Figure 2 shows the *Application* and *Partitioner* modules and conceptual architecture of the proposed framework.

The *Application* module represents the applications running on the vehicle's operating system. As seen in Figure 2, it sends data to a *Partitioner* module in order to analyze whether the application workload can be partitioned. If it is possible, the *Partitioner* divides the application workload into smaller tasks that can be executed on different devices and in a parallel, asynchronous, and independent way. The application workload then moves to the *Decision Maker* module. After the workload has been processed, the *Application* receives the results through the *Local Execution* or the *Data and Context Gatherer* module, when the results come from remote devices. Information from local sensors and other devices is also captured through the *Data and Context Gatherer* module.

Next, we describe each module of the proposed framework.

4.1.1. Sensors

The *Sensors* module is responsible for sending to the *Data and Context Gatherer* module local information such as location (via Global Positioning

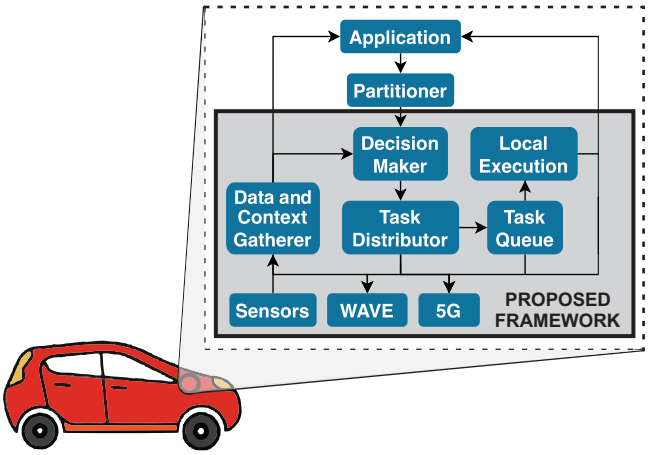


Figure 2: *Application* and *Partitioner* modules and conceptual architecture of the computation offloading framework proposed for vehicular systems. The box with dotted lines represents an application running on a vehicle, along with the *Partitioner* module and the proposed framework. The latter is represented by the smaller box with a gray background. The arrows indicate the direction of the information flow between the modules.

System), speed, and direction.

4.1.2. Task Queue

Tasks from remote devices or the client are placed in the *Task Queue* by the *Task Distributor* module. After a task passes through the queue, it goes to the *Local Execution* module. Information about the storage capacity of the queue and the number of tasks in it is periodically passed on to the *Data and Context Gatherer* module.

4.1.3. Local Execution

With CPU and other resources, the *Local Execution* module process tasks that come from the *Task Queue* with the processing delays described in Section 3.3. Through information contained in the task, this module checks whether it is local or came from a remote device. Then, it can forward the processing result to the local application or to remote devices (via the *WAVE* or *5G* modules).

4.1.4. WAVE and 5G

The *WAVE* module is responsible for sending and receiving messages from the WAVE network via V2V. The *5G* module is responsible for sending and receiving messages from the 5G network via V2I.

These modules can send periodic signaling and safety messages and other data to remote devices. Upon receiving messages, these modules forward them to the *Data and Context Gatherer* module. These messages can be: requests to execute a task to a remote device (offloading request), replies to offloading requests (offloading reply), tasks, data downloaded to a running application, and context data from other devices.

4.1.5. Task Distributor

This module analyzes the tasks received from the *Decision Maker* and the location where they will be executed. If the decision is to execute a task locally, the *Distributor* forwards it to the local *Task Queue*. If the decision is to run a task remotely, the *Distributor* sends it to the *WAVE* or *5G* modules (or both), depending on the choice made by the *Decision Maker*.

This module is also responsible for storing a backup of tasks that have been offloaded to remote servers. Thus, upon being informed that the remote server has failed to return the processing result, this module then sends the stored copies of the lost tasks to be executed locally.

4.1.6. Data and Context Gatherer

Through the *WAVE* and *5G* modules, the *Data and Context Gatherer* can receive messages involving application data or remote processing results. Then, the *Gatherer* forwards it to the local *Application* module. If the received message involves remote contextual information, this module forwards it to the *Decision* module. If the *Gatherer* receives periodic local contextual information through the *Sensors*, the *Task Queue* and the *Local Execution* module (CPU capacity), this module forwards it to the *Application* module or to the *Decision* module. Upon receiving an offloading request, a reply for an offloading request, or a task

to be executed, this module forwards them to the *Decision Maker* module.

This module also monitors possible offloading failures. This is done by analyzing the signaling messages received from the chosen remote servers and verifying their connectivity with the client. After failure detection, the *Gatherer* notifies the *Decision Maker* that triggers the *Task Distributor*. More details of failure handling are described in Section 4.2.4.

4.1.7. Decision Maker

The *Decision Maker* module receives tasks from the *Partitioner*, starts resource discovery, and gathers all real-time contextual information (local and remote) from the *Data and Context Gatherer*. Then, the *Decision Maker* module calculates additional information such as link lifetime, distances between devices, transmission and processing time, and whether the client and server will be within the communication range of each other at a specific time.

With all this information, this module assigns, through a greedy algorithm (see Section 4.2.2), tasks for the chosen devices to execute. Subsequent, it informs the task assignment decision to the *Task Distributor* module.

This module also receives other types of messages. For example, when receiving offloading requests, the *Decision Maker* triggers the *Distributor* to reply to the requesting device agreeing to process its tasks. Upon receiving a positive reply for an offloading request, this module sends the tasks to the remote device through the *Distributor* and *WAVE* or *5G* module. Upon receiving a task to be executed, it forwards it to the local *Task Queue*, through the *Distributor* module.

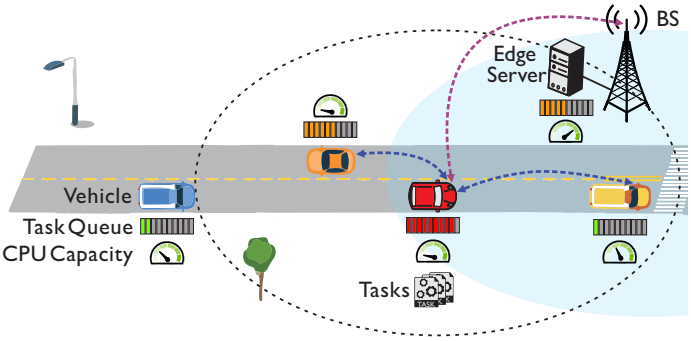
4.2. Computation Offloading Process

This Section presents the computation offloading process managed by the proposed framework. The process consists of four main parts: 1) Resources discovery, 2) Offloading decision, 3) Send/receive tasks, and 4) Failure recovery.

4.2.1. Resources Discovery

The computation offloading process starts when the *Decision Maker* module receives tasks from the *Application/Partitioner*. Then, through the *Distributor*, *WAVE* and *5G* modules, it triggers the resources discovery. At this time, as shown in Figure 3, the client sends a one-hop WAVE request via broadcast to the other vehicles and sends a 5G request via unicast to the edge server within the respective communication ranges of each technology. These requests are used to gather real-time contextual information from possible servers and allow the *Decision* module to provide feasible solutions to problem P1.

Figure 3: Client (red) discovering computational resources of vehicles within its WAVE communication range (black dotted line) via V2V connections (blue dotted lines) and of an edge server whose base station is within its 5G communication range (light blue background) via V2I connection (purple dotted line). The client has three tasks to be executed, its computational capacity is low (pointer almost at least), and its task queue for execution is almost full (bar with small red blocks).



After the client´s first contact, each possible server replies with a tuple containing its location, data rate and communication range (WAVE or 5G), CPU capacity, and task queue condition. If it is a vehicle, it also sends its speed, direction, and, if available, its estimated arrival time to the destination and its complete route. This reply reaches the client via the *WAVE* and *5G* modules, which forwards it to the *Data and Context Gatherer* module, which in turn sends it to the *Decision* module. Subsequent waiting ψ milliseconds, the client discovers possible servers and proceeds to the task assignment decision.

4.2.2. Offloading Decision

The decision process is the core of the framework. This step is also the most complex because the optimal distribution of tasks for maximum reduction in application execution time is an NP-hard problem [9]. Hence, no algorithm provides an exact solution for this type of problem in a polynomial time. Thus, we propose a greedy heuristic algorithm called Greedy Task by Task (GTT), which delivers reasonable solutions to the problem in a feasible time, although it does not guarantee to find the optimal solution [46].

After congregating all the necessary information, the *Decision Maker* module executes the GTT to decide the assignment of tasks. The GTT strategy is to build task assignment solutions in stages considering the problem P1 and contextual information. Accordingly, the algorithm tries to minimize the execution time of each task by assigning them to the best possible server at the moment (local or remote). With this strategy, the GTT provides feasible solutions to the problem P1. After finding a solution, the framework allows tasks to be sent simultaneously to edge servers (via 5G) and neighboring vehicles (via WAVE). We present the GTT's pseudocode in Algorithm 1.

GTT receives as input a set M of reply messages and a set T of tasks to be processed. In line 1, GTT initializes the sets S (servers), W (tasks to be executed locally), Y (backup of T) and variable i (*client* ID). In line 2, the set F of feasible servers is initialized with a tuple from the *client*, indicating that it can also execute tasks. Afterwards, in line 3, F receives more feasible servers on return from the Function *AddFeasibleServers* (Algorithm 2).

Then, in line 4, the set F of feasible servers is sorted in increasing order according to the distance to i, computational capacity, queue time, and the constants σ, ρ, and δ. These parameters are some of the most used in computation offloading processes [7]. In particular, the computational capacity and the queue time directly influence the task execution time. Distance is also important to avoid failures and delays due to retransmissions. In fact, the closer a server is to the client, the less likely it is to experience failures caused by interference,

Algorithm 1: GTT Algorithm

Input: M, T

Output: $result$

1 $S \leftarrow \emptyset$; $W \leftarrow \emptyset$; $Y \leftarrow T$; $i \leftarrow getClientId()$;

2 $F \leftarrow \{i; d_{ii}; t_i^{queue}; C_i; 0; 0; \emptyset\}$;

3 $AddFeasibleServers(M, F, i)$;

4 Sort F by $(d * \sigma + \frac{1}{C} * \rho + t^{queue} * \delta)$;

5 **foreach** $\tau \in T$ **do**

6 $z \leftarrow getIdOfFirstElement(F)$;

7 **if** $z = i$ **then**

8 $AddTasksToClient(i, \tau, F, T)$;

9 **else**

10 $AddTasksToServer(z, \tau, F, T)$;

11 **if** $(T = \emptyset)$ **then**

12 $Process(W)$; $SendWorkloads(S)$;

13 **return** $true$;

14 **else**

15 $Process(Y)$;

16 **return** $false$;

propagation loss, and 5G signal blocking [47, 48]. Thus, GTT prioritizes servers with high computational capacities, low queue times, and short distances to the client to minimize application execution time and avoid failures.

In loop of lines 5-10, T is traversed task by task so that each task τ is assigned to a server (local or remote). The evaluation is always done by the first element of the sorted set F, that is, the best-evaluated server at the moment. In lines 7-10, if the first element is the client itself, the Function *AddTasksToClient* is called (Algorithm 3). Otherwise, the Function *AddTasksToServer* is called (Algorithm 4).

Finally, in lines 11-16, if there are no tasks left in T, the *client* processes tasks in W locally, and the other tasks are sent to be executed in servers in S. If tasks remain, a problem has occurred, and the initial set of tasks (Y) is all executed locally.

Function AddFeasibleServers. This Function checks each reply message m received from a possible server j. This check is made to know if j is feasible to execute tasks for the *client*. In line 3 of the Algorithm 2, t_j receives the estimated time to execute the task $\tau \in T$ on the possible server j. This estimate, according to Sections 3.2.3 and 3.3.2, considers several contextual parameters such as bandwidth, computational capacity, queue time, and others. In line 4, a check is made if t_j is less than the estimated link lifetime between i and j (partial constraint C2 of problem P1). This estimated link lifetime is calculated according to Section 3.2.1. This calculation considers the communication ranges of the devices (WAVE or 5G) and that edge servers are static. Then, if the check of line 4 returns *true*, F receives a tuple of the possible server j. This tuple contains, among other things, how much it will process (h_j), how much data it will receive and return (a_z), and the set of tasks that will process (X_j), all initialized as zero or empty.

Algorithm 2: *AddFeasibleServers* Function

Input: M, F, i

1 **foreach** $m \in M$ **do**
2 $\quad j \leftarrow getServerId(m)$;
3 $\quad t_j \leftarrow calcServerTime(\tau, r^{i,j}, d_{ij})$;
4 $\quad$ **if** $(t_j < t_{link}^{i,j})$ **then**
5 $\quad\quad h_j \leftarrow 0$; $a_j \leftarrow 0$; $X_j \leftarrow \emptyset$;
6 $\quad\quad F \leftarrow F \cup \{j; d_{ij}; t_j^{queue}; C_j; h_j; a_j; X_j\}$;

Function AddTasksToClient. It is responsible for assigning tasks to the *client*. In Algorithm 3, W receives τ, that is removed from T. Then, the

queue of i is increased by the time to process τ, as if τ was already queued in i and considering the queue time and computational capacity. Finally, F is sorted again.

Algorithm 3: *AddTasksToClient* Function

Input: i, τ, F, T

1 $W \leftarrow W \cup \tau$;

2 $T \leftarrow T - \tau$;

3 $t_i^{queue} \leftarrow t_i^{queue} + \frac{c_\tau}{C_i}$;

4 Sort F by $(d * \sigma + \frac{1}{C} * \rho + t^{queue} * \delta)$;

Function AddTasksToServer. It manages the task assignment to remote servers. In Algorithm 4, line 1, as the server z will process the task τ, how much it will process (h_z) receives an addition of c_τ, how much data on the server z will be received and returned (a_z) receives an addition of s_τ^{up} and s_τ^{down}. In line 2, t_z receives the estimated time for server z to execute its tasks. According to Sections 3.2.3 and 3.3.2, this estimate considers several contextual parameters such as bandwidth, computational capacity, queue time, in addition to h_z and a_z. In lines 3-4, if the route of the server is known (K_z), l receives true if i and z are still within the communication range of each other after t_z seconds. This last check is done through Function *withinRange*, following the calculations in Section 3.2.2. Function *withinRange* assumes that K_z is always *true* if z is an edge server because it is static. In addition, the values of the WAVE and 5G ranges are also taken into account.

In lines 5-15, the algorithm checks whether the server can execute its assigned tasks (constraint C2 of the problem P1). With the estimate of the execution time of the assigned tasks, the verification is done by predicting positioning, if the route is known, or by the estimated link lifetime (considering the WAVE and 5G ranges of the devices and the immobility of edge servers). Suppose the check returns *true* (lines 5-11). Then, if the server is not in S, it is added. Next, X_z receives τ, τ is removed from T, the task queue of the server is increased

Algorithm 4: *AddTasksToServer* Function

Input: z, τ, F, T

1 $h_z \leftarrow h_z + c_\tau$; $a_z \leftarrow a_z + s_\tau^{up} + s_\tau^{down}$;

2 $t_z \leftarrow calcServerTime(h_z, a_z, r^{i,z}, d_{iz})$;

3 **if** $(K_z = true)$ **then**

4 $\quad l \leftarrow withinRange(i, z, t_z)$;

5 **if** $((K_z = true \ and \ l = true) \ or \ (t_z < t_{link}^{i,z}))$ **then**

6 $\quad$ **if** $(z \notin S)$ **then**

7 $\quad\quad S \leftarrow S \cup z$;

8 $\quad X_z \leftarrow X_z \cup \tau$;

9 $\quad T \leftarrow T - \tau$;

10 $\quad t_z^{queue} \leftarrow t_z^{queue} + \frac{c_\tau}{C_z}$;

11 $\quad$ Sort F by $(d * \sigma + \frac{1}{C} * \rho + t^{queue} * \delta)$;

12 **else**

13 $\quad h_z \leftarrow h_z - c_\tau$; $a_z \leftarrow a_z - s_\tau^{up} - s_\tau^{down}$;

14 $\quad t_z^{queue} \leftarrow t_z^{queue} + \varsigma$;

15 $\quad$ Sort F by $(d * \sigma + \frac{1}{C} * \rho + t^{queue} * \delta)$;

by the time to process τ (as if the task τ was already queued at *server*), and F is sorted again. Suppose the check returns $false$ (lines 12-15). In that case, the procedures for executing τ are undone, the queue of the server receives a high value constant ς so that the server stays in the last positions of F, and F is sorted again.

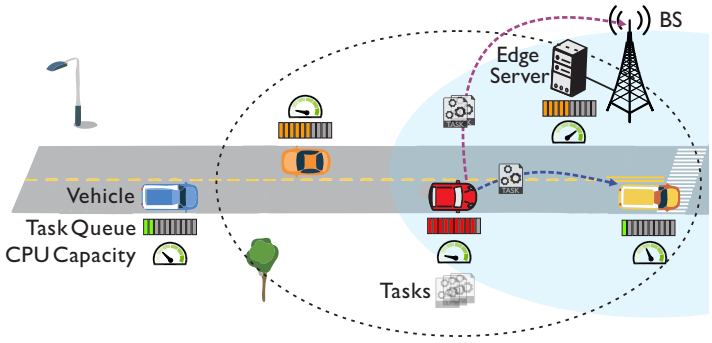
Complexity Analysis. The most critical computational complexity aspect of the GTT algorithm is the loop of lines 5-10, which depends on the size of T. In each iteration of the loop, the algorithm executes the Function *AddTasksToClient* or the Function *AddTasksToClient*. Each of these functions has computational complexity determined mainly by the sorting of the set of feasible servers F, which is $O(F \ log \ F)$ on the size of the set F. Since these sortings are

done at each iteration of the loop of lines 5-10, the computational complexity of the GTT algorithm is $O(TF\ log\ F)$.

4.2.3. Send/Receive Tasks

Later the decision, the framework puts into practice the solution proposed by the GTT algorithm. Thus, the *Decision Maker* module transfers the tasks to the *Distributor* module that forwards the tasks to the appropriate modules (for local or remote execution). If the option is to send some tasks to remote devices, they are sent to the *WAVE* or *5G* modules. In this way, as shown in Figure 4, tasks are distributed to remote servers (vehicle and edge server) to start processing. Thus, multiple servers can simultaneously collaborate to provide computing services to the client.

Figure 4: Client (red) sends tasks to the chosen servers. Two tasks are sent to be executed on the edge server (whose base station is within its 5G communication range - light blue background) via V2I connection (purple dotted line). One task is sent to be executed on the vehicle ahead (within its WAVE communication range - black dotted line) via V2V connection (blue dotted line).



Following processing, each chosen server returns the processing result to the client. This result comes through the *WAVE* or *5G* modules. Then, the result passes through the *Data and Context Gatherer* module and finally reaches the *Application* to continue its execution.

4.2.4. Failure recovery

After the client sends the tasks to the remote servers, it triggers a function to monitor the connectivity between client and servers, through the *Data and Context Gatherer* module. If beacon messages from a server continue to be received by the client, connectivity still exists. If no beacon messages arrive from a server in ξ milliseconds, and if the processing result from that server has not yet been returned, a failure is detected. In this case, the *Data and Context Gatherer* module informs the *Decision Maker* module. The *Decision Maker* module activates the *Task Distributor* module that has the tasks of the lost server and distributes it to the *Task Queue* to be executed locally on the client.

5. Experiments

This section presents the details of the experiments performed to evaluate the proposed framework and algorithm. All experiments follow the general network structure, communication, and computation models described in Section 3. Table 4 presents a summary of the main characteristics of the experiments. Aspects of network, mobility, scenario, vehicular density, application, and comparison with other algorithms are detailed below.

5.1. Network

We employ simulation-based experiments to evaluate the proposed framework. For this, we used the ns-3 simulator [49] (version 3.29) running on a computer with an Intel Xeon E5645 processor @ 2.40GHz and 32 GB RAM. All scenarios use common packet traffic on vehicular networks.

5.2. Scenario

To build the simulated scenarios, we use Simulation of Urban MObility (SUMO) [50]. The first scenario, seen in Figure 5, consists of a highway in a metropolitan area with the following characteristics: 5km long, two lanes in each direction, and a maximum speed of 60km/h.

Table 4: Main simulation parameters.

General	
Scenarios	Highway and Urban
# of vehicles in highway	11, 55, and 120 per km
# of vehicles in urban	25, 120, and 250 per km^2
Simulation time	150 seconds
# of simulations carried out	200 times
Mobility model	Krauss
Servers offering offloading	All (vehicles and edge servers)
Known Routes of Vehicles	50%
CPU Capacity of Edge Servers	1.5, 2.0, and 2.5 GHz
CPU Capacity of Vehicles	0.5 and 1.0 GHz
CPU Requirement of a Task	3.5 Gigacycles
Task size (upload)	558 KB
Result size (download)	1000 B
Workload type	ALPR
Workloads	4, 6, 8, 10, and 12 tasks
Transport protocol	UDP (discovery), TCP (offloading)
Others packets traffics	Beacon messages
WAVE	
Communication range	250 meters
Radio propagation model	Two-Ray Ground
Layer 2 protocol	IEEE 802.11p
Data rate	27 Mbps
5G	
Communication range	220 meters
Radio propagation model	5G mmWave systems [4]
Layer 2 protocol	5G mmWave systems [4]
Data rate	450 Mbps

The second scenario, seen in Figure 6, consists of an urban area with $2km^2$ area and a maximum speed of 60km/h.

Figure 5: Highway scenario used in the experiments. An adapted stretch of a Brazilian highway with returns at the ends. The black dotted lines highlight the smaller section showing vehicles and a 5G base station with a coupled edge server.

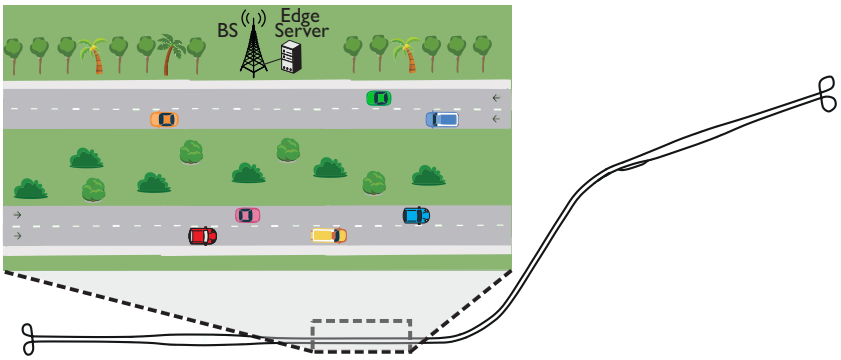
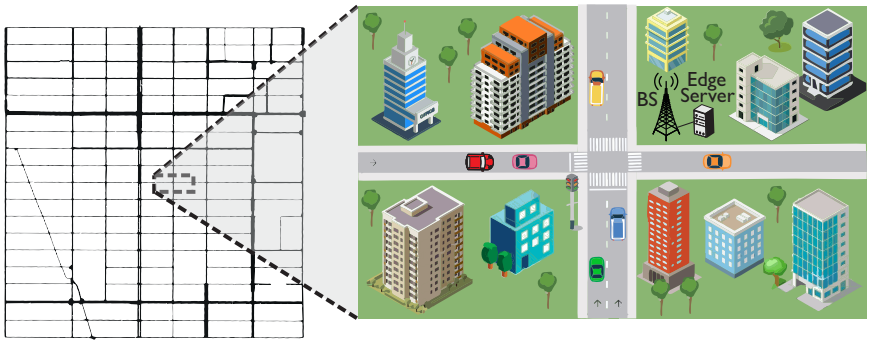


Figure 6: Urban scenario used in the experiments. An adapted stretch of the Manhattan region, New York, USA. The black dotted lines highlight an intersection showing vehicles and a 5G base station with a coupled edge server.



5.3. Mobility

We consider three types of nodes with predefined quantity: vehicles, edge servers, and base stations. Base stations and edge servers have a fixed location and are spatially distributed to provide complete communication coverage.

Concerning vehicles, their initial positions are spatially and randomly distributed across different points in the scenario. They follow random paths with different starting and ending points. The vehicles move at different speeds and directions, according to the microscopic car-following model of Krauss. Thus,

the vehicle's speed depends on the maximum speed of the road, the speed of the vehicle ahead (if any and not to collide), the difference in vehicle positions, and static parameters such as the driver's reaction time [50, 51].

The simulator used to generate the mobility of the network nodes was the SUMO. It was chosen for having mobility models that reflect the real-world behavior of vehicular traffic and integration mechanisms with network simulators. Thus, most computation offloading experiments in vehicular scenarios utilize simulation-generated mobility, with SUMO being one of the most used simulators [9, 50].

5.4. Vehicular Density

For each scenario, we use three types of vehicular density: low, medium, and high. Vehicular density is considered low if it has approximately 11 vehicles/km in highway scenario and 25 vehicles/km^2 in urban scenario. In medium density are there are approximately 55 vehicles/km in highway scenario and 120 vehicles/km^2 in urban scenario. Finally, in high density, we have approximately 120 vehicles/km in highway scenario and 250 vehicles/km^2 in urban scenario [9].

5.5. Application

The application used was Automatic License Plate Recognition (ALPR), which consists of image capture, vehicle detection, plate detection, and optical character recognition. For each captured image, we used SSD-300 with MobileNet [52] to detect vehicles and Tiny YOLOv3 [53] to identify the license plate. Then, we used an algorithm of optical character recognition to recognize the characters. As this application's tasks are compute-intensive, the vehicles can offload them to be executed on remote servers (edge servers or other vehicles) to reduce the execution time.

In our experiments, we considered an ALPR task (τ) involving independent images with a total number of necessary CPU cycles equal to 3.5 Gigacycles ($c_\tau = 3.5Gc$), size of the upload image equal to 558KB ($s_\tau^{up} = 558KB$) and

the size of the processing results equal to 1000B ($s_{\tau}^{down} = 1000B$, representing strings containing each recognized license plate). We also consider five different types of ALPR workloads. Workloads 1, 2, 3, 4 and 5 have, respectively, 4, 6, 8, 10 and 12 tasks of type τ.

As it is not possible to simulate the workload processing in ns-3, we use real offloading experiments. In this way, we calculate the execution times of ALPR tasks and the size of the packets transferred during the offloading process. For edge servers, we use CPUs with capacities of 1.5, 2.0, and 2.5 GHz. For vehicles, we use CPUs with 0.5 and 1.0 GHz.

5.6. Comparison with other Algorithms

We compared the proposed decision algorithm (GTT) with four other algorithms: FIFO (First In, First Out), HVC (Hybrid Vehicular edge Cloud) [32], MDO (Multi-Decision based Offloading) [19], and GCF (Greedy for CPU Free) [33]. All algorithms used the failure recovery mechanism described in Section 4.2.4 and were tested under the same scenarios and conditions. The FIFO algorithm was implemented to select the servers that reply first, sending a task to each one. If the FIFO does not find enough servers, it executes all tasks locally. The HVC, MDO, and GCF were implemented following the descriptions of their respective papers. The HVC algorithm prioritizes sending the requests via 5G (in our scenario, to the edge server). The MDO uses only neighboring vehicles as remote servers, choosing the capable ones and with the best scores (based on the relative speeds between vehicles and waiting times). The GCF algorithm seeks servers close to the client and with the highest free CPU availability (low queue times) in the sequence: edge servers, client, nearby vehicles. However, the GCF does not take into account the computational capacity of the network nodes. In addition, known routes of vehicles are not considered in FIFO, HVC, MDO, and GCF.

6. Results Analysis

This section discusses the main results obtained from the experiments mentioned above.

6.1. Impact of Known Routes of Vehicles on the Number of Recovered Tasks

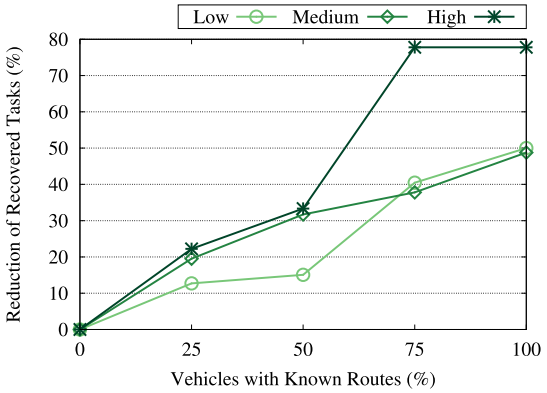
As mentioned earlier, some vehicles can share their routes with other nodes in the network. To analyze the impact of this information shared in computation offloading, we used the urban scenario for having more different route options. Besides, we used the workload with 12 tasks and the algorithm that uses information about known routes (i.e., GTT). Such a number of tasks was chosen because distributing more tasks increases the chance of failure.

In Figure 7, we can see that the more vehicles with known routes (x-axis), the greater the reduction of failures/recovered tasks. For example, in low vehicular density and with 50% and 100% of vehicles with known routes, GTT decreased by 15.1% and 50.0%, respectively, the number of tasks recovered. In medium density, the reduction was from 31.7% and 48.8% to 50% and 100% of vehicles with known routes, respectively. With high vehicular density, having 50% and 100% of vehicles with known routes helped GTT to reduce the number of tasks recovered by 33.3% and 77.8%, respectively.

6.2. Type of Occurrence per Task

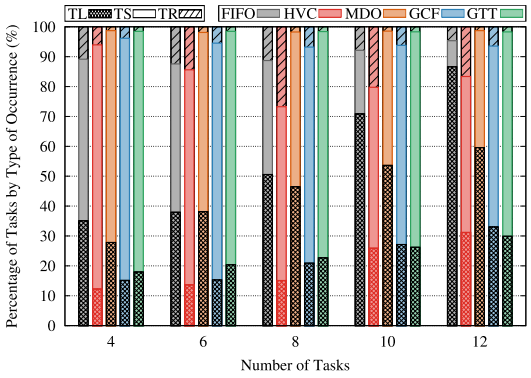
This section presents the performance of the proposed algorithm and others from the literature concerning what happens with each task of the different workloads. Thus, in Figures 8 and 9, a task can have three types of occurrences: TL, TS, or TR. TL (parts of the bars with checkered lines) represents tasks executed locally by the client. TS (part of the bars without lines) represents tasks that were successfully offloaded, executed remotely, and the results were returned to the client. TR (parts of the bars with simple diagonal lines) represents tasks that were offloaded, and there was some failure in the process, causing them to need recovery on the client, as seen in Section 4.2.4.

Figure 7: Impact of known routes of vehicles on the number of failures/recovered tasks with low, medium, and high vehicular density. The baseline is the number of recovered tasks when the percentage of known routes is zero. Scenario: urban, 12 tasks, and GTT algorithm.

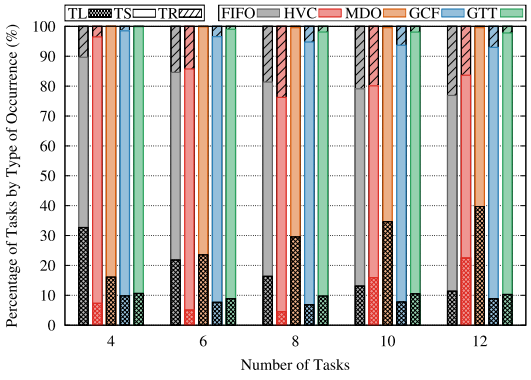


In Figures 8a to 8c, we have the percentage of tasks by type of occurrence of the FIFO, HVC, MDO, GCF, and GTT algorithms in the highway scenarios. We can see that the GTT has the highest percentages of TS and low percentages of TR. With low vehicular density (Figure 8a), the percentage of TS in GTT was on average 75.0%, while in FIFO, HVC, MDO, and GCF, it was, respectively, 34.4%, 63.6%, 53.3%, and 71.9%. The average percentage of TR in GTT was 1.6%, while in FIFO, HVC, MDO, and GCF were, respectively, 9.4%, 16.8%, 1.5%, and 5.8%. In the medium density scenario (Figure 8b), the percentage of TS in GTT was on average 88.6%, while in FIFO, HVC, MDO, and GCF it was, respectively, 63.3%, 73.4%, 71.0%, and 87.1%. Regarding the average percentage of TR, that of GTT was 1.4%, while FIFO, HVC, MDO, and GCF were, respectively, 17.7%, 15.6%, 0.3%, and 4.7%. With high vehicular density (Figure 8c), the GTT had a higher average percentage of TS (89.8%) than FIFO (62.3%), HVC (74.9%), MDO (66.0%), and GCF (88.2%). Finally, the average percentage of TR in GTT was 1.4%, while in FIFO, HVC, MDO, and GCF, it was 19.1%, 14.0%, 0.3%, and 4.7%, respectively.

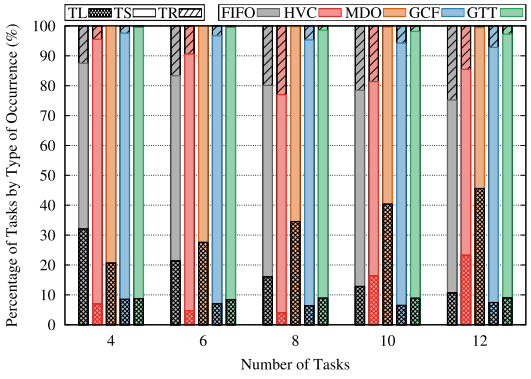
In Figures 9a to 9c, we present the percentage of tasks by type of occurrence of the FIFO, HVC, MDO, GCF, and GTT algorithms in the urban scenarios.



(a) Low vehicular density.



(b) Medium vehicular density.



(c) High vehicular density.

Figure 8: Percentage of tasks by type of occurrence for workloads with 4, 6, 8, 10, and 12 tasks. We can have three types of occurrences: TL - task executed locally; TS - task successfully executed remotely; TR - task executed with recovery. For comparison, we used five algorithms: FIFO, HVC, MDO, GCF, and GTT in *highway scenarios*.

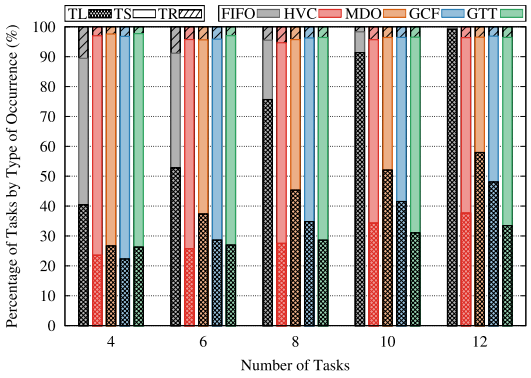
We can also see that GTT has a higher percentage of TS and a lower percentage of TR than other algorithms. In low density scenario (Figure 9a), GTT had, on average, a percentage of TS of 67.6%, while FIFO, HVC, MDO, and GCF had, respectively, 22.9%, 66.1%, 52.5%, and 61.4%. In terms of TR, the GTT's average percentage was 3.2%, the FIFO was 5.2%, the HVC was 4.1%, the MDO was 3.6%, and the GCF was 3.6%. With medium vehicular density (Figure 9b), the average percentage of TS in GTT was 83.8%, while in FIFO, HVC, MDO, and GCF it was, respectively, 60.8%, 77.0%, 75.3%, and 79.2%. The average percentage of TR in GTT was 1.0%, while in FIFO, HVC, MDO, and GCF, it was, respectively, 9.0%, 4.0%, 2.3%, and 3.7%. With high vehicular density (Figure 9c), GTT had an average TS percentage of 88.6%, while FIFO, HVC, MDO, and GCF had, respectively, 66.6%, 80.3%, 81.0%, and 84.7%. Finally, on average, the percentage of TR for GTT was 0.8%, while for FIFO, HVC, MDO, and GCF, it was, respectively, 11.5%, 4.5%, 1.8%, and 3.5%.

6.3. Reduction in Execution Time

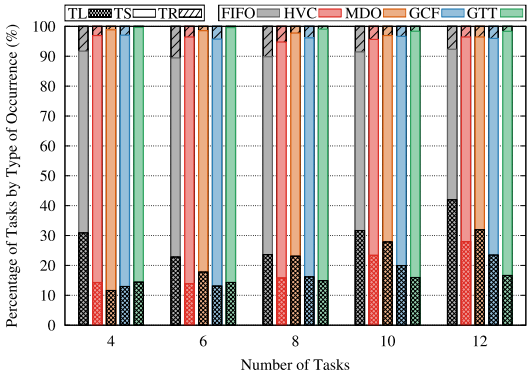
This section compares the performance of each algorithm concerning the metric of reduction in execution time of different workloads. This metric is calculated by comparing each algorithm's time with the baseline, which is the time to execute all tasks locally on the client[1]. The error bars in the figures of this section show the 95% confidence interval of the corresponding data based on the standard normal distribution. Furthermore, the time taken to decide where each task should be processed was around 10 nanoseconds for all algorithms considered.

Thus, in the highway scenarios (Figures 10a to 10c), we can see that the most significant reductions in execution time, in general, are from the GTT algorithm. For example, in the low vehicular density scenario (Figure 10a), on average, GTT algorithm had a reduction in execution time of 59.4%. In contrast, FIFO, HVC, MDO, and GCF algorithms had, respectively, reductions

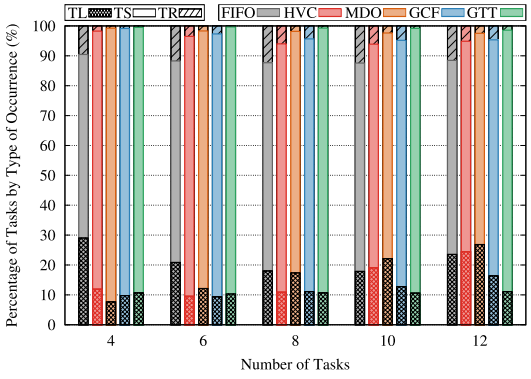
[1] A similar comparison was made with this same baseline in [32].



(a) Low vehicular density.



(b) Medium vehicular density.



(c) High vehicular density.

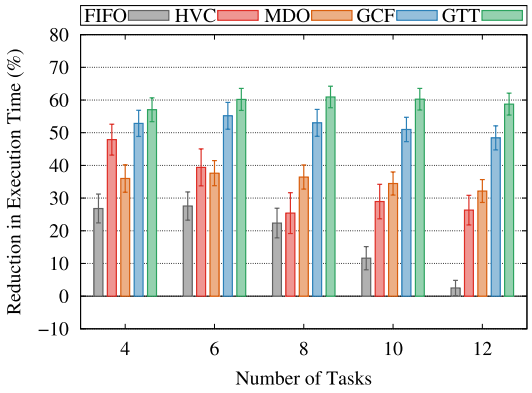
Figure 9: Percentage of tasks by type of occurrence for workloads with 4, 6, 8, 10, and 12 tasks. We can have three types of occurrences: TL - task executed locally; TS - task successfully executed remotely; TR - task executed with recovery. For comparison, we used five algorithms: FIFO, HVC, MDO, GCF, and GTT in *urban scenarios*.

of 18.2%, 33.6%, 35.3%, and 52.1%. In medium density (Figure 10b), GTT had an average reduction of 69.7%, while FIFO, HVC, MDO, and GCF had, respectively, reductions of 37.8%, 44.2%, 50.8%, and 62.9%. With high vehicular density (Figure 10c), GTT had an average reduction of 70.3%, while FIFO, HVC, MDO, and GCF had, respectively, reductions of 36.3%, 45.4%, 49.2%, and 64.0%.

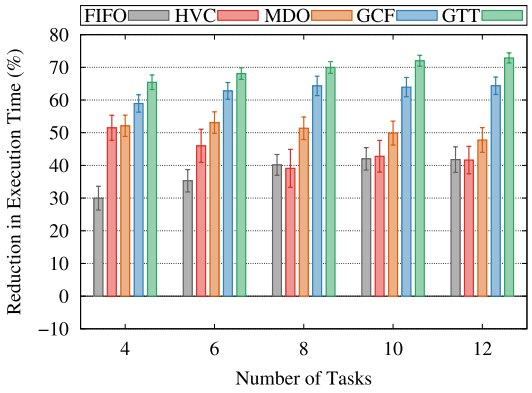
In urban scenarios (Figures 11a to 11c), we can see that the greatest reductions in execution time are also, in general, of the GTT algorithm. For example, in the low vehicular density scenario (Figure 11a), on average, GTT algorithm had a 52.4% reduction in execution time. On the other hand, FIFO, HVC, MDO, and GCF algorithms had, respectively, reductions of 9.5%, 38.8%, 32.1%, and 42.7%. In the medium density scenario (Figure 11b), the average reduction in GTT was 64.8%, while in FIFO, HVC, MDO, and GCF, it was, respectively, 35.3%, 49.6%, 48.2%, and 53.8%. With high vehicular density (Figure 11c), GTT had an average reduction of 68.9%, while FIFO, HVC, MDO, and GCF had, respectively, average reductions of 40.3%, 53.3%, 55.9%, and 59.2%.

6.3.1. Statistical Tests

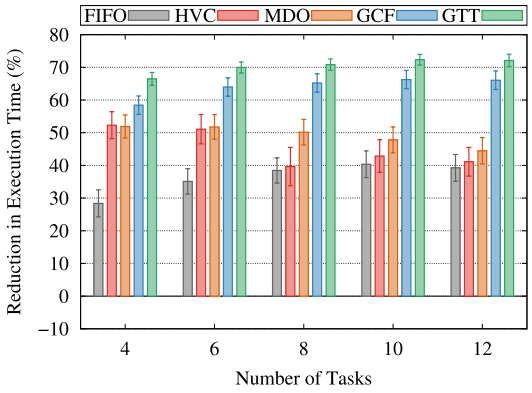
As shown in Figures 10 and 11, GTT has the most significant reductions in task execution time. However, in Figures 10a and 11a, there are overlapping confidence intervals between the GTT and the GCF with four and six tasks. This type of overlapping also exists in Figure 11a between GTT and HVC with four tasks. Then, we performed two-sample t-tests (one-tailed) to assess pairwise whether there are significant differences in the performance of the algorithms in these cases. The null hypothesis ($H0$) is that the averages of the two algorithms evaluated in each case are equal. However, in all tests, we obtained p-values less than 0.05, rejecting $H0$. Therefore, the GTT average reductions in execution time are significantly different from the other algorithms in all evaluated cases.



(a) Low vehicular density.

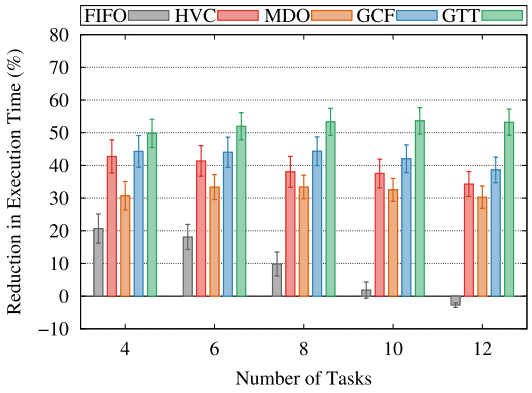


(b) Medium vehicular density.

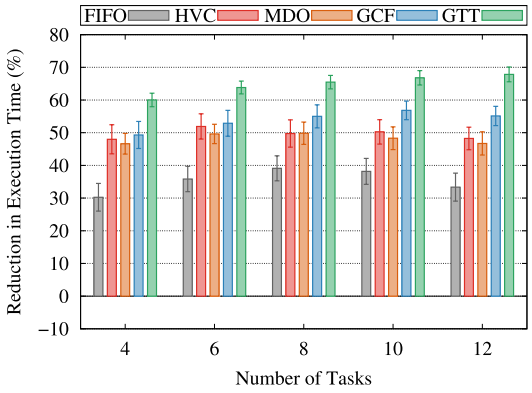


(c) High vehicular density.

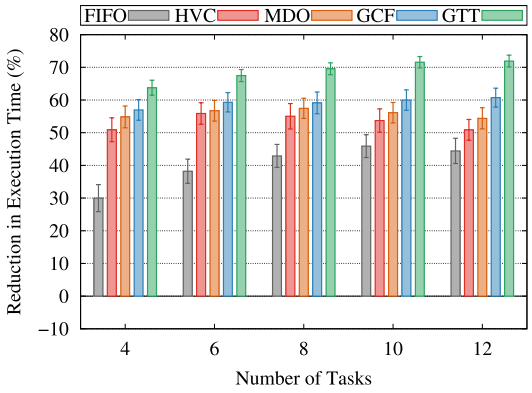
Figure 10: Average reduction in execution time, with 95% confidence interval, for workloads with 4, 6, 8, 10, and 12 tasks. The baseline is the execution time when the workload is fully executed on the client. For comparison, we used five algorithms: FIFO, HVC, MDO, GCF, and GTT in *highway scenarios*.



(a) Low vehicular density.



(b) Medium vehicular density.



(c) High vehicular density.

Figure 11: Average reduction in execution time, with 95% confidence interval, for workloads with 4, 6, 8, 10, and 12 tasks. The baseline is the execution time when the workload is fully executed on the client. For comparison, we used five algorithms: FIFO, HVC, MDO, GCF, and GTT in *urban scenarios*.

6.4. Discussion

According to Figure 7, we can see that, even with different vehicular densities, the information of known routes of vehicles reduces the number of tasks recovered, i.e., the number of failures. This reduction occurs because this real-time contextual information helps GTT to better predict the network nodes' position at a given time. Consequently, GTT is able to calculate more precisely the best server to send the tasks so that failures/recoveries do not happen.

We also note in Figures 8 and 9 that the GTT has the highest percentages of tasks successfully executed remotely (TS) and the lowest percentages of tasks recovered (TR) in most cases. Having a high TS percentage is a good indicator because it relieves the client's processing overload and tasks can be executed more quickly on remote servers with better computing resources. A low TR percentage is also an important indicator. Sending tasks to be performed remotely involves using bandwidth and processing on the remote server. It also involves time to transmit, process, and wait for their results. When a failure occurs, these times and computational resources are wasted. Also, as the result of task processing does not arrive, it is necessary to recover the lost task. So, time is still needed to detect the failure and re-execute the task locally. Thus, a low TR percentage indicates a low percentage of failures, saving time and computational resources.

In Figures 10 and 11, with the most significant reductions in execution time compared to other algorithms, GTT can help an application to execute its tasks more quickly. These reductions are very important in environments as dynamic as vehicular networks. In this sense, the GTT's best time performance is confirmed by Section 6.3.1. In it, we can see that GTT has execution times significantly different from all other algorithms, including GCF (algorithm with performance closer to GTT) in all scenarios. Another interesting point is that the algorithms spent negligible time deciding where tasks should be processed. Hence, this time did not influence the workloads' execution times.

Therefore, the results show that computation offloading processes can reduce an application's total execution time in vehicular systems. We have also seen

that GTT has the greatest reductions in execution times of tasks, the highest rates of tasks successfully offloaded, and the lowest failure rates (with fewer recovered tasks in most cases) compared to other algorithms.

Some factors can explain the better performance of the GTT. First, as a greedy algorithm, GTT periodically updates the best possible server (including the client itself) to execute each task, taking advantage of the computational resources available from all nodes in the network. This evaluation of the best server is done through various real-time contextual information parameters. In this way, with more contextual information, evaluations become more accurate. Consequently, GTT is able to make better offloading decisions, choosing the best servers with feasible link lifetimes, shorter distances, and good CPU availability and capacity. With this, servers can execute tasks faster and the results can be returned to the client before they lose connectivity, avoiding failures.

Second, if available, information about known routes of vehicles contributes to reducing the offloading processes' failures and, consequently, reducing task recoveries. With fewer recoveries, the reduction in the total execution time of the workload is greater. The other algorithms, with more recoveries, end up delaying more their total execution time. In the case of the MDO, although it has low percentages of TR, it does not take advantage of the computational resources of edge servers and has more tasks executed locally (TL), overloading client vehicles.

Third, the simultaneous use of the WAVE and 5G networks also contributes to GTT's good performance. This use allows the GTT to take advantage of both the vehicular clouds (with V2V) and the edge servers (with V2I). It also allows to increase the transmission capacity of tasks and reduce latencies. Besides, it combines the independence of infrastructure and direct communications from WAVE networks and the large data rates and increased band spectrum availability of 5G networks.

In this way, GTT makes more accurate contextual evaluations, better server choices, better task assignment, and takes advantage of technological advantages with the simultaneous use of WAVE, 5G, and the vehicular edge computing

system. Through extensive simulations and diverse vehicular environments, we show that GTT achieves the greatest reductions in the total execution times of tasks and the lowest rates of offloading failures in most cases, providing good and feasible solutions to problem P1.

7. Conclusion

This work presented a context-oriented framework for computation offloading in vehicular edge computing systems with the objective of reliably improving the performance of the application. The framework modules support discovering computational resources, gathering real-time contextual data, distributing computation tasks, and providing failure recovery. Then, the core of the framework, a greedy algorithm called GTT, can focus on task assignment, deciding where each application task should be executed. For this decision, GTT taking contextual information into account and taking advantage of WAVE and 5G networks and the computing resources of vehicular clouds and edge servers. Through extensive experiments in different vehicular environments and workloads, we evaluated the proposed framework's efficiency and the GTT algorithm compared to other solutions in the literature. As seen in the results, on average, GTT achieved to offload up to 89.4% of all tasks and needed to recover only 0.8% of them. Besides, on average, GTT has reduced up to 70.3% of tasks execution time compared to entirely local execution and up to 42.9% if compared to other algorithms. Thus, our solution showed good adaptation to the challenges encountered, such as fast vehicular mobility, contextual information gathering, and NP-hardness of the task distribution decision. Our solution also performed better than other solutions in terms of reducing total execution time and failure rates. Some characteristics of the GTT contributed to obtaining the best performance. For example, the use of diverse real-time contextual information parameters contributed to finding the best possible servers for each task of the application, executing the tasks more quickly, and minimizing failures. The special information about known routes of vehicles made it possible to pre-

dict vehicle positioning more accurately and avoid offloading failures. In fact, this information can reduce the number of failures/recoveries by up to 77.8%. Finally, the WAVE and 5G networks' simultaneous use increased the task transmission capacities, decreasing communication delays and remote task execution time. In future works, we intend to improve our solution by considering the task order and new information and techniques in the computation offloading processes.

References

[1] K. Wevers, M. Lu, V2x communication for its-from ieee 802.11 p towards 5g, IEEE 5G Tech Focus 1 (2) (2017) 5–10.

[2] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti, H. Zedan, A comprehensive survey on vehicular ad hoc network, J. Netw. Comput. Appl. 37 (2014) 380–392.

[3] D. Jiang, L. Delgrossi, Ieee 802.11 p: Towards an international standard for wireless access in vehicular environments, in: Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE, IEEE, 2008, pp. 2036–2040.

[4] M. Mezzavilla, M. Zhang, M. Polese, R. Ford, S. Dutta, S. Rangan, M. Zorzi, End-to-end simulation of 5g mmwave networks, IEEE Commun. Surv. Tutor. 20 (3) (2018) 2237–2263.

[5] C. R. Storck, F. Duarte-Figueiredo, A 5g v2x ecosystem providing internet of vehicles, Sens. 19 (3) (2019) 550.

[6] J. Zhang, K. B. Letaief, Mobile edge intelligence and computing for the internet of vehicles, Proc. IEEE 108 (2) (2019) 246–261.

[7] A. Boukerche, V. Soto, Computation offloading and retrieval for vehicular edge computing: Algorithms, models, and classification, ACM Comput. Surv. (CSUR) 53 (4) (2020) 1–35.

[8] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, F. Yang, Dependency-aware task scheduling in vehicular edge computing, IEEE Internet Things J. 7 (6) (2020) 4961–4971. doi:10.1109/JIOT.2020.2972041.

[9] A. B. Souza, P. A. Rego, T. Carneiro, J. C. Rodrigues, P. P. Rebouças Filho, J. N. Souza, V. Chamola, V. H. C. Albuquerque, B. Sikdar, Computation offloading for vehicular environments: A survey, IEEE Access 8 (2020) 198214–198243. doi:10.1109/ACCESS.2020.3033828.

[10] P. A. Rego, P. B. Costa, E. F. Coutinho, L. S. Rocha, F. A. Trinta, J. N. de Souza, Performing computation offloading on multiple platforms, Comput. Commun. 105 (2017) 1–13.

[11] D. Xu, Y. Li, X. Chen, J. Li, P. Hui, S. Chen, J. Crowcroft, A survey of opportunistic offloading, IEEE Commun. Surv. Tutor. 20 (3) (2018) 2198–2236.

[12] H. Vahdat-Nejad, A. Ramazani, T. Mohammadi, W. Mansoor, A survey on context-aware vehicular network applications, Veh. Commun. 3 (2016) 43–57.

[13] Y. Shin, H. Choi, Y. Nam, E. Lee, Data delivery protocol using the trajectory information on a road map in vanets, Ad Hoc Netw. 107 (2020) 102260.

[14] S. Wang, C. Ding, N. Zhang, X. Liu, A. Zhou, J. Cao, X. S. Shen, A cloud-guided feature extraction approach for image retrieval in mobile edge computing, IEEE Trans. on Mob. Comput.

[15] M. Tang, L. Gao, J. Huang, Enabling edge cooperation in tactile internet via 3c resource sharing, IEEE J. on Sel. Areas in Commun. 36 (11) (2018) 2444–2454.

[16] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, X. Wang, Learning-aided computation offloading for trusted collaborative mobile edge computing, IEEE Trans. on Mob. Comput. 19 (12) (2019) 2833–2849.

[17] L. Liu, C. Chen, Q. Pei, S. Maharjan, Y. Zhang, Vehicular edge computing and networking: A survey, Mob. Netw. and Appl. (2020) 1–24.

[18] G. Fan, H. Jin, Q. Liu, W. Qin, X. Gan, H. Long, L. Fu, X. Wang, Joint scheduling and incentive mechanism for spatio-temporal vehicular crowd sensing, IEEE Trans. on Mob. Comput.

[19] A. U. Rahman, A. W. Malik, V. Sati, A. Chopra, S. D. Ravana, Context-aware opportunistic computing in vehicle-to-vehicle networks, Veh. Commun. 24 (2020) 100236.

[20] M. Liwang, J. Wang, Z. Gao, X. Du, M. Guizani, Game theory based opportunistic computation offloading in cloud-enabled iov, IEEE Access 7 (2019) 32551–32561.

[21] M. Charitos, G. Kalivas, Mimo hetnet ieee 802.11 p–lte deployment in a vehicular urban environment, Veh. Commun. 9 (2017) 222–232.

[22] N. Dreyer, A. Moller, Z. H. Mir, F. Filali, T. Kurner, A data traffic steering algorithm for ieee 802.11 p/lte hybrid vehicular networks, in: 2016 IEEE 84th Vehicular Technology Conference (VTC-Fall), IEEE, 2016, pp. 1–6.

[23] S. Ucar, S. C. Ergen, O. Ozkasap, Multihop-cluster-based ieee 802.11 p and lte hybrid architecture for vanet safety message dissemination, IEEE Trans. Veh. Technol. 65 (4) (2015) 2621–2636.

[24] Z. Ning, X. Wang, J. J. Rodrigues, F. Xia, Joint computation offloading, power allocation, and channel assignment for 5g-enabled traffic management systems, IEEE Trans. Ind. Inform. 15 (5) (2019) 3058–3067.

[25] G. Qiao, S. Leng, K. Zhang, Y. He, Collaborative task offloading in vehicular edge multi-access networks, IEEE Commun. Mag. 56 (8) (2018) 48–54.

[26] H. Wang, X. Li, H. Ji, H. Zhang, Federated offloading scheme to minimize latency in mec-enabled vehicular networks, in: 2018 IEEE Globecom Workshops (GC Wkshps), IEEE, 2018, pp. 1–6.

[27] X. Fan, T. Cui, C. Cao, Q. Chen, K. S. Kwak, Minimum-cost offloading for collaborative task execution of mec-assisted platooning, Sens. 19 (4) (2019) 847.

[28] S. Raza, W. Liu, M. Ahmed, M. R. Anwar, M. A. Mirza, Q. Sun, S. Wang, An efficient task offloading scheme in vehicular edge computing, J. Cloud Comput. 9 (2020) 1–14.

[29] G. Barb, M. Otesteanu, 4g/5g: A comparative study and overview on what to expect from 5g, in: 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), IEEE, 2020, pp. 37–40.

[30] T. S. Rappaport, Y. Xing, G. R. MacCartney, A. F. Molisch, E. Mellios, J. Zhang, Overview of millimeter wave communications for fifth-generation (5g) wireless networks—with a focus on propagation models, IEEE Trans. Antennas Propag. 65 (12) (2017) 6213–6230. doi:10.1109/TAP.2017.2734243.

[31] M. Tehrani-Moayyed, F. Restuccia, S. Basagni, Comparative performance evaluation of mmwave 5g nr and lte in a campus scenario, Proceedings of IEEE VTC 2020 Fall.

[32] J. Feng, Z. Liu, C. Wu, Y. Ji, Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling, IEEE Veh. Technol. Mag. 14 (1) (2018) 28–36.

[33] A. B. Souza, P. A. L. Rego, P. H. G. Rocha, T. C. Pessoa, J. N. d. Souza, A task offloading scheme for wave vehicular clouds and 5g mobile edge computing, in: 2020 IEEE Global Communications Conference (Globecom), IEEE, 2020, pp. 1–6.

[34] A. B. Souza, J. Celestino, F. A. Xavier, F. D. Oliveira, A. Patel, M. Latifi, Stable multicast trees based on ant colony optimization for vehicular ad hoc networks, in: The International Conference on Information Networking 2013 (ICOIN), IEEE, 2013, pp. 101–106.

[35] J. Härri, C. Bonnet, F. Filali, Kinetic mobility management applied to vehicular ad hoc network protocols, Comput. Commun. 31 (12) (2008) 2907–2924.

[36] H. Menouar, M. Lenardi, F. Filali, Movement prediction-based routing (mopr) concept for position-based routing in vehicular networks, in: Vehicular Technology Conference, 2007. VTC-2007 Fall. 2007 IEEE 66th, IEEE, 2007, pp. 2101–2105.

[37] V. Namboodiri, L. Gao, Prediction-based routing for vehicular ad hoc networks, IEEE Trans. Veh. Technol. 56 (4) (2007) 2332–2345.

[38] J. Zhang, H. Guo, J. Liu, Y. Zhang, Task offloading in vehicular edge computing networks: A load-balancing solution, IEEE Trans. Veh. Technol. 69 (2) (2019) 2092–2104.

[39] Y. Sun, J. Song, S. Zhou, X. Guo, Z. Niu, Task replication for vehicular edge computing: A combinatorial multi-armed bandit based approach, in: 2018 IEEE Global Communications Conference (GLOBECOM), IEEE, 2018, pp. 1–7.

[40] C. Chen, L. Chen, L. Liu, S. He, X. Yuan, D. Lan, Z. Chen, Delay-optimized v2v-based computation offloading in urban vehicular edge computing and networks, IEEE Access 8 (2020) 18863–18873.

[41] S. Shaham, M. Ding, M. Kokshoorn, Z. Lin, S. Dang, R. Abbas, Fast channel estimation and beam tracking for millimeter wave vehicular communications, IEEE Access 7 (2019) 141104–141118.

[42] M. Giordani, A. Zanella, T. Higuchi, O. Altintas, M. Zorzi, Performance study of lte and mmwave in vehicle-to-network communications, in: 2018 17th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net), IEEE, 2018, pp. 1–7.

[43] T. Cui, Y. Hu, B. Shen, Q. Chen, Task offloading based on lyapunov optimization for mec-assisted vehicular platooning networks, Sens. 19 (22) (2019) 4974.

[44] S. Midya, A. Roy, K. Majumder, S. Phadikar, Multi-objective optimization technique for resource allocation and task scheduling in vehicular cloud architecture: A hybrid adaptive nature inspired approach, J. Netw. Comput. Appl. 103 (2018) 58–84.

[45] M. Chen, Y. Hao, M. Qiu, J. Song, D. Wu, I. Humar, Mobility-aware caching and computation offloading in 5g ultra-dense cellular networks, Sens. 16 (7) (2016) 974.

[46] G. Zobolas, C. D. Tarantilis, G. Ioannou, Exact, heuristic and metaheuristic algorithms for solving shop scheduling problems, in: Metaheuristics for scheduling in industrial and manufacturing applications, Springer, 2008, pp. 1–40.

[47] I. F. Akyildiz, C. Han, S. Nie, Combating the distance problem in the millimeter wave and terahertz frequency bands, IEEE Commun. Mag. 56 (6) (2018) 102–108.

[48] F. A. Teixeira, V. F. e Silva, J. L. Leoni, D. F. Macedo, J. M. Nogueira, Vehicular networks using the ieee 802.11 p standard: An experimental analysis, Veh. Commun. 1 (2) (2014) 91–96.

[49] G. F. Riley, T. R. Henderson, The ns-3 network simulator, in: Modeling and tools for network simulation, Springer, 2010, pp. 15–34.

[50] D. Krajzewicz, Traffic simulation with sumo–simulation of urban mobility, in: Fundamentals of traffic simulation, Springer, 2010, pp. 269–293.

[51] J. Song, Y. Wu, Z. Xu, X. Lin, Research on car-following model based on sumo, in: The 7th IEEE/International Conference on Advanced Infocomm Technology, IEEE, 2014, pp. 47–55.

[52] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications, arXiv preprint arXiv:1704.04861 (2017) eprint.

[53] J. Redmon, A. Farhadi, Yolov3: An incremental improvement, arXiv preprint arXiv:1804.02767 (2018) eprint.