Context-aware task offloading with QoS-provisioning for MEC multi-RAT vehicular networks

Lucas Bréhon–Grataloup, Rahim Kacimi, André-Luc Beylot
IRIT, Université de Toulouse, CNRS, Toulouse INP, UT3, Toulouse, France.
{lucas.brehon-grataloup, rahim.kacimi, andre-luc.beylot}@irit.fr

2022 International Conference on Computer Communications and Networks (ICCCN) | 978-1-6654-9726-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICCCN54977.2022.9868873

***Abstract*—The next step towards vehicular networks in smart cities would be the deployment of autonomous shuttles with multiple on-board applications. Their need to offload task towards Road Side Units (RSUs) is inevitable, especially with a certain proportion of urgent data needing to be processed in the shortest possible delay. Therefore, QoS-provisioning appears as imperative, along with the optimization of resource requesting at RSUs. To this end, we propose CAVTOMEC, a multi-RAT location-aware, context-aware task offloading solution with QoS provisioning for MEC vehicular networks. Our solution consists of three intertwined mechanisms: traffic classification, location-awareness exploiting the contents of CAM beacons and V2N-enhanced resource polling. Traffic classification identifies high, low, and indifferent task priorities, while location and resource awareness help to select the most appropriate RSU to offload tasks to depending on these priorities. Performance evaluations show that our proposal offers better load balancing at the RSUs than traditional offloading schemes, thus satisfying high priority task offloading at better rates and freeing up more resources in case an unexpected event occurs.**

***Index Terms*—MEC, Task offloading, C-V2X, Connected autonomous vehicles, V2I, QoS-provisioning, OMNET++ and Veins simulation.**

I. Introduction

With the widespread deployment of 5G and the implementation of smart cities, connected autonomous vehicles (CAVs) become increasingly attractive, yet their acceptance by the general public seems to progress at slow speeds, generally calling for reassuring, accessible and reliable use cases [1]. In that regard, the objective of shuttles is to pick up and drop off passengers at pre-defined stops on a given circuit. Bringing connected services to this use case could allow a variety of new services to be integrated within autonomous shuttles, such as better road safety, onboard entertainment, optimized road traffic and so on.

To this end, multiple tasks involving massive amounts of data need to be computed within strict time constraints. Consequently, the advent of multi-access edge computing (MEC) allows a vehicular host to offload tasks to edge servers located at Road-Side Units (RSUs) close to the road, using Vehicle-to-Infrastructure (V2I) communications [2]. However, with connected vehicles being highly mobile in nature and often harboring urgent applications, the issues of connectivity stability, communication efficiency, time, and resource management need to be addressed.

In this paper, we present CAVTOMEC, a context-aware task offloading mechanism with QoS-provisioning for multi-RAT (multiple Radio Access Technologies) vehicles, aimed at balancing resource load at RSUs while ensuring satisfactory execution delay for urgent tasks. Such behavior is encouraged so as to guarantee a minimal amount of available CPU resources at all times at each RSU, in case a very urgent task would need to be treated as soon as possible. CAVTOMEC's features tackle multiple issues at once:

1) Categorizing traffic into three priority classes: high, medium and low, serves as an entry point to determine how much leeway there is in terms of execution time.
2) A more thorough exploitation of long-range, Vehicle-to-Network (V2N) communications allow for the centralization of resource data. This way, vehicles receive regular updates on RSU usage state, which helps them to select the best offloading destination.
3) The retrieval of Cooperative Awareness Message (CAM) beacons allows vehicles to position themselves with each other and relative to the RSUs, which facilitates the selection of a destination. With D2D communications, urgent tasks greatly benefit from short, obstacle-free distances in terms of delay and reliability.

This work falls within the autOCampus project, aiming at transforming a university campus into an intelligent, innovative and ecological site through the means of autonomous vehicles, automatic barriers and intelligent lighting [3]. The reminder of the paper is organized as follows: section II presents the related works. Section III characterizes the problem of task offloading in multi-RAT vehicular networks. Section IV introduces the system design of our solution. In section V, we provide performance evaluation, results and discussion, while Section VI summarizes the major takeaways. Finally, section VII concludes this paper.

II. Related works

This section introduces recent related works and discusses how our contribution represents novelty from them. Being a prerequisite of delay reduction, the concepts of task and data offloading in vehicular networks have been greatly studied in recent years. Since a major portion of the research has been conducted on for V2I and/or V2V (Vehicle-to-Vehicle) task

978-1-6654-9726-8/22/$31.00 ©2022 IEEE

offloading, our focus is on a V2N-enhanced task offloading mechanism.

Deng *et al.* [4] introduced vehicular multi-hop to propagate data ahead of the emitting host, allowing tasks to be computed at an RSU further up the road before the offloading vehicle reaches it to gather the results. This solution solves the issue of high mobility in the context of V2I task offloading and result retrieval, however no route nor destination is clearly defined when the vehicle asks for a task to be offloaded, our solution strives for precise destination selection.

Tang *et al.* [5] tackle a similar problem to ours, but in the context of a straight lane with a single-RAT 5G MEC architecture. The RSUs are replaced with gNBs, connected to one and only one MEC server which confines the study to a single-hop task offloading scheme. Our study is closer to a real-life situation, with our simulation taking place on a university campus and considering not all urban blocks are currently being serviced with 5G coverage. We share similar differences with Saleem *et al.*, whose scheme is exclusively focused on V2I data offloading with 802.11p technology, in the context of a 30-kilometer straight highway [6]. Our urban, slower-moving context raises different questions on reliability and connectivity.

On the topic of sharing data between MEC hosts, Guo *et al.*'s architecture makes edge servers use a multi-layered deep learning network to share intelligence located at RSUs and aggregate data from onboard vehicular sensors as well as from the network itself [7]. The diversity of this multi-level data, gathered from the applications to the hosts, is fed through a decision algorithm to select which task to execute, and at which host. The wealth of equipments populating the network are the backbone of this collaboration-based framework. However, few pieces of research integrate the possibility of having very few equipments available within the architecture.

To the best of our knowledge, there does not exist any heterogeneous vehicular task offloading scheme that considers either shuttle-oriented architectures or V2N as a supporting role for context awareness. We exploit both long and short range communications in order to optimize data transmission for most of the V2X (Vehicle-to-Everything) applications that could be seen in smart cities. We also model a realistic playground with an actual city block instead of a single straight lane, and a small amount of connected vehicles as would be expected for the first steps of real-world deployment.

III. Task offloading in multi-RAT vehicular networks

This section introduces the concept of task offloading in vehicular networks with hosts being equipped with multiple network interfaces. With the appeal of smart cities, multiple use cases have been imagined and reported by groups such as 5GAA to deploy novel applications at vehicular hosts in order to enhance performance for all users, including those around and not inside vehicles. Among them are delay-sensitive use cases like road safety and remote driving, intelligence-driven use cases like traffic jam avoidance, occasional and latency-agnostic applications like usage reports and software updates and so on [8]. This variety calls for a flexible network architecture, able to adapt services depending on which application relies on it at a given time. However, although each connected vehicle hosts on-board computing capacities, they are limited by nature, and generally not conceived to handle all applications at once. Consequently, the largest amounts of data and the most complex tasks and operations need to be offloaded away from the vehicle to larger computing units called mobile or multi-access edge computing (MEC) servers and located at RSUs. After the tasks are executed over there, their results are sent back to the offloading vehicle. We can define for a given task u its total offloading delay t_u as:

$$t_u = t_u^{ul} + t_u^{comp} + t_u^{dl} \tag{1}$$

where t_u^{ul} is the communication delay for the transmission of the task from the vehicle to the MEC server, t_u^{comp} is the computation delay at the MEC server, and t_u^{dl} is the communication delay for the transmission of results from the MEC server back to the vehicle.

The RSUs' MEC servers being equipped with potent CPUs and even GPUs at times, as well as their being positioned in physical proximity to the road, greatly reduce both computing and communication delays, thus satisfying most applications' QoS requirements. We can formulate the computation delay of a given task u at the MEC server t_u^{comp} as:

$$t_u^{comp} = \Delta t_u^{queue} + \alpha_u / F_{CPU} \tag{2}$$

where Δt_u^{queue} is the duration the task spends on the server waiting to be computed, α_u is the task's computation needs estimated in number of operations, and F_{CPU} is the clocking frequency of the MEC server's CPU.

To fully exploit the potential of task offloading, the first challenge to overcome is the selection of the network to send data through. In vehicular networks, most hosts are equipped with multiple network interfaces, as most current hardware is able to at least handle C-V2X and the 802.11p-based DSRC/ITS-G5 communications [9]. The subsequent heterogeneous vehicular networks allow for the management of multiple cells with various coverage areas, thus adaptive to a wide variety of applications. Indeed, V2X applications generally rely on two major kinds of communications : V2V and V2I, both benefiting from 802.11p and C-V2X thanks to their dedication to short-range communications. However, additional communication types are being explored, such as vehicle-to-pedestrian (V2P) and V2N.

802.11p was introduced by the IETF as a mobility-friendly amendment to the two-decade-old Wi-Fi protocol, where vehicles can skip the traditional association and authentication phases to save precious communication time. It was quickly adopted as a standard for vehicular communications, however the constraints of its foundation, the 802.11a protocol, do not save it from fatal limitations, especially with the emergence of strict and resource-hungry applications in the era of 5G. Wi-Fi-based vehicular network protocols seem doomed to be overcome by cellular technologies.

The main concerns with vehicular networks comprising both coverage and capacity, it seems logical for cellular networks, starting with LTE ([10]), to be perceived as ideal candidates to tackle these issues. With Cellular-V2X (C-V2X), vehicles benefit from two interfaces, thus becoming able to transmit data directly to nearby peers via device-to-device communications (D2D), while keeping the ability to communicate through long ranges like any other cellular device. In this case, the transmissions go through a central access point: eNodeB or gNodeB, for LTE and 5G respectively. These Vehicle-to-Network communications tend to be overlooked in most vehicular applications due to their higher delay compared to D2D, however we could imagine a solution where the centralization of information at the Access Point (AP) contributes to resource management.

Indeed, long range V2X transmissions were not particularly explored as a solution, limited in the standard to sporadic road traffic updates in the case of an unexpected event occurring ahead ([11]). The presence of this dedicated Uu interface represents, in our opinion, untapped potential for optimized task offloading, given its possibility to share useful data to distances and numbers of hosts unimaginable for short-range interfaces.

IV. System design of our solution

This section introduces the system design of our task offloading solution, in response to the previously-mentioned needs related to smart cities and the potential of multi-RAT. In the latest models of vehicular networks, hosts rely on task and/or data offloading to enhance performance while reducing delay. However, if a situation calls for high-priority tasks to be sent and computed in close-to-real-time, issues can occur where the closest MEC server is overloaded, and thus not able to compute any more tasks without freeing up capacity first. In these cases, the new, urgent tasks either have to wait or be offloaded to a farther RSU, giving prolonged delays in both cases. Our task offloading mechanism strives for a solution to this situation, by focusing on all parameters involved: task priority, RSUs' currently available computing capacity, and distance between offloading host and candidates. This is CAVTOMEC, a Context-Aware V2X Task Offloading Multi-RAT mechanism for MEC networks.

A. Problem definition

In our architecture, we model a small vehicular network comprised of a few multi-RAT hosts with C-LTE and 802.11p interfaces. While both the 802.11p interface and the C-V2X PC5 interface may seem redundant due to their short-range design, they serve different purposes in our study. The first one serves as a propagation tool for CAM beacons and other position messages, while the second one is used to send and receive offloading-related packets. This way, no time will be lost with vertical handover and no data links will be disrupted.

The usual process of V2I task offloading is divided in three parts: the vehicle prepares and offloads the task and its related data to its computing candidate, then the task is processed at the MEC server, and finally the RSU sends the task results back to the offloading vehicle. However a major flaw with this process is the lack of intelligence with the selection of an offloading destination. Worried about delay, vehicles usually offload their tasks to the nearest RSU, regardless of the task's urgency or the destination's current load. The subsequent risks of offloading a task to a fully loaded RSU or MEC server include task execution failure, retransmissions and generally speaking, increased delay. In the context of vehicular applications, where each millisecond counts, such a risk cannot be entertained. This situation calls for an offloading mechanism aware of extensive context information. We thereby design a decision-making assistance mechanism with QoS-provisioning, so as to balance and increase reliability for V2I task offloading.

The utility of such a mechanism in the context of a shuttle roaming a smart city is basically enhanced resource availability for a vehicle that is most likely going to need to offload more tasks as well as larger amounts of data than the usual car. The contribution of V2N paves the way to the efficient exploitation of both C-V2X interfaces, resiliently even in the case of higher loads, by making the vehicle able to send data farther than its D2D range alone.

B. RSU usage updates

In V2I task offloading architectures, RSUs exploit their short-range interface (in our case the C-V2X PC5 one) to receive task offloading requests and to send task results back to vehicles. In these architectures, a go-to offloading scheme often consists in sending vehicular tasks to the RSU closest to the offloading host. We call this scheme ”location-based”. When receiving offloaded tasks, the RSUs allocate the necessary computing resources to execute these tasks in the shortest amount of time. Due to mostly economic reasons, the edge servers' integrated CPU has limited amounts of resources, which puts them at risk of overloading.

In our solution, when the proportion of allocated computing resources changes (due to either reception or completion of tasks), the RSU updates the system with the information on its current available resources. These updates are propagated through the long-range Uu interface so as to reach the cellular AP, making this update-based system available at all times. We position an extra edge server at this AP, dedicated to the management of RSU usage data. Upon reception of these messages, a small recap message is constructed, containing the latest data on every RSU's available CPU resources. These recaps are then propagated regularly to all vehicular hosts using the long-range Uu interface again, giving them mutual intelligence on offloading capacity with very low network overhead. A visual representation of this architecture can be seen in Fig 1.

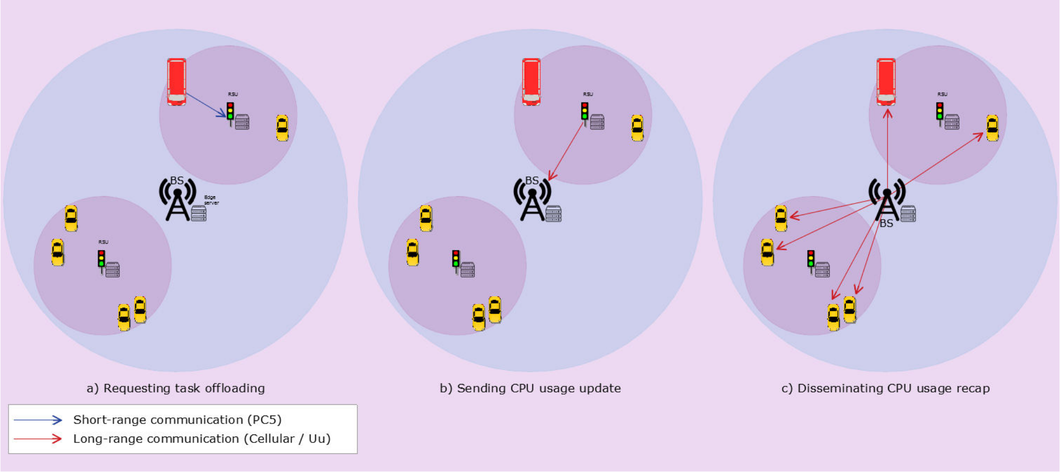
RSU
BS
Edge server
RSU
a) Requesting task offloading
BS
b) Sending CPU usage update
BS
c) Disseminating CPU usage recap
Short-range communication (PC5)
Long-range communication (Cellular / Uu)

Fig. 1. RSU sending usage updates.

C. Offloading destination selection

The backbone of the solution we propose is definitely the selection mechanism for offloading destinations. We make the choice of making the position of all RSUs known to all vehicles at all times, within a list $List_{pos}$, allowing them to integrate the distance to infrastructure elements within their selection algorithm. The behavior also depends on QoS preference at a given time, thus relying on priority of task. When a task u needs to be offloaded, the emitting vehicle notes the necessary CPU resources Cr_u, checks for priority flags P_u, and compares its current position to the ones of the playground's RSUs. Having received usage recaps from the eNB-located MEC server, they can exploit this data to select the best MEC candidate for V2I task offloading.

The detail of such behavior is written in Algorithm 1, with two distinct procedures to be executed concurrently. First, the vehicle needs to retrieve RSU usage data from recap messages received on the Uu interface, and store this data as an update to their usage table $List_{usage}$. In this table, each RSU i.d. is paired to a percentage corresponding to its proportion of available CPU resources. With each update, the table is sorted in two ways and from best to worst: closeness to the RSU (named $List_{usage}^{prox}$) and percentage of free resources (named $List_{usage}^{desc}$). Second, following these operations, whenever a task needs to be offloaded, these tables will be the ones to be consulted. Depending on the QoS requirements, deducted from the priority P_u of task u, the selection criteria vary. High priority focuses on offloading as quick as possible to a server that is capable of executing it, which amounts to browsing the $List_{usage}^{prox}$ table and seeking the corresponding usage entry. Starting from the RSU closest to the vehicle, the first one to have enough available resources is immediately sent the task. Medium priority is insensitive to RSU proximity, thus only selecting the RSU with maximum CPU availability. Finally, low priority triggers an effort to deliberately find the RSU with the lowest amount of available CPU. In all of these cases, if, according to the vehicle's usage table, no RSU currently has enough resources to compute the task, it is queued waiting for the next update. Consequently, the MEC servers are assumed to compute tasks as they come, without queuing them in case of overload, giving $\Delta t_u^{queue} = 0$. This way, we benefit from greater visibility over the offloading process. Reusing the previous t_u^{comp} formula, we obtain:

$$t_u^{comp} = \alpha_u / F_{CPU} \tag{3}$$

V. Performance analysis

This section presents the performance evaluation of our mechanism (denominated ”CAVTOMEC” or ”CAV” from now in our figures) compared to the previously introduced location-based scheme (denominated ”Loc” from now our in our figures), usually selected for its ease of implementation and low computing overhead. Our simulation is done in the OMNET++ simulator with its integrated iNET framework version 4.2.5, coupled with the Veins framework version 5.1 [12] and SimuLTE version 1.2 [13]. Finally we use SUMO 1.9.2 [14] for generating mobility data and visuals for the vehicles within our urban scenario. OMNET++ was chosen for its modularity and for its integrated internet protocol model library iNET, making it an accessible and flexible network simulator with a strong basis to research over. Adding the Veins framework allows us to simulate vehicular communications, especially 802.11p-based, along with road traffic using cosimulation with the SUMO simulator. For the purposes of multi-RAT, SimuLTE was integrated as an OMNET++/iNET LTE network simulator, so as to provide hosts with the ability to switch between 802.11p and LTE-V2X interfaces. The software versions selected for the purposes of this work were necessary to the better reliability of the interactions between each part of the simulation.

A. Implementation preliminaries

Preliminary to the experiments, the first need to be addressed concerns the simulation scenario itself. We hereby

Algorithm 1 Offloading destination selection

```
Require: $List_{pos}, List_{usage}$
procedure UPDATE USAGE INTELLIGENCE
    while Every T time do
        if Usage Recap received then
            Update List $List_{usage}$
            New $List^{prox}_{usage} \leftarrow List_{usage}$ sorted by distance to vehicle
            New $List^{desc}_{usage} \leftarrow List_{usage}$ sorted by available CPU
            if Vehicle has a task $u$ to send then
                Select_Destination($P_u$,$Cr_u$)
            end if
        end if
    end while
end procedure
procedure SELECT DESTINATION($P_u, Cr_u$)
    if $P_u$ is HIGH then
        for $k = 0, \ldots, len(List^{prox}_{usage})$ do
            if $List^{prox}_{usage}(k) > Cr_u$ then
                Send $u$ to corresponding RSU
            else
                if $k == len(List^{prox}_{usage})$ then
                    Wait for next update
                end if
            end if
        end for
    else if $P_u$ is MEDIUM then
        if $List^{desc}_{usage}(0) < Cr_u$ then
            Wait for next update
        else
            Send $u$ to corresponding RSU
        end if
    else if $P_u$ is LOW then
        for $k = len(List^{desc}_{usage}), \ldots, 0$ do
            if $List^{desc}_{usage}(k) > Cr_u$ then
                Send $u$ to corresponding RSU
            else
                if $k == 0$ then
                    Wait for next update
                end if
            end if
        end for
    end if
end procedure
```

summarize the choices we made regarding the simulator, to then determine the available hardware at each host, the kinds of applications involved and the amount of expected traffic through the duration of the simulation.

Previous versions of iNET allowed hosts to exploit multiple network interfaces along with their dedicated network and transport layers. However, the most recent versions like the one we chose deleted this ability, making all packets leaving a given network interface go through the same upper layers. Considering that Veins-based frames and iNET-based frames depend on different code modules, this behavior made us implement harmonization methods in the SAP between layers, so as to present all frames the same way regardless of the RAT it comes from or goes to. We then tweaked the IP address allocation mechanism so as to facilitate network interface selection when applications send data to their sockets.

Usage packets contain very little information, comprised of the RSU i.d. and the percentage of available CPU resources so as to induce as little load on the cellular network as possible: each beacon is of the order of the hundred bytes.

In order to simulate the transmission of usage beacons after centralization at the eNodeB, we gave the eNodeB itself the ability to host an application capable of creating packets and sending them to vehicular hosts. In other words, the eNodeB itself becomes part of MEC server. We made this choice instead of creating an actual server because this would have implied traffic in the core network, through the PGW and so on, as is meant in the base product because a BS is not supposed to host an MEC server in classic architectures. However, in a V2X MEC-based architecture, such changes remain reasonable so as to limit core network traffic to a minimum, which we are aiming for. Indeed, our simulation design choice yields an amount of edge traffic similar to the one expected from our architecture, as we mention once more that core network traffic is intended to be greatly reduced, and cloud computing is zero. Consequently, on the vehicle side, we developed three apps. The first one is the usage beacon reception app as introduced earlier. It shares its results with the second one: the task generation app, also integrating Algorithm 1 for the selection of an offloading destination. Lastly, the result reception app gathers the completed offloaded tasks and generates statistics. For RSUs, we developed the full usage beaconing mechanism and task reception.

B. Context of experimentation

The vehicles generate new tasks at a frequency following a Poisson distribution with parameter λ. Each task u is offloaded along with some amount of data of size γ, selected between constraints γ_{min} and γ_{max}. Task complexity α is proportional to the task data size, with constraints α_{min} and α_{max}, and expressed in percentages of CPU resources. Given the population of the depicted vehicular network, we estimate that a λ value of 1 Hz is most likely going to induce low enough channel load to transmit data both ways and complete a vehicle's task before it tries to offload another one. On the other hand, increasing this value ten-fold is likely to induce

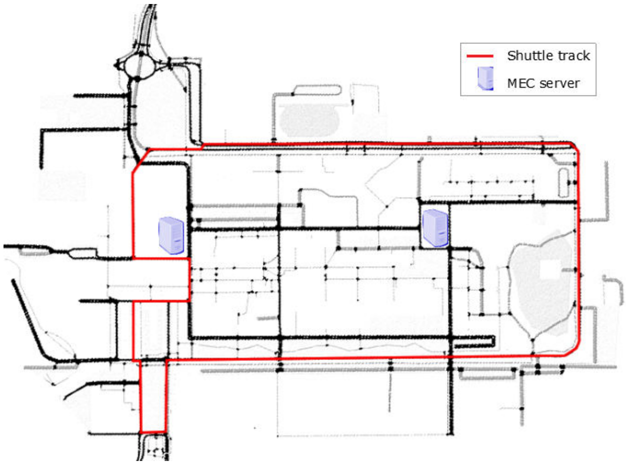

Fig. 2. University environment map.

network and CPU load, bearing an impact on success rates and transmission delays, thus on completion delays as well.

The simulation aims at comparing performances of the two aforementioned solutions with evolving values of λ, γ_{min}, γ_{max}, α_{min} and α_{max}. In other words, our objective with these scenarios is to gather information on, firstly, the overall usefulness of CAVTOMEC compared to a baseline offloading mechanism. Secondly, by exploiting different offloading-related variables like task emission frequency and task complexity, our solution's performance is tested in multiple contexts. The more frequent and complex the tasks are, the more challenging the context becomes. Groups of metrics emerge from these experiments, distinguished by the type of offloading mechanism, the λ value, and the presence of an asterisk (*) signaling that the studied scenario involves tasks with higher CPU needs, thus more complex.

A map of the university we used as playground, considering it to be the best place for a smart shuttle to be deployed along with RSUs, can be seen on Figure 2. Table I summarizes our simulation parameters, with a small amount of hosts relative to the available surface, as was justified previously. At the moment, the connected shuttle behaves like the other vehicles network-wise, albeit with a capped speed and a predefined circuit to drive around. A dedicated behavior for shuttles will be tested in the future so as to ingrain the capabilities of our solution in an urban scenario.

Each solution s was tested in multiple scenarios, basing our evaluation on the several criteria mentioned above. The resulting data was combined through 30 runs with varying seeds for the upper-layer parts, providing us with metrics such as:

- Available CPU $Ca_k^{\lambda,s}$ at each MEC server k: checking if the load is well balanced between RSUs.
- Available CPU standard deviation $\sigma\_k^{\lambda,s}$ at each MEC server k: checking that the computing resources are used consistently between RSUs, thereby leaving room for unexpected urgent tasks.

TABLE I
SIMULATION PARAMETERS

Name	**Value**
Simulation runs	30
Simulation time (s)	1500
Number of shuttles - cars	1 - 5
Number of RSUs - eNodeBs	2 - 1
RSU CPU frequency (MHz)	800
Car - Shuttle max speed (m.s^{-1})	13.9 - 2.8
CAM beacon interval (ms)	100
λ values (Hz)	$\{1 ; 5 ; 10\}$
λ^* values (Hz)	$\{1 ; 5 ; 10\}$
γ_{min} - γ_{max} (KB)	[20 ; 200]
α_{min} - α_{max} (%)	[1 ; 35]
γ^*_{min} - γ^*_{max} (KB)	[100 ; 400]
α^*_{min} - α^*_{max} (%)	[5 ; 50]

- Task computation success rate $R^{\lambda,s}$: measuring the impact (if any) on transmission reliability.
- Task computation delay $t_T^{\lambda,s}$: measuring the total time, including transmissions, to offload tasks and send back the results.

C. CPU load balancing and stability

In Fig 3 is presented a comparison of CPU load mean value 3(a) and standard deviation 3(b) between RSUs in each scenario, with varying λ and α values. We can note firsthand in Fig 3(a) that MEC servers solicit greater amounts of CPU resources with higher λ values, especially with less complex tasks. This behavior is explained by the fact that low-complexity tasks are more inclined to be accepted by the MEC server without reaching full CPU load. Also, the higher task emission frequency allows for some amount of parallel processing, while a λ value of 1 Hz represents a low amount of tasks to compute at the same time, thus needing fewer resources. The same reasoning can be applied backwards to the $\lambda^* = 10$ case: tasks with high CPU needs are less likely to be accepted by the MEC server even though some amount of computing resources is still available, so as to prevent overload. However, it can be seen that our mechanism reaches higher amounts of CPU usage compared to the baseline mechanism, especially with more complex tasks. Fig 3(b) depicts high σ values indifferent of scheme and RSU for low λ values with high complexities. This is due to the fact that MEC servers are not active at all times when small amounts of tasks are generated, but the high CPU requirements of these tasks call for massive amounts of allocated resources when accepted, thus leading to fluctuations in CPU usage. However, when $\lambda = 10$, a clear difference can be noted, with our solution showing 25% lower σ values than the location-based one. The CPU usage is thus more stable.

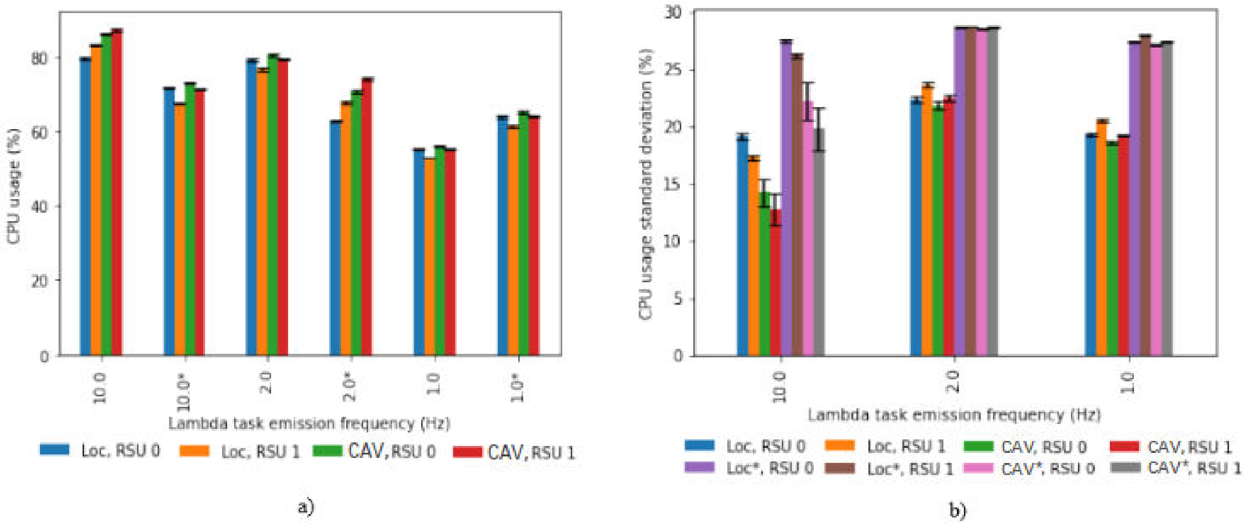

Fig. 3. CPU usage for each RSU and scenario.

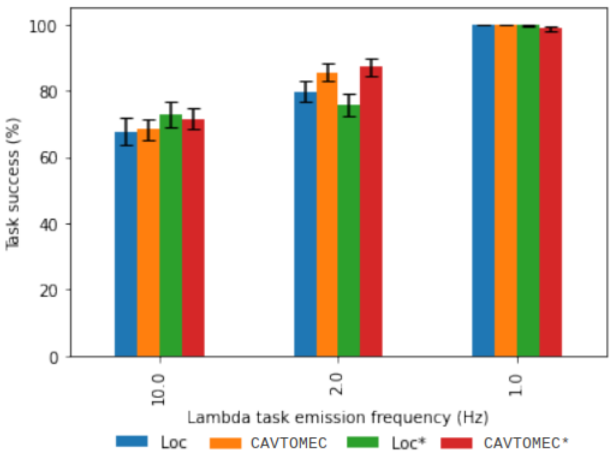

Fig. 4. Task success rate.

Overall, it is shown in Fig 3(a) that our mechanism allows edge servers to smoothen their load between each other, thus being much less exposed to overload or even moments of very low available computing power. Moreover, Figure 3(b) shows lower values of CPU usage standard deviation, meaning that RSUs are able to handle fluctuations in task computation needs more smoothly. This way, at any moment, urgent tasks can be offloaded immediately from almost anywhere and still expect to be completed in time.

D. Task success rate

Fig 4 portrays the average task success rate $R^{\lambda,s}$ in various offloading situations. Confidence intervals allow us to deduct that for an intermediate frequency of task emission ($\lambda = 2.0$, *i.e.* one new task is emitted per half a second on average), CAVTOMEC achieves better task success rates by 6 percentage points for non-complex tasks, and by 12 points for complex tasks. This highlights two contributions: firstly, that with our mechanism, tasks tend to not wait or be rejected by RSUs as often, thus allowing them to be completed and their results to be sent back in time. Secondly, that transmissions are more reliable overall, hence not requiring packets to be sent again, wasting time in the process.

E. Task execution delay

Fig 5 shows the evolution of task execution delay for both solutions s through the duration of the simulation $t_T^{\lambda,s}$, separated by task priority: H for high and L for low. Respectively, the evolutions of $t_T^{1.0,s}$ and $t_T^{1.0^*,s}$ are depicted in sub-figures (a) and (b). Fig 6 shows the evolutions of $t_T^{10,s}$ and $t_T^{10^*,s}$ in the same manner.

Firstly, a great difference in t_T values can be noted depending on the chosen value of λ. When a task is generated each second on average, the network load is minimal to communicate with the RSU, and the CPU load leaves a great margin as was deducted from Fig. 3. Consequently, offloaded tasks take very short times to reach the edge server and to be computed over there, leading to multiple sub-second task completion delays in the case of small γ values. With more complex tasks (Fig 5(b)), $t_T^{1.0^*,s}$ increases due to the bigger size of task data to transmit beforehand, as well as the slightly higher CPU load at RSUs as was seen on Fig 3(a). However the values are still satisfactory, of the order of the second.

Secondly, especially on Fig. 6, it is clearly visible that our solution produces a different treatment regarding delay depending on task priority, favoring quick transmissions of high priority tasks at the detriment of low-priority ones. On the other hand, the location-based solution treats all tasks equally, as can be seen with the similarity between the L and H plots. It can also be seen that our mechanism guarantees much lower computation delays for high priority tasks in the

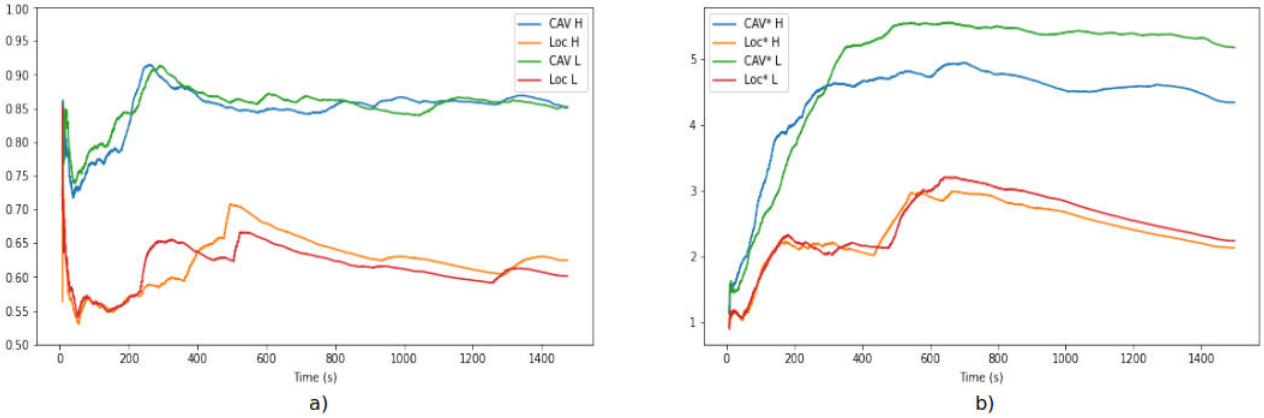

Fig. 5. $t_T^{1.0,s}$ (a) and $t_T^{1.0^*,s}$ (b) over time, with task priority.

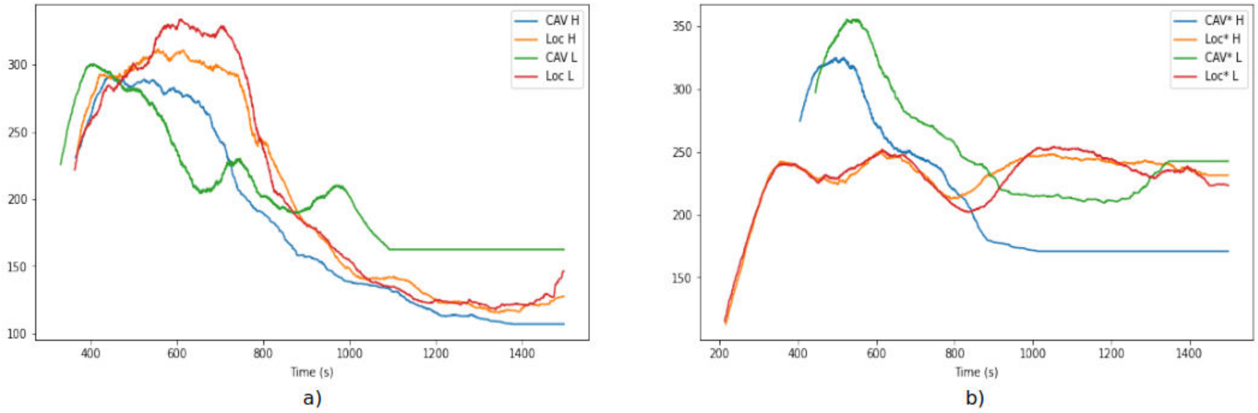

Fig. 6. $t_T^{10,s}$ (a) and $t_T^{10^*,s}$ (b) over time, with task priority.

high frequency scenario, and still holds its ground with lower frequencies, albeit with slightly higher but still acceptable delay. In the situation shown on Fig 5, the presence of the algorithm and the use of V2N induces a slight delay, visible when small amounts of tasks are generated, but greatly compensated with the better performance obtained when managing higher amounts of tasks. Overall, it can be noted that the transient state reaches its end quicker with our solution.

F. Confidence intervals

Considering the impact of randomness on our system regarding task generation, like creation time, data size selection and complexity draw, we evaluate the system with repeated simulation runs for each scenario. Using 30 runs per scenario, we can guarantee satisfactory confidence intervals at 95% for our Fig. 3 and Fig. 4 histograms, with a variability of less than 10 percent of the plotted values. In the previous subsections, we have also considered the presence of such intervals in our interpretation of the given measurements. Confidence intervals allow us to draw trustworthy conclusions from the data extracted with our experiments.

VI. Major takeaways

This section summarizes our conclusive remarks with regards to our simulation results. From a conceptual standpoint, our V2N-enhanced task offloading mechanism allows vehicles to access much longer ranges thanks to their LTE-V2X Uu interface. Thanks to sporadic usage updates sent from RSUs close and far, the best offloading destination can be selected via our QoS-aware algorithm depending on task priority, providing much better ranges than the straightforward location-based offloading mechanism. This has been demonstrated by the smoothing of CPU usage between RSUs with our solution shown in Fig 3(a), translating an efficient load-balancing behavior, as well as by the small extra delay seen on Fig 5(a). Indeed, having to reach for farther RSUs implies longer transmission delays, visible in less populated network scenarios but compensated when more tasks are generated every second.

Fig 3(a) has shown that CAVTOMEC systematically reaches higher amounts of CPU usage compared to the location-based mechanism, with this behavior being even more observable with more complex tasks. A higher amount of CPU usage signifies increased consumption, and thus more tasks being treated at a given moment: CAVTOMEC renders task offloading more efficient. This is made possible thanks to the updates on RSU CPU usage sent to each vehicle, optimizing destination selection.

Moreover, Fig 3(b) lets us conclude that the CPU usage at each MEC server fluctuates less with time in our architecture. This translates to a more constant exploitation of computing resources at the RSUs, giving more control over task treatment and thus more robustness in the case of unexpected high priority tasks being offloaded: a better delay guarantee can be obtained for urgent tasks.

It was understood from Fig 4 that task success rates were improved by our offloading solution. Consequently, fewer tasks need to be offloaded again after the first try, giving more efficiency to the system and allowing it to compute greater amounts of tasks in a given time frame.

It has been deducted from Fig 5 and Fig 6 that our task offloading solution is beneficial over a baseline mechanism when confronted to a highly populated scenario with complex tasks. The reason behind the fluctuating task execution delay throughout the measurement lies behind the combination of multiple factors. When the frequency of task emissions grows, as well as the complexity of tasks, the subsequent load on the LTE-V2X network becomes too much to guarantee lower transmission delays. However, the contribution of our solution allows for fewer task queuing and better offloading destination selection, which counterbalances the load-induced delay. This behavior explains the better performance of CAVTOMEC for more frequent, more complex tasks.

Overall, we can deduct that our V2N-enhanced task offloading mechanism with QoS-provisioning is more effective than a baseline, location-based offloading scheme. Even though the proximity-centered solution we used as a point of comparison benefits from the shortest transmission times using D2D at short distances, the benefits brought by our destination selection algorithm outweigh these gains by a margin large enough to consider it successful. It would be interesting to experiment on this solution with 5G-V2X architectures, to gather its full delay-reduction potential.

VII. Conclusion

Smart cities are currently being massively tested and deployed, calling for diverse, accessible and reliable applications. Henceforth, the conception of autonomous vehicles and, more particularly, shuttles, is meant to satisfy public appeal towards new-generation mobility while staying reassuring about their safety. In this paper, we proposed an optimized task-offloading mechanism with QoS-provisioning, relying on existing network protocols. Using a three-fold QoS classification, our proposal accomplishes better reliability and CPU availability at the edge servers when compared to the go-to V2I task offloading solution, thus allowing for more demanding applications such as more thorough road safety protocols to be implemented in smart shuttle / smart city contexts. Scaling our architecture up appears to be feasible while keeping positive results. In future works, we would plan to integrate RSU-to-RSU task backhauling as well as V2V offloading strategies to operate hand-in-hand with our current solution in order to withstand mobility even better. As part of the autOCampus project, we also plan to deploy a test-bed integrating CAVTOMEC in the near future.

Acknowledgment

This work was supported by the French Government in the framework of the major investment plan Territoires d'Innovation, an action by the Grand Plan d'Investissement embedded in the third wave of the Plan France 2030, by Toulouse Métropole and the neOCampus GIS.

References

[1] E. Kassens-Noor, Z. Kotval-Karamchandani, and M. Cai, "Willingness to ride and perceptions of autonomous public transit," *Transportation Research Part A: Policy and Practice*, vol. 138, pp. 92–104, 2020.

[2] L. Bréhon–Grataloup, R. Kacimi, and A.-L. Beylot, "Mobile edge computing for V2X architectures and applications: A survey," *Computer Networks*, vol. 206, p. 108797, 2022.

[3] IRIT, "autOCampus project," Sep 2019, accessed on 03.19.2022. [Online]. Available: https://www.irit.fr/autocampus/en/home/

[4] Z. Deng, Z. Cai, and M. Liang, "A Multi-Hop VANETs-assisted offloading strategy in Vehicular Mobile Edge Computing," *IEEE Access*, vol. 8, pp. 53 062–53 071, 2020.

[5] L. Tang, B. Tang, L. Zhang, F. Guo, and H. He, "Joint optimization of network selection and task offloading for vehicular edge computing," *Journal of Cloud Computing*, vol. 10, 2021.

[6] Y. Saleem, N. Mitton, and V. Loscri, "A Vehicle-to-Infrastructure Data Offloading Scheme for Vehicular Networks with QoS Provisioning," in *International Wireless Communications and Mobile Computing (IWCMC)*, 2021, pp. 1442–1447.

[7] J. Guo, W. Luo, B. Song, F. Yu, and X. Du, "Intelligence-sharing vehicular networks with mobile edge computing and spatiotemporal knowledge transfer," *IEEE Network*, vol. 34, no. 4, pp. 256–262, 2020.

[8] 5GAA, "C-V2X Use Cases: Methodology, examples and service level requirements," 5G Automotive Association, Tech. Rep., June 2019.

[9] Commsignia Inc., *ITS-OB4, Powerful V2X Onboard Unit*, 2020. [Online]. Available: https://www.commsignia.com/wp-content/uploads/2020/11/Commsignia_ITS_OB4_ Product-Brief_v0.9.5_22062020_web.pdf

[10] 3GPP, "Study on LTE-based V2X services (Release 14)," 2016, rel.14 v14.0.0.

[11] 5GAA, "Cellular-Vehicle-to-Everything (C-V2X): today and next steps," 5G Automotive Association, Tech. Rep., July 2020.

[12] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing (TMC)*, vol. 10, no. 1, pp. 3–15, January 2011.

[13] A. Virdis, G. Stea, and G. Nardini, *Simulating LTE/LTE-Advanced Networks with SimuLTE*, 01 2016.

[14] P. Lopez, M. Behrisch, and L. Bieker-Walz, "Microscopic traffic simulation using sumo," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575–2582.