A Cluster-based Virtual Edge Computation Offloading Scheme for MEC-enabled Vehicular Networks

Leontios Sotiriadis
University of West Attica, Dept. of Informatics and Computer Engineering
lsotiriadis@uniwa.gr

Basilis Mamalis
University of West Attica, Dept. of Informatics and Computer Engineering
vmamalis@uniwa.gr

Grammati Pantziou
University of West Attica, Dept. of Informatics and Computer Engineering
pantziou@uniwa.gr

ABSTRACT

Internet of Vehicles (IoV) has received a great deal of attention in recent years from many researchers. Recently, vehicular edge computing has been a new paradigm to support computation-intensive and latency-sensitive services in IoV. Moreover, with the cellular-vehicle to everything technology, many tasks and applications can be efficiently offloaded to another node for processing. In this paper a novel cluster-based virtual edge computation offloading scheme is proposed, which has as its main objective to efficiently find the most suitable multi-hop neighbor to act as a virtual edge computing (VEC) server for task offloading. The proposed scheme is initially based on the formation and maintenance of multi-hop clusters with high stability, whereas its efficiency is further enhanced by the local/distributed computations taking place where it's possible.

CCS CONCEPTS

• **Networks** → Algorithms; Network types; Ad hoc networks.

KEYWORDS

Internet of Vehicles, Multi-hop Clustering, Vehicular Edge Computing, Virtual Edge, Computation Offloading

ACM Reference Format:
Leontios Sotiriadis, Basilis Mamalis, and Grammati Pantziou. 2022. A Cluster-based Virtual Edge Computation Offloading Scheme for MEC-enabled Vehicular Networks. In *26th Pan-Hellenic Conference on Informatics (PCI 2022), November 25–27, 2022, Athens, Greece.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3575879.3575989

1 INTRODUCTION

The emergence of vehicular edge computing has been a new paradigm to support the rapidly growing computation-intensive and latency-sensitive services in intelligent transportation systems (ITS) [1]. However, there are still some challenges in conducting efficient edge computing tasks due to the high mobility of vehicles and the limited bandwidth in vehicular networks [2]. Smart vehicles are equipped with devices to connect with each other, and have powerful processing unit to execute computation tasks. It is forecasted that smart vehicles on the roads could reach nearly two billion by 2025, and each one may produce up to 30 terabytes of data every day. On the other hand, the workloads in the vehicles are not evenly distributed. For example, slow-moving vehicles have less computation tasks in a relatively stable road traffic context, whereas fast-moving vehicles have to process a much larger amount of data in a shorter time to guarantee a safe driving. Generally, the problem of the efficient utilization of the idle vehicle computational resources needs further investigations.

Improving task execution performance in computation offloading is one of the key challenges in vehicular edge computing. To face the challenge, researchers conduct many kinds of studies, and the first attempts were mainly focused on the energy-efficient allocation of computing resources [3], binary computation offloading [4], and partial computation offloading [5], usually following well studied theoretical techniques to optimize the task offloading decision (MDP, game theory, branch and bound etc. [6-8]). However, most of the existing studies only consider the offloading computation tasks from vehicles to edge servers, such as roadside units (RSU) or cloud servers (see also [9, 10]), and the problem of how to utilize the computing capabilities of moving vehicles efficiently in dynamic vehicular networks has not been sufficiently clarified yet.

Two of the most valuable recent attempts towards that direction are presented in [1, 2]. In [1] the authors propose Virtual Edge, which is an efficient scheme to utilize free computational resources of multiple nearby vehicles as a virtual server to facilitate collaborative vehicular edge computing. Extensive simulations are conducted to show the advantage of the proposed scheme in terms of the completion ratio and average task execution time. In [2] a cluster-based collaborative offloading scheme is proposed, in which the collaboration between the MEC servers and vehicles with idle computation resources is explored (both theoretically and experimentally). According to simulation results, the proposed scheme reduces offloading latency and energy consumption compared to other existing schemes. Several other related attempts can also be found in [11-14] where moving and parked cars with idle resources are viewed as collaborating/fog nodes for task offloading and the idea of vehicles as an infrastructure (VaaI) is introduced, as well as in [15-17] where relevant combined solutions of vehicular MEC with collaborative task offloading are further studied. The reader may also find some interesting discussions on joint resource allocation and computation offloading in [18-20].

In this work we present a novel virtual edge computation offloading scheme for MEC-enabled vehicular networks (i.e. in the form of Fig. 1) which is mainly based on clustering as well as on the efficient calculation of the companion times between the vehicles. The adopted cluster formation algorithm first creates highly stable

PCI 2022, November 25–27, 2022, Athens, Greece
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9854-1/22/11…$15.00
https://doi.org/10.1145/3575879.3575989

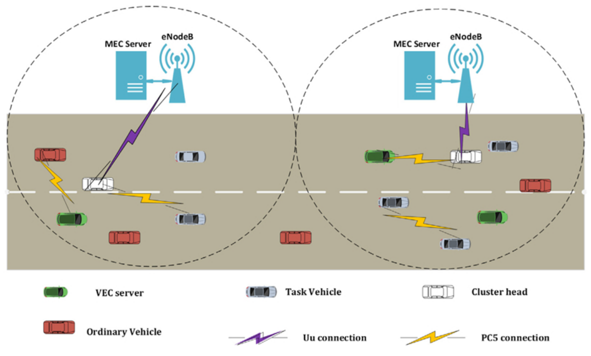


Figure 1: Basic IoV architecture with eNodeB/RSU and MEC server support [2]

and secure multi-hop clusters, and also uses a suitable maintenance mechanism to further increase the stability of the formed clusters. Given the underlying stable cluster organization, when a task is generated on a vehicle/member of a cluster, the main purpose of the proposed offloading scheme is to efficiently find another (the most suitable) cluster member (CM) that can act as a virtual edge computing (VEC) server for offloading the requested task. Local processing of the task or offloading to the closest MEC server is also conducted in specific cases. The efficiency of the proposed scheme is further enhanced by the local / distributed computations applied in specific time-consuming steps. The followed multi-hop clustering approach is briefly described in section 2. The proposed cluster-based computation offloading scheme as well as the detailed description of the distributed VEC-server finding procedure and its generalization are presented in section 3. Our evaluation plan is discussed in section 4, whereas section 5 concludes the paper.

2 THE CLUSTER FORMATION ALGORITHM

As mentioned above, our overall offloading approach is initially based on the formation of stable clusters, whose members are then considered as the best candidate VEC servers for task offloading. Specifically, a suitable clustering algorithm is adopted (proposed in our previous work of [21]), in which multi-hop clusters are formed, with not only stable CHs, but also having highly stable neighborhoods. This high stability feature is inherited recursively till the leaf nodes of the cluster. Also, a suitable RSU/MEC-server assisted maintenance mechanism is applied in order to further increase the stability of the formed clusters. Finally, an RSU/MEC-server assisted trust management mechanism is also introduced to ensure the resistance of the whole network to specific kinds of security attacks. In brief, the proposed clustering formation algorithm consists of the following steps:

i. First, each vehicle i computes its *Mobility Difference* (*MD value*) with each neighbor *j*, and constructs a set *U* of eligible neighbors, the ones for which the computed *MD* value doesn't exceed a specific threshold (*Ts*) and have the same direction. In this way it detects the most appropriate neighbors (i.e. the other ones in its neighborhood with the most similar mobility behavior), in order to choose later one of them to follow as its *parent* node. The *MD* value for each neighbor vehicle *j* with respect to *i* is computed based on the *speed* and *acceleration differences* (*SD(i,j) and, AD(i,j)* respectively – see [3] for their exact definition) between the two vehicles, which are the most important factors indicating the actual similarity in the mobility behavior of the two vehicles. The final *MD* value is given by the following formula, where c_1 and c_2 are appropriately selected constant coefficients (with $c_1 + c_2 = 1$):

$$MD\,(i,j) = c_1 \cdot SD\,(i,j) + c_2 \cdot AD\,(i,j)$$

ii. Each vehicle *i* also computes its overall Stability Factor (*SF*) value. The *SF* value of a vehicle is computed based on the total/average differences in speed and acceleration as well as on the degree of the vehicle and its total/average distance form all its neighbors. More concretely, it's computed as follows:

$$SF\,(i) = \alpha \cdot SD_{av}\,(i) + \beta \cdot AD_{av}\,(i) + \gamma \cdot D_{av}\,(i) + \delta \cdot d\,(i)$$

In the above computation α, β, γ and δ are (appropriately selected) constant coefficients, with $\alpha + \beta + \gamma + \delta = 1$, *d(i)* is the degree of node *i*, and $SD_{av}(i)$, $AD_{av}(i)$, $D_{av}(i)$ are the justified measurements of the average speed difference, average acceleration difference and average relative distance, respectively, between vehicle *i* and all its neighbors (see [3] for their exact definition).

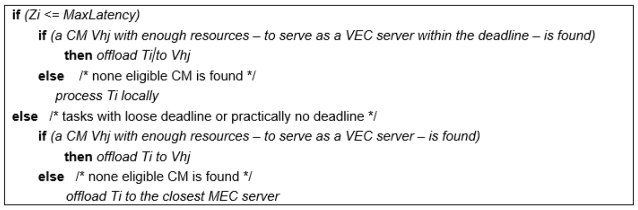
iii. Based on the advertised SF values of all it's neighbor vehicles, each vehicle *i* elects the best neighbor node to follow as its *parent* (among the ones already put in set *U* as eligible candidates). More concretely, *i* elects as its *parent* node (from the nodes of set *U*) the one which the maximum overall Stability Factor (SF) value, provided that this value is also greater than its own SF value.

iv. Finally, the vehicle which doesn't have a neighbor node with both sufficient mobility similarity (sufficiently small *MD* value) and greater *SF* value announces itself as a *clusterhead* (*CH*).

Thus, at the end of execution each vehicle has elected as its *parent* a vehicle that has sufficiently similar mobility behavior and the maximum overall stability among its neighbors. Based on this feature, which progressively extends till the root node (which is elected as the *CH*), the proposed algorithm naturally leads to the formation of highly stable clusters with increased lifetime. The reader may also refer to [21] for more details with regard to the stability maintenance and trust management mechanisms also adopted.

3 THE PROPOSED COMPUTATION OFFLOADING SCHEME

Let's assume a task Ti is generated on a vehicle *Vhi* and it's described by$T_i = (G_i, C_i, Z_i)$ where G_i denotes the size of Ti,Ci denotes the amount of computing resources needed to execute Ti, and Zi is the maximum latency of Ti To enhance network utility, vehicles can process task by themselves. Alternatively, and usually as a preferred option in order to exploit possible idle resources of neighboring vehicles, a vehicle *Vhi* may offload a task to another cluster member (CM, which in this case acts as virtual edge computing – VEC – server) or to the closest mobile edge computing (MEC) server for processing. Specifically, the host vehicle *Vhi* schedules the necessary communication with the other cluster members and calculates the necessary parameters to finally decide if offloading the task to another CM or to a MEC server is possible. We also distinguish between tasks that need to be completed within a strict deadline i.e., Zi =< max $Latency,$ and tasks that have a loose deadline or practically, have no deadline i.e., $Zi >$ max $Latency,$ where



```
if (Zi <= MaxLatency)
    if (a CM Vhj with enough resources – to serve as a VEC server within the deadline – is found)
        then offload Ti to Vhj
    else   /* none eligible CM is found */
        process Ti locally
else   /* tasks with loose deadline or practically no deadline */
    if (a CM Vhj with enough resources – to serve as a VEC server – is found)
        then offload Ti to Vhj
    else   /* none eligible CM is found */
        offload Ti to the closest MEC server
```

Figure 2: The basic offloading decision algorithm

MaxLatency is a predefined high threshold value meaning that any task with at least such latency practically will not fail in our algorithm. More concretely, the host vehicle *Vhi* acts for each task Ti as described in Fig. 2.

Note that in case of tasks with loose deadline or practically no deadline, it's not necessary to compute the *companion time* between vehicle *Vhi* and the candidate CMs (virtual edge computing servers), which is a quite time-consuming task, and the offloading decision procedure is further simplified. Also, in that case the results of the task execution may probably not sent back directly but through the closest MEC server. More details on the above cases as well as with regard to the exact procedure through which a specific CM is chosen to act as a VEC server for offloading are given in sections 3.2 and 3.3.

3.1 Local Processing

As mentioned above, to enhance network utility, vehicles can process task by themselves. However local processing is encouraged only if no other solution is feasible (i.e. offloading to another CM or to the closest MEC server is prohibited due to time limit restrictions or lack of the required resources). In that way the vehicle remains free and with enough resources to execute crucial tasks (i.e. tasks with strict deadlines/low latency values Z_i etc) either of its own or offloaded by other cluster members. Obviously, when a vehicle processes a task by itself there is no data transmission cost, so the delay is due only to execution time. Otherwise, the vehicle decides to offload a task to either a CM (virtual edge server) or a MEC server, which involves transmission cost through the wireless network.

3.2 Offloading to a MEC server

As indicated in the basic decision algorithm (Fig. 2), a decision for MEC server offloading is taken for tasks with loose (or practically no) deadline that no other CM can be found to execute them due to lack of resources. More concretely, if the host vehicle decides to offload task Ti to the closest MEC server, the whole transmission is then performed through the CH of the cluster the vehicle belongs to. Therefore, the task transmission is done in two phases. In phase 1, Ti is transmitted from the vehicle to the CH through the unique (multi-hop in general) path connecting them. In phase 2, Ti is transmitted from the CH to the MEC server. By involving the CH (and the underlying cluster structure) in the whole transmission procedure, our scheme guarantees the proper balancing of the communication overhead and eliminates congestions. Moreover, due to the stable and secure nature of the clustering algorithm, our approach leads to reliable communication routes between nodes for efficient task offloading, and hence minimizes communication overhead among vehicles.

The transmission of the results back to the host vehicle (*Vhi*) may be done in one of the following two ways:

- If the vehicle remains in the communication range of the MEC server the results are sent back to the vehicle in the same way as the offloading was done (following the opposite direction).
- If the vehicle lies now outside the range of the MEC server, the results are sent to the next (neighboring) MEC servers to finally disseminate them to the host vehicle when it comes into their range.

In both cases the whole transmission is performed again in two phases, through the CH of the cluster the vehicle belongs to.

3.3 Offloading to a Cluster Member (virtual edge computing server)

The basic offloading decision algorithm (which is illustrated in Fig. 2) clearly encourages finding another vehicle (with enough resources) to act as a VEC server for offloading the requested task. In the proposed computation offloading scheme the search for such a suitable VEC server is guided among the other members of the same cluster, trying to take advantage of the relevant attractive features of the adopted cluster formation algorithm (see section 2 for more details). More concretely, as mentioned in section 2, the relevant clustering procedure enforces the formation of stable and secure clusters, whereas the corresponding RSU/MEC-server assisted maintenance strategy enhances the overall clusters stability even more. Based on the above considerations, when a task is generated in a CM, the other CMs should be considered as the best candidates to act as VEC servers for task offloading, especially when strict deadlines have to be satisfied. In this case the companion time between the vehicle in which the task is generated and the vehicle in which the task is finally offloaded should be long enough to cover both the necessary processing time and the total communication times (required to offload the task and get back the results).

Specifically, the *companion time CTIJ* between two neighboring vehicles *Vhi* and *Vhj* is expressed as the upper limit of the link duration between the two vehicles and it's derived based on a general discretized longitudinal kinematic motion equation of vehicles as described in [1, 22]. Further, the companion time between two vehicles communicating through multiple relay nodes/vehicles (multiple hops – i.e. assume that *Vhi* communicates with *Vhj* through m relay nodes/vehicles $Vhr1 \ldots Vhr_m$ is expressed as follows.

$$CTij = min\left\{CTir_1, CTr_1r_{2,}CTr_2r_{3,...,}CTr_{m-1}r_{m,}CTr_mj\right\}$$

In other words, in that case the companion time is expressed as the minimum companion time among the companion times of all the consequent pairs of neighboring nodes on the path connecting *Vhi* and *Vhj*.

Additionally, the proposed scheme is enhanced suitably by applying local and distributed computations where it's possible. More concretely, both the calculation of the *companion times* of each other CM *Vhj* with the host vehicle *Vhi* and the estimation of the expected *processing times* on each other CM *Vhj,* are performed in a distributed manner in order to save both energy and time comparing to the alternative of making all the necessary calculations centrally in the host vehicle (*Vhi*), as for example is suggested in the work presented in [1].

The detailed searching procedures for each case (tasks with strict or loose deadlines) have as follows.

A. In case of tasks with strict deadlines ($Zi <= \max Latency$) *the following steps are taking place to find a suitable CM to act as a VEC-server:*

A1. First, the host vehicle *Vhi* disseminates the necessary information (of both itself and task *Ti*) to all CMs as the root of the cluster tree. More concretely, the relevant info message INFO_MSG consists of (a) the vehicle movement values *(Li, Vi, Ai)* where *Li* denotes its current location, and *Vi, Ai* denote its current velocity and acceleration respectively, and (b) the task description values (G_i, C_i, Z_I) where *Gi* denotes the size of *Ti*, *Ci* denotes the amount of computing resources needed to execute *Ti*, *Ci* , and *Zi* is the maximum latency of *Ti* . Moreover, in order to optimize the distributed calculation of the companion times between *Vhi* and the other CMs, while disseminating the *info message* each relay node *Vhr* substitutes the vehicle movement values contained in the message with its own corresponding values (Lr, Vr, Ar).

A2. Each vehicle *Vhj* that receives such an INFO_MSG message, calculates the following parameters:

(i) A binary flag *Fprj* denoting if it has enough resources (memory, cpu etc.) to process task *Ti* or not (taking value '1' or '0' respectively). This flag is calculated based on the remaining resources of its own and the values *Gi, Ci* just received.

(ii) The estimated time *Tprj* it needs to process task *Ti.* The corresponding calculation is also based on the remaining resources of its own and the values *Gi, Ci* just received, and it's being performed only if F*prj* is equal to '1'. Also, in case that *Tprj* is greater than *Zi Tprj* is set to '0' (to indicate that *Ti* isn't possible to be processed by *Vhj* within the relevant deadline).

(iii) The estimated *companion time CTjk* between itself and the relay node (let's name it *Vhk)* from which the info message was received (i.e the last node in the path connecting *Vhi* and *Vhj*). The calculation of the *companion time* is being performed only if F*prj* is equal to '1' and *Tprj* is greater than '0'. Also, in case that *Tprj* is greater than *CTjk*, *CTjk* is set to '0' (to indicate here too that *Ti* can't be processed by *Vhj* within the relevant deadline since the companion time between the two vehicles will be definitely not enough).

A3. Each vehicle *Vhj* that has received the INFO_MSG message and has already calculated the above parameters sends back a *response message* RES_MSG to the host vehicle *Vhi*, based on the values *Fprj*, *TPrj* and *CTjk.* More concretely, if *Fprj* is equal to *'1'* and both *Tprj* and *CTjk* are greater than '0', RES_MSG consists of the values *Tprj* and *CTjk*, whereas in any other case a zero-value RES_MSG is returned. Moreover, to suitably support the optimized distributed calculation of the companion times, while forwarding back the *response message* each relay node *Vhr* substitutes the value of the *companion time* contained in the message with the minimum between that value and its own corresponding value *(CTjk', where Vhk' is* the last node in the path connecting *Vhi* and *Vhr).*

A4. The host vehicle *Vhi,* for each *RES_MSG message* it receives (corresponding to the response sent by a cluster member vehicle *Vhj*), it first checks if it's not a zero-value message, and if so, it calculates the final *completion time Tcj* in case that *Ti* is offloaded to vehicle *Vhj.* More concretely, *Tcj* is calculated as the sum of the value *Tprj* and the estimated transmission time *Ttrj* to offload *Ti* to *Vhj* and get back the results. If *Tcj* is not greater than the estimated companion time *CTij*, the vehicle *Vhj* is regarded as an eligible CM that than can act as virtual edge computing server for task *Ti.* Task *Ti* is finally offloaded to the vehicle with the shortest *completion time Tc.*

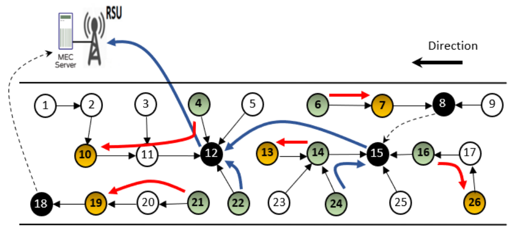


Figure 3: Example of the proposed task offloading procedure

B. In case of tasks with loose deadline or practically no deadline ($Zi >$ maxLatency) *the relevant procedure is simplified as follows:*

B1. In the first step, only the task description values (*Gi, Ci*) are disseminated to all CMs.

B2. In the second and third steps, only the flag *Fprj* and the value *Tprj* are calculated (by each other cluster member – except *Vhi*) and then returned (through the *RES_MSG* message) to the host vehicle *Vhi.*

B3. In the last step, among all the vehicles which returned a non-zero *RES_MSG* message, the vehicle *Vhj* with the shortest *processing time Tprj* is chosen for offloading task *Ti.*

Here also a special case has to be considered; specifically, the case in which the connection between the two vehicles *Vhi* and *Vhj* has been lost when the results have to be returned to *Vhi.* In that case, the results should be sent from *Vhj* (through its *CH*) to the closest MEC server and then sent back to *Vhi* either by the same or the next (neighboring) MEC servers when it comes into their range.

An overview of the proposed task offloading scheme is given in Fig. 3.

More concretely, in Fig. 3 an example of the cluster structure is first shown; the *black-colored* nodes are the CHs elected through the cluster formation procedure, and the rest ones are the cluster members (forming four clusters in total). Further, the *green-colored* nodes are the ones which have tasks to be completed, whereas the *orange-colored* nodes are the nodes to which a task has been finally offloaded. Nodes 4, 6, 14, 16 and 21 have finally offloaded their tasks to nodes 10, 7, 13, 26 and 19 respectively (as implied by the red arrows), whereas nodes 22 and 24 have offloaded their tasks to the closest MEC server (through the intra-cluster and inter-cluster communication pattern implied by the underlying cluster structure – see also [21] for more details). Before taking their offloading decision, nodes 4, 6, 14, 16, 21, 22 and 24 have been assumed to execute the basic decision algorithm implied by Fig. 2, as it is stated in more details in the present section. Note that during the execution of the algorithm each one of the source nodes (4, 6, 14, 16, 21, 22 and 24) first disseminates the necessary info message to all the other cluster members (acting as the root of its cluster tree) and then waits for the relevant responses as implied by the details of the proposed algorithm.

3.3.1 Generalization for multiple tasks / subtasks. The proposed scheme mainly focuses on an optimized distributed algorithm for finding a VEC-server to offload a single task; and also fits appropriately in the case of multiple tasks that may appear in periodic time intervals in the host vehicle.

However, it can also be appropriately generalized for (a) the case of multiple concurrent tasks (i.e. tasks that are being ready to execute at the same time in the waiting queue of the host vehicle) or (b) the case of a large task that may be initially divided into a number of subtasks before offloading. For such cases, beyond the straightforward solution of executing the proposed algorithm separately for each task/subtask (which keeps the fully highly distributed execution advantage, however it may not be regarded sufficiently efficient), a more comprehensive modification of our basic scheme may be followed, as described below.

- The estimation for each other CM *Vhj* whether it has enough resources to process one or more tasks/subtasks of *Vhi* (calculation of the *Fprj* flags), as well as the calculation of the expected processing time *Vhj* needs to process each task/subtask (values *Tprj*), should be performed centrally by vehicle *Vhi* itself (instead of the local/distributed calculation by each *Vhj* separately). To achieve that, instead of disseminating the task description values to all the CMs, the resource description values of each CM should be sent to the host vehicle *Vhi*.
- In this way, the optimized treatment of the waiting tasks/subtasks as a batch (i.e. leading to probable offloading of groups of tasks/subtasks on the same CM/vehicle Vhj etc.) would be feasible in the most effective manner (as well as highly efficient, since multiple message exchanging separately for each task/subtask would be avoided), following for example some of the existing relevant approaches presented in the literature [1, 2, 10, 20].

Note also that the corresponding *companion time*s (*CTij* values) would still be computed distributedly (not centrally as proposed for example in [1]), and only once for each group of tasks/subtasks waiting in the queue of the host vehicle, thus leading to an even more efficient offloading scheme.

4 PERFORMANCE EVALUATION

The proposed scheme is being evaluated through extended simulations. Our simulation process involves OMNeT++, Veins and SUMO simulation tools as follow. OMNeT++ [23] is a well-established network simulation framework available as open-source software for academic usage. Because of its modular approach, it has been extended by many third-party frameworks focusing on specialized communication technologies like LTE and IEEE 802.11p. Veins [24] is an open-source model library for (and a toolbox around) OMNeT++, which supports researchers conducting simulations involving communicating road vehicles—either as the main focus of a study or as a component.

Veins already includes a full stack of simulation models for investigating cars and infrastructure communicating via IEEE 802.11 based technologies in simulations of VANET and ITS. Traffic simulation in Veins is performed by the microscopic road traffic simulation package SUMO [25], which can import city maps from a variety of file formats and allows high-performance simulations of huge networks with roads consisting of multiple lanes, using simple right-of-way rules or traffic lights etc.

For the performance evaluation an area from Athens, GR, is extracted. Simulation parameters to be set include (a) general parameters, such as simulation time/area, vehicles distribution, total number of vehicles, MAC protocol, total number and distribution of RSUs/MEC servers, communication ranges etc., (b) parameters relevant to the cluster formation procedure, such as MD, SF and Trust thresholds, c1,c2 and $\alpha, \beta, \gamma, \delta$ coefficients etc., and (c) parameters relevant to the computation offloading procedure, such as tasks requirements (*Gi,Ci, Zi*), *MaxLatency*, vehicles computation resources etc. The completion ratio of the offloaded tasks and the average task execution time are primarily measured, as well as the total number of exchanged messages and the average energy consumption. Extensive measurements are taken for varying number and resource values of vehicles, varying number and computation/latency requirements of the generated tasks, varying number of RSUs/MEC servers etc., which are then compared to other existing schemes of the literature (like [1] and [2]). The evaluation process is an ongoing work with promising results (especially in the case of dense clusters with low-to-medium speed vehicles and low-to-medium number of hops) in our first simulation experiments.

5 CONCLUSION

A novel cluster-based virtual edge computation offloading scheme for MEC-enabled vehicular networks is presented throughout the paper. The efficiency of the proposed scheme is based on the formation of highly stable multi-hop clusters (whose members are suitably requested to act as virtual edge computing – VEC – servers for task offloading) as well as on the local/distributed computations taking place where it's possible. The completion of the evaluation process, as well as the efficient generalization of the proposed scheme for multiple concurrent tasks/subtasks in the context of the total edge power of the underlying cluster structure, are of high priority in our future work.

ACKNOWLEDGMENTS

The publication of this article was fully funded by the University of West Attica.

REFERENCES

[1] Cha, N., Wu, C., Yoshinaga, T., Ji, Y., Yau, K.A. 2021. Virtual edge: Exploring Computation Offloading in Collaborative Vehicular Edge Computing. IEEE Access 9:37739–37751.

[2] Bute, M. S., Fan, P., Liu, G., Abbas, F., Ding, Z. 2022. A cluster-based cooperative computation offloading scheme for C-V2X networks. *Ad Hoc Networks*, *132*, 102862.

[3] C. You, K. Huang, H. Chae, and B.-H. Kim. 2017. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411.

[4] S. Bi and Y. J. Zhang. 2018. Computation rate maximization for wireless powered mobile-edge computing with binary computation of_oading. *IEEE Trans. Wireless Commun.*, vol. 17, no. 6, pp. 4177–4190.

[5] Y. Wang, M. Sheng, X. Wang, L. Wang, and J. Li. 2016. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Trans. Commun.*, vol. 64, no. 10, pp. 4268–4282.

[6] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, M. Bennis. 2019. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning, IEEE Internet Things J. 6 (3), pp. 4005–4018.

[7] Z. Hong, W. Chen, H. Huang, S. Guo, Z. Zheng. 2019. Multi-hop cooperative computation offloading for industrial IoT–edge–cloud computing environments, IEEE Trans. Parallel Distrib. Syst. 30 (12), pp. 2759–2774.

[8] X. Yang, X. Yu, A. Rao. 2019. Efficient energy joint computation offloading and resource optimization in multi-access MEC systems. In Proc. 2019 IEEE Intl. Conf. on Electronic Information and Communication Technology (ICEICT), Harbin, China, pp. 151–155.

[9] J. Zhao, Q. Li, Y. Gong, K. Zhang. 2019. Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks. IEEE Trans. Veh. Technol. 68 (8), pp. 7944–7956.

[10] H. Guo, J. Zhang, J. Liu. 2019. Fiwi-enhanced vehicular edge computing networks: Collaborative task offloading. IEEE Veh. Technol. Mag. 14 (1), pp. 45–53.

[11] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, S. Chen. 2016. Vehicular fog computing: A viewpoint of vehicles as the infrastructures. IEEE Trans. Veh. Technol., vol. 65, no. 6, pp. 3860–3873.

[12] Y. Wang, K. Wang, H. Huang, T. Miyazaki, S. Guo. 2019. Traffic and computation co-offloading with reinforcement learning in fog computing for industrial applications. IEEE Trans. Ind. Informat., vol. 15, no. 2, pp. 976–986.

[13] X. Wang, Z. Ning, and L.Wang. 2018. Offloading in Internet of vehicles: A fog-enabled real-time traffic management system. IEEE Trans. Ind. Informat., vol. 14, no. 10, pp. 4568–4578.

[14] Z. Ning, J. Huang, and X. Wang. 2019. Vehicular fog computing: Enabling real-time traffic management for smart cities. IEEE Wireless Commun., vol. 26, no. 1, pp. 87–93.

[15] G. Qiao, S. Leng, K. Zhang, Y. He. 2018. Collaborative task offloading in vehicular edge multi-access networks. *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 48–54.

[16] C. Wu, Z. Liu, D. Zhang, T. Yoshinaga, Y. Ji. 2018. Spatial intelligence toward trustworthy vehicular IoT. IEEE Commun. Mag., vol. 56, no. 10, pp. 22–27.

[17] Y. Sun, J. Song, S. Zhou, X. Guo, and Z. Niu. 2018. Task replication for vehicular edge computing: A combinatorial multi-armed bandit based approach. In Proc. IEEE Global Commun. Conf. (GLOBECOM), Abu Dhabi, United Arab Emirates, pp. 1–7.

[18] C. Wang, C. Liang, F. R. Yu, Q. Chen, L. Tang. 2017. Computation offloading and resource allocation in wireless cellular networks with mobile edge computing. *IEEE Trans. Wireless Commun.*, vol. 16, no. 8, pp. 4924–4938.

[19] J. Yan, S. Bi, Y. J. Zhang, M. Tao. 2020. Optimal task offloading and resource allocation in mobile-edge computing with inter-user task dependency. *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 235–250.

[20] X. Xu, Y. Xue, L. Qi, Y. Yuan, X. Zhang, T. Umer, S. Wan. 2019. An edge computing-enabled computation offloading method with privacy preservation for Internet of connected vehicles. Future Gener. Comput. Syst., vol. 96, pp. 89–100.

[21] L. Sotiriadis, B. Mamalis, G. Pantziou. 2021. Stable and Secure Clustering for Internet of Vehicles with RSU-assisted Maintenance and Trust Management. PCI 2021: 25th Pan-Hellenic Conference on Informatics. Nov. 2021, pp. 124–129.

[22] B. Khondaker, L. Kattan, "Variable Speed Limit: A Microscopic Analysis in a Connected Vehicle Environment," Transp. Res. C, Emerg. Technol., vol. 58, Sep. 2015, pp.146–159.

[23] OMNeT++ Official Documentation. Accessed: Sept. 7, 2022. [Online]. Available: https://omnetpp.org/documentation/

[24] Veins. The open source vehicular network simulation framework. Accessed: Sept. 7, 2022. [Online]. Available: https://veins.car2x.org/

[25] SUMO. Simulation of Urban Mobility. Accessed: Sept. 7, 2022. [Online]. Available: http://sumo.dlr.de/wiki/Simulation_of_Urban_Mobility