DeTTO: Dependency-Aware Trustworthy Task Offloading in Vehicular IoT

Prajnamaya Dass, *Graduate Student Member, IEEE* and Sudip Misra, *Fellow, IEEE*

***Abstract*—In this paper, we investigate the dependency-aware trustworthy task offloading problem (DeTTO), especially in an IoT-enabled vehicular network, where a large computation-intensive task offloaded from a vehicle is fragmented into multiple subtasks and then offloaded to multiple trusted nodes. First, we formulate the task offloading problem as a graph optimization problem intending to find an optimal set of trustworthy nodes for offloading the subtasks. We aim to minimize the task completion delay and energy consumption, while satisfying the dependency relations between the subtasks and the trust requirements of the tasks. We consider three types of dependency structures – fully independent task, fully dependent task, and partially dependent task. For a fully independent task with no dependency between the subtasks, we propose a greedy algorithm to get the optimal set of nodes for task offloading. After showing the NP-hardness of solving the dependent task offloading problem, we propose a two-fold efficient heuristic approach for the tasks with all dependent subtasks. We adopt the solution approaches used by the first two types of tasks for a partially dependent task. Through simulation experiments, we analyze the performance of the proposed algorithms for three types of intra-task dependencies. The experimental results show that the proposed algorithms significantly reduce the delay and energy consumption, when compared to the benchmark schemes.**

***Index Terms*—Task Offloading, Trust, Task Type, Task Dependency, Vehicular IoT.**

I. Introduction

The current trend of computing techniques in vehicular IoT is shifting towards edge computing, wherein most of the computations are performed at the roadside units (RSUs) and the services are brought close to the vehicles [1]. Recent studies consider the RSUs as computationally capable entities in vehicular applications [2]. In an IoT-enabled vehicular network, the on-road moving vehicles offload the computation-intensive tasks to the nearby RSUs. With the help of distributed computing, larger tasks are further divided into subtasks and offloaded to different RSUs for parallel processing [3]. Then, the RSUs, through mobility-aware service provisioning, propagate the services to the vehicles at their current locations [4]. Recent researches consider delay and energy consumption as the main factors while taking offloading decisions [5]–[8]. However, other factors, such as dependency between subtasks, vehicle service destination, and type of the task offloaded play important roles in task offloading. Further, the trustworthiness of the RSUs is also crucial, when sensitive tasks are considered for offloading. In this work, the tasks from the vehicles are offloaded to the computationally capable nodes, known as RSUs. The RSUs cooperatively execute the fragmented task, known as subtasks, and provide real-time vehicle services.

P. Dass and S. Misra are with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, India.
E-mail: prajnamaya@iitkgp.ac.in, smisra@cse.iitkgp.ac.in

A. Motivation

A typical task offloading process in an IoT-enabled vehicular network is depicted in Figure 1(a). A vehicle offloads the task to the nearest RSU, which we consider as the offloading source and denote it as *src*. The *src* fragments the task into multiple subtasks and offloads them to different RSUs. Finally, the task output is delivered at the RSU near to the current location of the vehicle, denoted as *dst*. However, the dependencies between the subtasks are not considered, which highly affects the communication delay for delivering the output of one subtask to another dependent subtask. Further, the vehicular network is prone to many unreliable entities such as attackers, malicious, and selfish nodes [8]. Tasks offloaded to untrusted nodes may adversely affect the desired services of the vehicle. The trust level requirements of subtasks differ based on their data sensitivity, the required quality of service, and significance of the subtask in the task output [9], [10].

In real scenarios, there are some types of tasks, usually the data-oriented tasks [11], where the output size is much higher compared to its size before execution. For instance, a vehicle requesting satellite images and traffic data of a specific location, where the task input is a small request and the output contains large size images and traffic data. Consider the data subtask $t1$ in Figure 1(b), which has a trust requirement of 0.7 and hence, can be offloaded to RSU 1 or 2 or 4. If we select RSU 1 for offloading $t1$, the transmission delay and transmitting energy for 400 KB data will be added at four nodes in the path to the destination. However, if we choose RSU 4, the delay and energy consumption for data transmission from RSU 1 to RSU 3 will be added only for 10 KB data, and only RSU 4 will have the transmission delay for 400 KB data. On the other hand, there exist some processing tasks [12], where the output size is much less compared to its size before execution. For instance, a car sends different sensor data values such as speed, fuel consumption, emissions, and security, to the RSU. In return, the car requests the RSU for high-size data processing and only asks for emergency indications of significantly small size. The subtask $t2$ in Figure 1(b) can be offloaded to RSU 2 or 4. However, the optimal selection should be RSU 2 as it reduces the transmission delay and energy consumption in the path.

In Figure 1(c), we consider the dependency between the subtasks, types of subtasks, and trust level of RSUs during task offloading. The subtasks, $t1$, $t2$, and $t4$ are independent

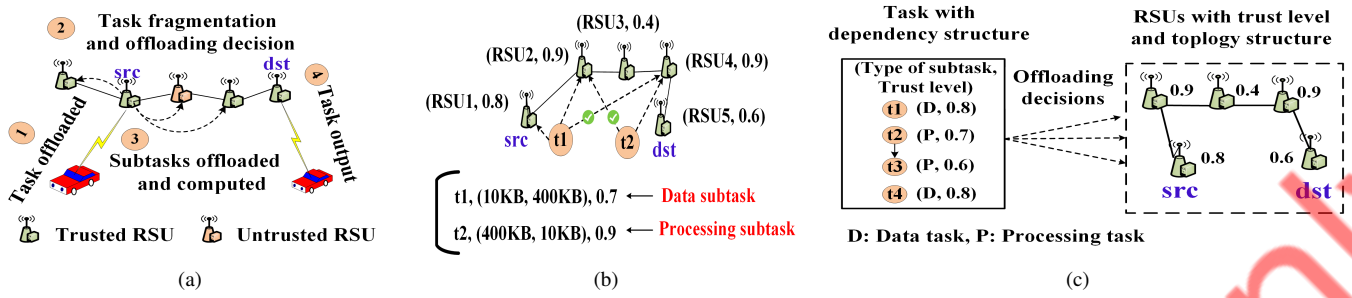


Fig. 1: Motivating scenarios: (a) existing approach on task offloading in IoT-enabled vehicular network; (b) task types and their significance in task offloading; and (c) proposed approach with trust level, task type, and dependency between the subtasks.

subtasks whereas $t3$ depends on the output of $t2$. The intra-task dependency structure can be represented in a directed acyclic graph (DAG). However, it is computationally hard to solve the DAG scheduling problem. Therefore, we use topological sorting to convert a DAG into a dependency graph with either sequential dependency or no dependency among the subtasks.

B. Contributions

Motivated by the above-mentioned points, in this work, we propose a novel task offloading approach for vehicular network, where trustworthiness, intra-task dependency, and task types are considered for task offloading. In particular, the main contributions of our work are as follows:

- Given the task offloaded location (source) and the location of service delivery (destination), we formulate the task offloading problem with intra-task dependency as a graph optimization problem. We also consider other constraints, such as subtask types, trust level requirement of the subtasks, and the trust score of the RSUs.
- Based on the dependencies, we consider three types of tasks – i) fully independent task, ii) fully dependent task, and iii) partially dependent task and solve all of them with different approaches. We propose a greedy approach to find optimal offloading decision for a fully independent task, where there is no dependency between the subtasks.
- We show the hardness of finding optimal offloading decision for a task with sequential dependency among the subtasks and propose a two-fold efficient heuristic algorithm. We solve the partial dependent tasks using both the greedy and heuristic approach.
- Through extensive numerical simulations, we show the performance of the proposed algorithms. The results show that proposed task offloading algorithms greatly reduce the delay and energy consumption.

II. Related Works

As a promising technology, edge and fog computing support the execution of computation-intensive tasks offloaded from the IoT devices, which attracted significant researches on task offloading. Zhan *et al.* [4] proposed a mobility-aware approach for task offloading and resource allocation among multiple moving users, which are served by a base station. Due to the hardness of the decision problem, the authors proposed a partial order-based heuristic to reach an approximate offloading decision. In another work, Misra and Bera [16] proposed an SDN-assisted mobility-aware task offloading mechanism for the vehicular network. The authors considered the location of the mobile vehicle before and after the offloaded task execution. Guo *et al.* [7] studied the problem of computational intensive task processing at the vehicles and proposed a vehicular edge computing network architecture. The authors proposed a deep learning approach for optimal task offloading, where the SDN-enabled architecture supports information collection and centralized management.

Ouyang *et al.* [8] proposed a trust-aware task offloading method, where an IoT device in a cluster with higher trust value is selected for task offloading. Trust evidence for each node is collected and evaluated in a timely manner, which helps to avoid malicious nodes during task offloading. Another trust-aware task offloading mechanism was proposed by Wu *et al.* [15], which considers the trustworthiness of the nodes while minimizing the delay and energy consumption. The proposed framework consists of three phases – trust evaluation, filteration of untrusted nodes, and optimal node selection.

Recently, Li *et al.* [6] studied the problem of energy consumption during the offloading of tasks with multiple subtasks. The authors first compute the expected energy consumption and then find an optimal offloading strategy. Similarly, a delay-aware task offloading in a fog shared network was proposed by Jiang and Tsang [5], where heterogeneous delay requirements of the tasks are considered. Their work also integrated the task migration cost due to the changes in system dynamics. Liu *et al.* [13] proposed a task scheduling algorithm, where the interdependent tasks from the vehicles are offloaded to the mobile edge computing servers on the RSUs. Another interdependent task offloading approach was proposed by Han *et al.* [14], where a heuristic approach was proposed to minimize the task latency and energy consumption.

Synthesis: Our work differs from the above works in the following aspects – i) the existing works only consider the task size before execution. On the contrary, in this work, we consider the task types, which gives information about the task size after execution and helps in reducing the delay and energy consumption, ii) we reach the offloading decisions, while considering the dependency structure of the tasks and trustworthiness of the RSUs in the path to the service destination, which was not considered in the previous works.

TABLE I: Summary of Existing Works

Work	Service Destination	Multi-Hop	Trust	Task Dependency	Task Type	Delay	Energy
Li *et al.* [6]	✗	✗	✗	✗	✗	✓	✓
Jiang and Tsang [5], Guo *et al.* [7]	✗	✗	✗	✗	✗	✓	✗
Liu *et al.* [13]	✗	✗	✗	✓	✗	✓	✗
Han *et al.* [14]	✗	✗	✗	✓	✗	✓	✓
Wu *et al.* [15]	✗	✗	✓	✗	✗	✓	✓
Ouyang *et al.* [8]	✗	✓	✗	✗	✗	✓	✓
Zhan *et al.* [4]	✓	✗	✗	✗	✗	✓	✓
Zhang *et al.* [1]	✓	✗	✗	✗	✗	✓	✗
Misra and Bera [16]	✓	✓	✗	✗	✗	✓	✓
DeTTO (Proposed Work)	✓	✓	✓	✓	✓	✓	✓

III. System Model

The RSU infrastructure of the vehicular network is modeled as an undirected graph $G_N = (\mathcal{V}_N, \mathcal{E}_N)$, where $\mathcal{V}_N$ is the set of vertices, each of which represents an RSU and $\mathcal{E}_N \subseteq \mathcal{V}_N \times \mathcal{V}_N$ is the set of weighted undirected edges representing the connection between the RSUs based on their area coverage [2]. Each RSU in the network has information about the connectivity to other RSUs and knows the shortest path to an RSU. An on-road moving vehicle places the task offloading request at the nearest RSU, considered as the source (src). The source RSU decides the offloading choices of the subtasks, considering their dependencies and trust requirements.

Destination prediction: The destination, where the output of the executed task is to be delivered, is either explicitly mentioned by the vehicle users or computed through some location prediction algorithm [16]–[19]. The cases where the destination location is not sent by the vehicle, we adopt the order-g Markov-predictor proposed in work [16] to predict the service destination (dst) of the vehicle. The reason for such selection is that the Markov predictor consumes less space and gives better accuracy compared to other location predictors for low values of g [20]. The predictor considers most recent g location contexts and computes the conditional probability that the vehicle with source RSU v_x will be associated to the RSU v_y after time period δ^{-}_{max}. Here, v_y is the RSU associated with the vehicle before the task completion deadline δ_{max}. The transition probability for the same is computed as [16]:

$$P(dst = v_y|\theta, \mathcal{H}) = \frac{\#(\theta v_y, \mathcal{H})}{\#(\theta, \mathcal{H})} \tag{1}$$

where $\#(\theta v_y, \mathcal{H})$ represents total occurrences of the context θ in the history set $\mathcal{H}$. The most likely RSU as the destination from all the possible RSU $v_y \in \mathcal{V}_N$ is computed as:

$$dst = \arg\max_{v_y \in \mathcal{H}} P(dst = v_y) \tag{2}$$

Consider another directed graph $G_T = (\mathcal{V}_T, \mathcal{E}_T)$, where $\mathcal{V}_T$ is the set of subtasks and $\mathcal{E}_T \subseteq \mathcal{V}_T \times \mathcal{V}_T$ is the link set representing the dependency between the subtasks. Each RSU $v \in \mathcal{V}_N$ in the network has a trust reputation $\pi_v \in [0, 1]$. Similarly, each subtask $i \in \mathcal{V}_T$ has a trust requirement $\Psi_i \in [0, 1]$ and a subtask can only be offloaded if $\pi_v \geq \Psi_i$. We define each subtask as 5-tuple $\Omega_i = \langle \chi_i, w_i, \Psi_i, S_i^I, S_i^O \rangle$, where χ_i represents the task type – data ($\chi_i = 0$) or processing task ($\chi_i = 1$), w_i represents the CPU cycles needed to execute the subtask i, and Ψ_i is the trust requirement of the subtask, S_i^I and S_i^O represent the input and output size of the subtask i, respectively. For two dependent subtasks Γ_i and Γ_{i+1}, $S_{i+1}^I = S_i^O$. However, the subtask sizes and the CPU cycle requirements are not always apriori available, and we will see in Section IV that even without this knowledge, using the task types, the solution approach works correctly. The reachability from a node u to another node v is represented as:

$$path(u, v) = \begin{cases} 1, & \text{if } v \text{ can be reached from } u \in G_N \\ 0, & \text{otherwise} \end{cases}$$

We denote $dep[\Gamma_i, \Gamma_j]$[1] $= 1$, if Γ_j depends on Γ_i, otherwise $dep[\Gamma_i, \Gamma_j] = 0$. An independent subtask i can be defined as a vertex with no incoming edges and represented as $ind[i] = 1$. Therefore, if $ind[j] = 0$, there exist some tasks i where $dep[\Gamma_i, \Gamma_j] = 1$. Considering a subtask with dependency, $dep[\Gamma_i, \Gamma_j] = 1$, the subtasks i and j can be offloaded to RSUs u and v, only if $path(u, v) = 1$ in graph G_N. For fair use of the RSU resources, we enforce a constraint of maximum k subtasks of a particular task that can be offloaded to an RSU. We use a variable $offload(i)$, which returns the RSU index offloaded with the subtask i. We use another binary variable $offload(i, v)$ to denote whether i^{th} subtask is offloaded to RSU v or not. Therefore, $offload(i, v) = 1$, if i^{th} task is offloaded to RSU v and $offload(i, v) = 0$, otherwise.

TABLE II: Summary of Key Notations

Notation	Description
$\mathcal{V}_T$	Set of vertices representing subtasks in task graph G_T
$\mathcal{V}_N$	Set of vertices representing RSUs in network graph G_N
Γ_i	Profile of subtask i
w_i	CPU cycles needed to execute subtask i
S_i^I, S_i^O	Input size and approximate output size of i^{th} subtask
Ψ_i	Trust requirement of of subtask i
π_v	Trust score of RSU v
δ_i^{tx}	Transmission delay associated with i^{th} subtask
δ_i^{prop}	Propagation delay associated with i^{th} subtask
δ_i^{que}	Queuing delay associated with i^{th} subtask
δ_i^{exe}	Time taken to execute the i^{th} subtask
p_v^{tx}	Transmission power of RSU v
ε_i	Energy consumed for offloading subtask i
$\delta_i^x(v)$	Delay due to purpose x if subtask i offloaded to RSU v
$\varepsilon_i(v)$	Energy consumption if subtask i offloaded to RSU v
f_v	CPU frequency of RSU v
μ_v	Task service rate of RSU v
k	Maximum subtasks that can be offloaded to a node

[1]The brackets [] denote properties or known values, whereas () denotes the functions or variables.

A. Delay Model

The total delay for a task is the sum of all the delays of all its subtasks. For delay computation, we consider $offload(i-1) = u$ and $offload(i) = v$ for each dependent subtask $i \in \mathcal{V}_T$ and $u, v \in \mathcal{V}_N$. Similarly, for all independent subtasks i, we consider $offload(i) = v$.

The transmission delay δ_i^{tx} for offloading subtask i to node v is a function of task size, maximum data rate between two nodes, and the number of nodes in the path. For simplicity, we consider the data rate of each link B as the same. The transmission delay is computed as:

$$\delta_i^{tx} = \begin{cases} \frac{S_i^I}{B} \times n_{tot}(u,v), & 1 < i < m \\ \frac{S_i^I \times n_{tot}(u,v) + S_i^O \times n_{tot}(v,dst)}{B}, & i = m \\ \frac{S_i^I}{B} \times n_{tot}(src,v), & i = 1 \\ \frac{S_i^I \times n_{tot}(src,v) + S_i^O \times n_{tot}(v,dst)}{B}, & ind[i] = 1 \end{cases} \tag{3}$$

where the function $n_{tot}(x, y)$ returns the number of nodes in the path from node x to node y and m is the number of subtasks. The first three cases in Equation (3) are used to compute the transmission delay for dependent subtasks and the last case is used for independent subtasks.

The propagation delay experienced by each intermediate subtask i, $1 < i < m$, is computed as $\delta_i^{prop} = \frac{dist(u,v)}{S}$. Here, $dist(u, v)$ denotes the shortest distance between node u and v, and S denotes the propagation speed. For the first subtask $i = 1$, we consider the $dist(src, v)$ and for last subtask, i.e., $i = m$, the total distance is $dist(u, v) + dist(v, dst)$. For each independent subtask with $ind[i] = 1$, the total distance from (src, v) and (v, dst) is considered.

The time consumed for executing a subtask i at the offloaded node v is computed as $\delta_i^{exe} = \frac{w_i}{f_v}$, where w_i is the CPU cycles needed for subtask i and f_v represents the CPU frequency of node v. Consequently, the delay associated with each subtask i is computed as $\delta_i = \delta_i^{tx} + \delta_i^{prop} + \delta_i^{exe}$.

B. Energy Consumption Model

For energy consumption, we consider the energy consumed for offloading and executing the subtask at the offloaded node. The energy consumed for offloading a subtask i from node u is $p_u^{tx}\delta_i^{tx}$, where p_u^{tx} is the transmission power of RSU u and δ_i^{tx} denotes the transmission delay. However, the node selected for offloading may be placed multi-hop away from the current node. Therefore, the transmission energy consumption at each node in the path $\mathbb{P}$ is considered and computed as:

$$\varepsilon_i^{tx} = \sum_{q \in \mathbb{P}} p_q^{tx}\delta_i^{tx}, \text{where } \mathbb{P} = \begin{cases} path(src,v), & i = 1 \\ path(u,v), & 1 < i < m \\ path(u,v,dst), & i = m \end{cases}$$

Here, $path(u, v, dst)$ represents the path followed by the offloaded task from node u to dst through node v. Similarly, for all independent subtasks, the path from source to destination through the offloaded node v is considered. The computational energy consumed at the offloaded node v is computed as $\varepsilon_i^{comp} = \varphi(f_v)^2 w_i$, where φ is the energy coefficient depending on the chip architecture, f_v is the CPU frequency of RSU v, and w_i is the CPU cycles needed to compute subtask i [21]. Finally, the total energy consumed for offloading the subtask i is defined as $\varepsilon_i = \varepsilon_i^{tx} + \varepsilon_i^{comp}$.

C. Problem Formulation

Our objective is to minimize the delay and energy consumption of task offloading, while considering the service destination, task dependencies, and trust requirements. For all the independent subtasks, we consider the path from source to destination through the offloaded node v using Equation (4).

$$Y = offload(i,v) \times path(src,v) \times path(v,dst) \tag{4}$$

Similarly, for offloading each dependent subtasks i to node v, there should exist a path between the node u and v, where u is the node offloaded with $(i-1)^{th}$ subtask, represented as:

$$Z = offload(i-1,u) \times offload(i,v) \times path(u,v) \tag{5}$$

Using the delay and energy functions, we formulate the DeTTO problem as follows:

$$\textbf{minimize} \sum_{i=1}^{m} \sum_{v \in \mathcal{V}_N} \overbrace{ind[i] \times Y \times (\delta_i + \varepsilon_i)}^{\text{independent subtask}} + \overbrace{(1 - ind[i]) \sum_{u \in \mathcal{V}_N} Z \times (\delta_i + \varepsilon_i)}^{\text{dependent subtask}} \tag{6}$$

$$\text{s.t.} \quad Z = \begin{cases} Z, & \forall i \in \mathcal{V}_T \setminus \{m\} \\ Z \times path(v,dst), & i = m \end{cases} \tag{7}$$

$$Y, Z \in \{0,1\}, \forall i \in \mathcal{V}_T \tag{8}$$

$$offload(0) = src, \ offload(0,src) = 1 \tag{9}$$

$$\pi_v \geq \Psi_i, \forall \ offload(i) = v \tag{10}$$

$$\sum_{i=1}^{m} offload(i,v) \leq k, \forall v \in \mathcal{V}_N \tag{11}$$

$$\sum_{v=1}^{n} offload(i,v) = 1, \forall i \in \mathcal{V}_T \tag{12}$$

$$\sum_{i=1}^{m} \delta_i \leq \delta_{max} \tag{13}$$

The brief explanation of the DeTTO formulated model is as follows. Equation (6) defines the optimization objective, where all independent and dependent subtasks are handled differently. For m^{th} subtask, the additional path from node v to the destination is considered in Equation (7). In Equation (8), the binary properties of the variables Y and Z are mentioned, while considering the offloading decision and the path in the graph $\mathcal{V}_N$. Equation (9) states that the source node makes offloading decision and for subtask $i = 1$, the path from the source is considered. Equation (10) enforces the trust requirement of each subtask and for offloading, the node trust value must not be less than the required value of subtask. Equation (11) ensures that maximum k subtasks can be offloaded to a single node. However, a subtask can be offloaded only to a single node in the network, which is mentioned in (12). The maximum delay associated with a task should not exceed δ_{max}, which is enforced in Equation (13).

IV. Solution Approach

In this section, we propose solutions for the formulated problem, DeTTO. We consider three cases of the problem based on three types of intra-task dependency structures, as shown in Figure 2 and solve them through different approaches. Given any dependency task graph, which is a directed acyclic graph (DAG), we can convert it to one of the three considered task dependencies in Figure 2, after finding the topological sorting of the DAG. The source RSU finds the offloading decisions for each fragmented task. Along with the input task fragment, the offloading decisions are also delivered to the participating RSUs. Therefore, the executing RSU knows to which RSU it should deliver the output.

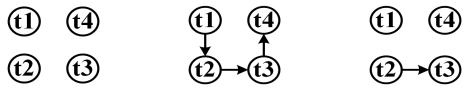


Fig. 2: Examples of task dependencies: (a) fully independent task, (b) fully dependent task, and (c) partially dependent task

A. Offloading of Fully Independent Tasks

We consider the independent subtasks as the first case, where all subtasks, Γ_i satisfy the dependency relation $ind[i] = 1, \forall i \in \mathcal{V}_T$. Therefore, we can follow any arbitrary order of offloading. The main idea is to consider the size of the subtasks and the shortest path to reach the destination. We compute the offloading cost of each node v using the following Equations.

$$\delta_i^{tx}(v) = \frac{\alpha \times n_{tot}(src, v) + \beta \times n_{tot}(v, dst)}{B} \tag{14}$$

$$\varepsilon_i(v) = \sum_{q \in \mathbb{P}(src,v)} \frac{\alpha \times p_q^{tx}}{B} + \sum_{q \in \mathbb{P}(v,dst)} \frac{\beta \times p_q^{tx}}{B} + \varphi(f_v)^2 w \tag{15}$$

$$\mathcal{U}_v = C_\delta \left(\delta_i^{prop}(v) + \delta_i^{tx}(v) + \frac{w}{f_v} \right) + C_\varepsilon \varepsilon_i(v) \tag{16}$$

We consider two integer values, α and β, as the skeleton to represent the input and output task size, respectively. All other values except the task size in Equation (16) has no effect on the offloading decision. A constant value of w CPU cycles is used to compute the computational delay and energy consumption. In Equations (14) and (15), we consider the case $\alpha > \beta$ for making the offloading decision of processing type task i with $S_i^I > S_i^O$ and denote the offloading cost of the node as $\mathcal{U}_v^{\alpha\beta}$. Similarly, we use another cost function $\mathcal{U}_v^{\beta\alpha}$ for the case $S_i^I < S_i^O$, where we consider $\alpha < \beta$ in the same Equations. The system parameters C_δ and C_ε represent the cost per unit delay and cost per unit energy, respectively.

In detail, Algorithm 1 works as follows. From the task deadline, the destination (*dst*) is computed using the Markov predictor. The predictor uses past mobility traces to predict the destination. In this work, we consider the Gauss-Markov mobility model to generate the mobility traces. We maintain two arrays $A[0][v]$ and $A[1][v]$ to store the offloading cost of each node $v \in \mathcal{V}_N$, while considering two different types

Algorithm 1: Offloading of Independent Tasks

Input: $\mathcal{V}_T, \mathcal{V}_N, src, dst, \Gamma, \alpha, \beta, k$
Output: $offload[i], \forall i$ with $ind[i] = 1$

1 **for** *each node* $v \in \mathcal{V}_N$ **do**
2 $counter[v] = 0$;
3 Compute $\mathcal{U}_v^{\alpha\beta}$ using $\alpha > \beta$ in Equation (16) ;
4 Compute $\mathcal{U}_v^{\beta\alpha}$ using $\alpha < \beta$ in Equation (16) ;
5 Store $A[0][v] = \mathcal{U}_v^{\alpha\beta}$ and $A[1][v] = \mathcal{U}_v^{\beta\alpha}$;
6 Sort $A[0][v]$ and $A[1][v]$ in increasing order of $\mathcal{U}_v^{\alpha\beta}$ $\mathcal{U}_v^{\beta\alpha}$;
7 Maintain node orders: $N[0][] \leftarrow A[0][v]$ and $N[1][] \leftarrow A[1][v]$;
8 **for** *each subtask* $i \in \mathcal{V}_T$ **do**
9 $offload[i] = 0$;
10 **for** $j \in [1, n]$ **do**
11 **if** $\pi[A[\chi_i][j]] \geq \Psi_i$ *and* $counter[j] \leq k$ **then**
12 $offload[i] =$N$[\chi_i][j]$;
13 $counter[j]$++ ;
14 return $offload[]$;

of relations, $\alpha > \beta$ and $\alpha < \beta$, respectively, to represent two task types. We sort both the arrays in increasing order and maintain the node ordering in two other arrays, namely, $N[0][]$ and $N[1][]$. Now, for each subtask, we consider the node with minimum offloading cost in the sorted order. However, the node should possess at least the trust requirement of the task and the number of offloaded tasks to the node should be less than k. If $\chi_i = 0$, the subtask is of processing type and if $\chi_i = 1$, it is a data-oriented subtask. The task type is used as the rows of the 2D arrays A and N. Once the offloading conditions are satisfied, we assign the task to a node and update the counter of that node.

Time complexity: For n number of nodes, Steps 1-5 take $O(n)$ time to compute the offloading cost of each node. The sorting algorithm takes $O(n \log n)$ time and the node order can be maintained during the sorting phase with constant time. A subtask is either data-oriented or processing type and therefore, Steps 8–13 take $O(n)$ time to traverse till the last of the array $N[\chi_i][]$, in the worst case. Considering m number of subtasks in Step 7, the time complexity for Steps 8–13 is $O(mn)$. Consequently, the total complexity of Algorithm 1 is $O(n+n \log n+mn) \approx O(mn)$ if $m < n \log n$ and $O(n \log n)$ when $m > n \log n$.

B. Offloading of Fully Dependent Tasks

As the second case of dependency, we consider all the dependent tasks, where for each subtask Γ_i, $ind[i] = 0$, $1 < i \leq m$, i.e., $dep[\Gamma_i, \Gamma_{i+1}] = 1$, $1 < i \leq m$. In this case, the solution approach should consider the task dependency graph and the optimization objective. The task dependency graph offloading problem is similar to the DAG job scheduling problem, which is NP-hard even for a general case [22], [23]. Therefore, finding an optimal set of nodes for offloading a fully dependent task is NP-hard.

Next, to solve the problem with NP-hardness, we propose an efficient heuristic approach. The basic idea of the solution approach is to consider the network topology of the RSUs. Usually, the RSUs are deployed to serve nearby on-road vehicles. Therefore, we consider the road routes from the source to the destination instead of all the RSUs forming

paths to the destination. The routes are static and available at every RSU. Our proposed approach works in two steps: in the first step, we check for the existence of the solution, and then, in the second step, optimize the offloading decision. First, select the route with the shortest distance from source to the destination, consider all the RSUs in the route, and assign unique indexing based on the RSU ordering in that route from source to destination. We use Algorithm 2 to check whether offloading of all the dependent subtasks is possible or not, considering the trust requirements. If there exists an offloading solution, we use Algorithm 3 to optimize the RSU selection for task offloading. However, if no solution exists in the first route, we check the next shortest route and so on. Furthermore, we also keep track of the subtasks for which no RSU is found in a route.

Algorithm 2: Offloading of Fully Dependent Tasks

Input: $\mathcal{V}_T, \mathcal{V}_N, \Gamma, k, routes, src, dst$
Output: $offload[i], \forall i$ with $ind[i] = 0$

1 $r = 1, off\_count = 0, last = 0, solution = 0$;
2 **while** $r <$ *total routes* **do**
3 **for** *each* $v \in route_r$ **do**
4 $counter[v] = 0$;
5 **for** *each subtask* $i \in \mathcal{V}_T$ **do**
6 $flag = 0$;
7 **for** $j \in [last + 1, n_r]$ **do**
8 **OFFLOAD**$(i, j, \pi_j, \Psi_i, off\_count, k)$;
9 **if** $flag == 1$ **then**
10 $last = j, status[r][i] = 0$;
11 **else**
12 **for** $j \in [last, 1]$ **do**
13 **OFFLOAD**$(i, j, \pi_j, \Psi_i, off\_count, k)$;
14 **if** *flag==0* **then**
15 $status[r][i] = 0$;
16 **if** *off_count==m* **then**
17 $solution = 1$; *break* ;
18 **else**
19 $route\_count(r) = off\_count$;
20 r++ ;
21 **if** $solution == 0$ **then**
▷ If no RSU found for a subtask
22 **FIND OFFLOAD**$(status, route\_count)$;
23 **OFFLOAD** $(i, j, \pi_j, \Psi_i, off\_count, k)$
24 **if** $\pi_j > \Psi_i$ *&& counter*$[j] < k$ **then**
25 $offload[i] = j$;
26 $flag = 1, counter[j]$++ ;
27 off_count++, $last = j$;
28 **else**
29 $flag = 0$;
30 return $flag$;
31 **FIND OFFLOAD**$(status, route\_count)$
32 $max = route\_count[1]$;
33 **for** $r \in [2, totalroutes]$ **do**
34 **if** $route\_count[r] > max$ **then**
35 $max = route\_count[r]$;
36 $max\_route = r$;
37 **for** *each subtask* $i \in \mathcal{V}_T$ **do**
38 **if** $status[max\_route][i] == 0$ **then**
39 Use Algorithm 1 to compute offload[i] ;

All the steps for offloading a fully dependent task are outlined in Algorithm 2. The sorted order of the routes based on the distance from source to destination is apriori available, which is computed once during the network deployment. We consider the nodes in the shortest route and check the offloading feasibility. Considering n_r nodes in route r, we set a boundary at the node that is assigned with the last subtask, denoted as $last$. If no eligible node is found between the nodes $[last + 1, n_r]$ then the nodes towards the source, i.e., nodes between $[last, 1]$, are considered. The subroutine OFFLOAD in Steps 23 – 30 checks the offloading conditions and if satisfied, updates the counter and the total number of tasks that find feasible nodes for offloading. In a particular route r, if all the subtasks can be offloaded, we get the solution and terminate. Otherwise, we proceed to the next shortest route and perform the feasibility check. We maintain the status of each subtask in each route to denote whether an RSU is found for offloading or not. If status[r][i]=1, a feasible RSU exists for the i^{th} subtask in the route r and the value is 0, if no RSU found. Finally, if $solution$ = 0, no route is found with a feasible RSU set for all the subtasks. In this case, we consider the route with the maximum subtask solution and use Algorithm 1 for the subtask with $status[r][i]$=0. The subroutine FIND OFFLOAD in Steps 31–39 finds the offloading decisions for the subtasks for which no feasible RSU is found in any of the routes.

Time complexity: Steps 3–4 initializes the counter array, which takes $O(n_r)$. For offloading a subtask, in worst case, we check all the nodes in the route. Therefore, Steps 5–15 take $O(mn_r)$ time. The subroutine OFFLOAD in Steps 23–30 takes $O(1)$ time. In the worst case, the while loop in step 2 runs for all the routes available, considering a source and destination. The subroutine FIND OFFLOAD uses Algorithm 1 and therefore, Steps 31–39 take $O(mn)$ time. Considering n number of nodes in the network, Algorithm 2 takes $O(mn^2 + mn) \approx O(mn^2)$ time.

Algorithm 3: Offloading Decision Optimization

Input: $offload[]$, $counter[]$, $route_r$, src, dst, k
Output: Updated $offload[]$

1 $flag$=1 ;
2 **while** $flag == 1$ *&& off_count* $== m$ **do**
3 $flag$ = 0 ;
4 **for** *each subtask* $i \in \mathcal{V}_T$ **do**
5 **if** $i > 1$ *&&* $i < m$ **then**
6 $v_{src} = offload[i - 1]$;
7 $v_{dst} = offload[i + 1]$;
8 **if** *i==1* **then**
9 $v_{src} = src, v_{dst} = offload[i + 1]$;
10 **if** *i==m* **then**
11 $v_{src} = offload[i - 1], v_{dst} = dst$;
12 Compute $min = \mathcal{U}_{offload[i]}$;
13 **for** *each* $j \in [v_{src}, v_{dst}]$ **do**
14 **if** $\pi_j > \Psi_i$ *&& counter*$[j] < k$ *&& src* $< dst$ **then**
15 $\alpha = S_i^I, \beta = S_i^O, w = w_i$;
16 Compute $\mathcal{U}_j$ using Equation (16) ;
17 **if** $\mathcal{U}_j < min$ **then**
18 $counter[offload[i]] - -$;
19 $offload[i] = v, flag = 1$;

When all the dependent subtasks can be offloaded to the nodes in a route, i.e., $off\_count = m$ in Algorithm 2, we optimize the offloading decision in terms of delay and energy consumption, using Algorithm 3. The main idea of optimizing offloading decision for a subtask i is to select a feasible node

j from the node-set $offload[i-1]$ to $offload[i+1]$ in the route. However, for the first subtask, we consider the nodes from the source to the $offload[i+1]$. Similarly, for the last subtask, the nodes between $offload[i-1]$ and destination are considered. The node which consumes less energy and delay for a subtask is considered as optimal.

Time complexity: In Algorithm 3, each subtask in Step 4 satisfies one condition – first, last, or an intermediate subtask. Therefore, Steps 5–11 take $O(1)$ time. The cost computation for $offload[i]$ takes constant time. In Steps 13–19, we compare the cost for each node and find the optimal one. In worst case, all the nodes are traversed and time complexity for this purpose is $O(n_r)$. For m such subtasks, the complexity is $O(mn_r)$. Further, there can be many iterations of optimizations in Step 2 and in the worst case, the time complexity is $O(mn_r)^2$. For a task with all dependent subtasks, Algorithm 2 and 3 together take $O(mn_r^2 + (mn_r)^2) \approx O(mn)^2$.

C. Offloading of Partially Dependent Tasks

A partially dependent task has both dependent subtasks and independent subtasks. If we remove all the independent subtasks from a partially dependent task, the resultant graph becomes a collection of some fully dependent sets of subtasks. As a fully dependent subtask with a precedence order is NP-hard to solve, finding optimal offloading decision for the partially dependent task is also NP-hard.

Algorithm 4: Offloading of Partially Dependent Tasks

```
   Input: ind[]
   Output: ind_set, dep_set, offload[]
1  Initialize i = 1, count = 0 ;
                ▷ count represents number of fully dependent tasks
2  while (1) do
3      if ind[i] == 1 && ind[i+1] == 1 then
4          ind_set = ind_set ∪ {i}, i++ ;
5      if ind[i] == 1 && ind[i+1] == 0 then
6          count++ ;
7          dep_set(count) = dep_set(count) ∪ {i} ; i++;
8          while (ind[i]) do
9              dep_set(count) = dep_set(count) ∪ {i} ;
10             i++ ;
11 Use Algorithm 1 for ind_set ;
12 for each dep_set do
13     Use Algorithm 2 followed by Algorithm 3 ;
```

All the steps for offloading a partially dependent task are outlined in Algorithm 4. The while loop starting in Step 2 decomposes the task into an independent task set and some fully dependent task sets. Then, we use the Algorithms of fully independent task and fully dependent task for handling each task set.

Time complexity: The while loop in Steps 2–10 takes $O(m)$ time for m number of subtasks. Now, considering m_x number of subtasks in the ind_set, Step 11 takes $O(m_x n)$ time. Then, in Step 12, if we consider $count = y$, i.e., y number of fully dependent tasks, the time complexity for Steps 11–13 is $O((m_1 n)^2 + (m_2 n)^2 + ... + (m_y n)^2) \approx O(mn)^2$. Finally, in total, Algorithm 4 takes $O(n + mn + (mn)^2) \approx O(mn)^2$ time.

V. Performance Evaluation

A. Simulation Settings

We evaluate the performance of the proposed algorithms through MATLAB simulation. Various parameters considered for the experiments are given in Table III. We considered roads with intersection points in an area of $5000 \times 5000m^2$, where the RSUs are deployed optimally to cover the on-road moving vehicles. We use the indirect trust assessment approach proposed in our previous work and successful task completion rate to assess the trustworthiness of the RSUs in the scale [0, 1] [24]. We use the same scale and randomly define the trust level requirement of the subtasks [9]. Our proposed algorithms will still work in applications with no trust level requirements by setting the trust requirement as zero. A vehicle is randomly selected that offloads the task to an RSU in its territory, considered the source. We adopt the Markov-predictor method proposed in work [16] to predict the service destination. We consider varying speed and random movement of the vehicles on the roads and therefore, use the Gauss-Markov mobility model [25] to generate the mobility pattern of the vehicles.

TABLE III: Simulation Parameters

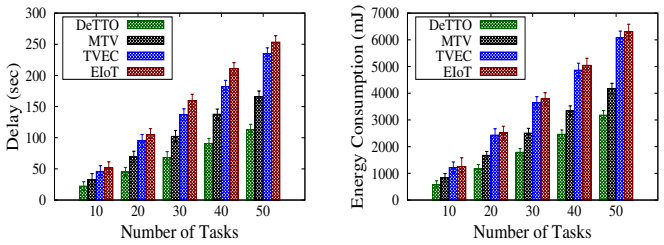
Parameter	Value
Number of RSUs	80
Number of vehicles	200
RSU CPU frequency	2–4 GHz [7]
Transmit power of RSU	23 dBm ≈ 0.2 W [21]
Number of tasks offloaded to an RSU	10-50
Number of subtasks per each task	5–10
Trust values of RSU and subtasks	[0, 1] [8]
Input and output size of data subtasks	10-50 KB, 50-200 KB [1]
Input and output size of processing subtasks	50–200 KB, 10–50 KB
Computational amount for subtasks	200–500 Megacycles

B. Benchmark Schemes

To show the effectiveness of our proposed algorithms, we consider some recently proposed task offloading methods with similar task dependency structures. For fully independent tasks, we consider the benchmark schemes that consider independent tasks, such as task offloading in vehicular edge computing networks (TVEC) [1], mobility-aware task offloading for vehicular network (MTV) [16], and energy-aware task offloading in IoT (EIoT) [6]. All these benchmark schemes consider independent tasks, and hence, suitable for the performance evaluation of the algorithm proposed for fully independent tasks. For finding offloading decisions for fully dependent tasks, which is NP-hard in nature, we select two existing heuristic schemes that consider interdependent tasks – dependency-aware task scheduling in vehicular edge computing (DTSE) [13] and task offloading with dependency in edge networks (TDEN) [14]. Finally, due to the unavailability of suitable existing schemes for tasks with partial dependency, we choose the best performing independent task offloading scheme, i.e., MTV, along with two dependency aware offloading schemes – DTSE and TDEN.

C. Results and Discussion

1) Fully Independent Task: All the subtasks of a fully independent task can be offloaded independently to the available RSUs. In Figure 3, the avergae task processing delay and avergae energy consumption of the proposed scheme are compared with the existing schemes – MTV, TVEC, and EIoT.



(a) Delay vs. tasks (b) Energy consumption vs. tasks

Fig. 3: Delay and energy consumption of fully independent tasks

i) Delay: The task processing delay of fully independent offloaded tasks is shown in Figure 3(a). We measure the task processing delay as the sum of all the subtask delays associated with a task. In DeTTO, we consider the cost function of each node, which is a function of distance and hop count from the source and destination, type of the subtasks, and the processing capability of the node. Therefore, in DeTTO, the offloading cost helps in finding the efficient offloading decision for each subtask. On the other hand, the schemes TVEC and EIoT only consider the distance of the nodes from the source. Due to this reason, the transmission delay can not be controlled for the large processing tasks. Further, in both schemes, the service destination is not considered, which increases the propagation delay. The MTV scheme considers the hop count and the service destination, due to which, MTV possesses reduced delay compared to TVEC and EIoT. With the increase in the number of tasks, the closer nodes to the source reach the maximum offloading limit, and hence, the subtasks are offloaded to distant nodes, which in turn yields an increased task processing delay. The benchmark schemes do not consider the tasks types and therefore, incur higher delay for data-oriented tasks as shown in 4, while compared to the processing type of subtasks.

ii) Energy consumption: The energy consumption mainly depends on transmission energy, which is badly affected, if the size of the task is not considered. However, it is difficult to predict what will be the size of a subtask after its execution. The cost function in DeTTO considers the type of each subtask and prepares a skeleton using the values α and β. All the compared schemes only consider the task size before execution and optimize the energy consumption until the offloaded node, leaving the rest of the path to the destination unconsidered. From figure 3(b), it is evident that DeTTO performs better, compared to other benchmark schemes. The MTV scheme considers hop count and the shortest path to the destination from the offloaded node, therefore, shows better results compared to TVEC and EIoT. From Figure 4, we observe that the energy consumption is high for all the benchmark schemes, when data-oriented tasks are considered. This is because data tasks are being offloaded to the nodes near to the source and possess high transmission energy.

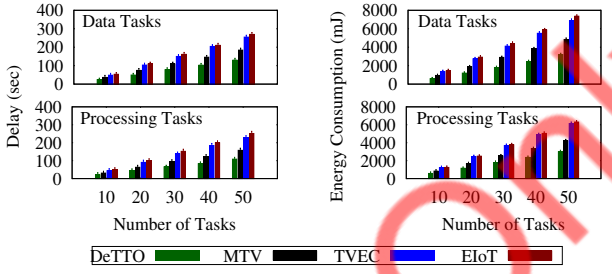
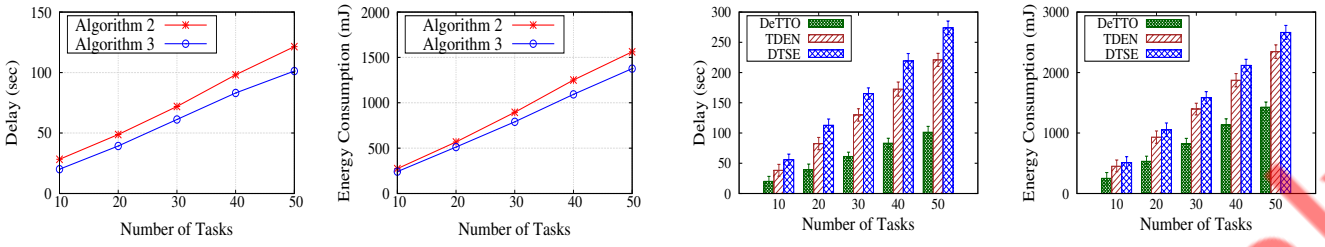


Fig. 4: Delay and energy consumption in different types of subtasks

2) Fully Dependent Tasks: DeTTO follows a two-step heuristic approach (Algorithm 2 and Algorithm 3) to find out the nodes that satisfy the trust requirements and connected to the nodes that are offloaded with the interdependent subtasks. Figure 5(a) and 5(b) show the delay and energy consumption in the first step (Algorithm 2) and the subsequent optimization after the second step (Algorithm 3).

i) Delay: In both the considered benchmark schemes shown in Figure 5(c), the high prioritized tasks are preferred over the low prioritized tasks. During the comparison, we consider the dependency task graph and prioritize the subtask with a lower index value. The DTSE approach considers the delay till a subtask is scheduled at an edge server. However, the execution of a dependent subtask depends on the node's distance, where the last subtask was executed and the size of the last executed subtask. Due to this reason, the DTSE approach possesses higher propagation and transmission delay. The TDEN heuristic approach considers the transmission delay. However, the executed task size can not be predicted apriori and hence, suffers from higher delay, when the output task size varies. DeTTO always tries to offload the data-oriented subtasks towards the destination, whereas the processing subtasks are offloaded as near as possible to the source. From Figure 5(a), it can be seen that the propagation delay is optimized by choosing the shortest route and the transmission delay is optimized in the second step, while aiming to reduce the transmission delay.

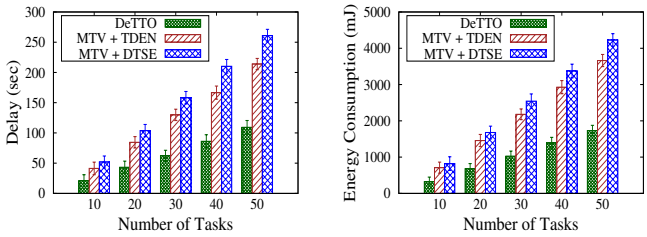
ii) Energy consumption: In a fully dependent task, the primary energy consumption factor is the energy consumed during the delivery of the executed result of a subtask to a dependent subtask. DeTTO optimizes both the transmission and computational energy using Algorithm 3, where the RSU (located in the route to the destination) with the lowest offloading energy cost is selected for each subtask. DTSE suffers from high energy consumption due to the fact that the transmission delay is not considered between two RSUs that execute two subsequent subtasks. On the other hand, the TDEN approach considers the transmission energy only for processing tasks and therefore, performs slightly better compared to the DTSE approach, as shown in Figure 5(d).



(a) Delay optimization (b) Energy optimization (c) Delay vs. tasks (d) Energy consumption vs. tasks

Fig. 5: (a) Delay and (b) energy optimization in DeTTO for fully dependent tasks; (c) delay and (d) energy consumption of different dependency-aware task offloading schemes.

3) Partially Dependent Tasks: To find optimal offloading decisions for the partially dependent tasks, we use the solution approaches of both fully independent and fully dependent tasks.



(a) Delay vs. tasks (b) Energy consumption vs. tasks

Fig. 6: Delay and energy consumption of partially dependent tasks

i) Delay: In a partially dependent task, the delay of the independent subtasks is comparatively high compared to that of fully dependent tasks. This happens due to the fact that each independent task output is delivered at the destination, whereas a dependent subtask delivers the output of a subtask to the node executing the subsequent subtask. It is worth mentioning that when the number of tasks increases, the number of independent subtasks increases, leading to increased delay in all the schemes. The offloading scheme in the DTSE approach does not consider the inter-RSU propagation delay, leading to the highest delay in the MTV+DTSE scheme. A better performance in MTV+DTEN is realized due to the consideration of the transmission delay between the RSUs. From Figure 6(a), we can observe that in DeTTO, the delay is very less compared to the other two schemes.

ii) Energy consumption: The DTSE approach primarily suffers from the propagation delay. Due to fact that energy consumption is effected by transmission and computational delay, the MTV+DTSE does not suffer much from energy consumption in Figure 6(b), while compared to the delay plot in Figure 6(a). On the other hand, the TDEN scheme considers the inter-RSU transmission delay, and hence, MTV+TDEN performs better than the MTV+DTSE approach. In DeTTO, the offloading decisions consider both transmission and computational energy. The degraded performance in all the schemes with respect to the increased number of tasks is noticed due to the existence of more number of independent subtasks.

iii) *Trust score of executed task:* The trust score of an executed task represents the sum of all the trust scores of the RSUs that executed the subtasks cooperatively. The trust score of each RSU is represented in between [0,1] and therefore, a task with m subtasks can have a maximum trust score of m.

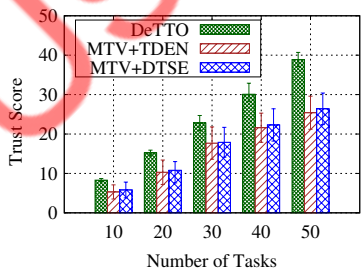


Fig. 7: Trust score of task after execution

In Figure 7, we can observe that the trust score of DeTTO is highest compared to the other schemes. DeTTO only tries to satisfy the trust requirements of the subtasks and minimizes the delay and energy consumption. Therefore, the trust score attained at an RSU may not be high if the trust requirement is low. In the other two schemes, the trust requirement is not considered. Due to this reason, for the benchmark schemes, the trust score attained at the RSUs are very random. Further, the trust scores are very close and follow a mostly similar trend in the benchmark schemes.

VI. Conclusion

In this paper, we studied the task offloading problem in a distributed computing-enabled vehicular network, where large computation-intensive tasks offloaded from the vehicles are fragmented into subtasks and executed in multiple RSUs. We considered three types of dependency structures between the subtasks. For fully independent tasks, a greedy approach is proposed. Next, to find offloading decisions for the fully dependent tasks with NP-hardness complexity, we proposed a two-fold heuristic approach. We showed the minimized delay and energy consumption achieved through the optimized steps of the solution approach. For partial dependent tasks, we used the mixed solution approach proposed for the first two types of tasks. Extensive simulation results are presented to show the effectiveness of the proposed method, DeTTO.

In future work, we plan to consider inter-vehicle task offloading, while considering the vehicle trustworthiness, computation abilities, and their mobility patterns. In addition, we plan to extend the solutions to a real-world scenario and incorporate the interference and the vehicle traffic at the RSUs.

REFERENCES

[1] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2092–2104, 2019.

[2] D. Kim, Y. Velasco, W. Wang, R. Uma, R. Hussain, and S. Lee, "A new comprehensive RSU installation strategy for cost-efficient VANET deployment," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 5, pp. 4200–4211, 2016.

[3] K. Kim, J. Lynskey, S. Kang, and C. S. Hong, "Prediction based sub-task offloading in mobile edge computing," in *2019 International Conference on Information Networking (ICOIN)*. IEEE, 2019, pp. 448–452.

[4] W. Zhan, C. Luo, G. Min, C. Wang, Q. Zhu, and H. Duan, "Mobility-aware multi-user offloading optimization for mobile edge computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3341–3356, 2020.

[5] Y. Jiang and D. H. Tsang, "Delay-aware task offloading in shared fog networks," *IEEE Internet of Things Journal*, vol. 5, no. 6, pp. 4945–4956, 2018.

[6] J. Li, M. Dai, and Z. Su, "Energy-aware task offloading in the internet of things," *IEEE Wireless Communications*, vol. 27, no. 5, pp. 112–117, 2020.

[7] H. Guo, J. Liu, J. Ren, and Y. Zhang, "Intelligent task offloading in vehicular edge computing networks," *IEEE Wireless Communications*, vol. 27, no. 4, pp. 126–132, 2020.

[8] Y. Ouyang, W. Liu, Q. Yang, X. Mao, and F. Li, "Trust based task offloading scheme in uav-enhanced edge computing network," *Peer-to-Peer Networking and Applications*, pp. 1–23, 2021.

[9] N. Torkzaban and J. S. Baras, "Trust-aware service function chain embedding: A path-based approach," in *2020 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2020, pp. 31–36.

[10] H. Hu, R. Lu, C. Huang, and Z. Zhang, "Ptrs: A privacy-preserving trust-based relay selection scheme in vanets," *Peer-to-Peer Networking and Applications*, vol. 10, no. 5, pp. 1204–1218, 2017.

[11] A. Nandan, S. Das, G. Pau, M. Gerla, and M. Sanadidi, "Co-operative downloading in vehicular ad-hoc wireless networks," in *Second Annual Conference on Wireless On-demand Network Systems and Services*. IEEE, 2005, pp. 32–41.

[12] L.-M. Ang, K. P. Seng, G. K. Ijemaru, and A. M. Zungeru, "Deployment of iov for smart cities: applications, architecture, and challenges," *IEEE access*, vol. 7, pp. 6473–6492, 2018.

[13] Y. Liu, S. Wang, Q. Zhao, S. Du, A. Zhou, X. Ma, and F. Yang, "Dependency-aware task scheduling in vehicular edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 4961–4971, 2020.

[14] Y. Han, Z. Zhao, J. Mo, C. Shu, and G. Min, "Efficient task offloading with dependency guarantees in ultra-dense edge networks," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[15] D. Wu, G. Shen, Z. Huang, Y. Cao, and T. Du, "A trust-aware task offloading framework in mobile edge computing," *IEEE Access*, vol. 7, pp. 150 105–150 119, 2019.

[16] S. Misra and S. Bera, "Soft-VAN: Mobility-aware task offloading in software-defined vehicular network," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2071–2078, 2019.

[17] N. J. Goodall, "Real-time prediction of vehicle locations in a connected vehicle environment," Tech. Rep., 2013.

[18] A. Monreale, F. Pinelli, R. Trasarti, and F. Giannotti, "Wherenext: a location predictor on trajectory pattern mining," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 637–646.

[19] Q. Liu, S. Wu, L. Wang, and T. Tan, "Predicting the next location: A recurrent model with spatial and temporal contexts," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[20] S. Sigg, D. Gordon, G. Von Zengen, M. Beigl, S. Haseloff, and K. David, "Investigation of context prediction accuracy for different context abstraction levels," *IEEE Transactions on Mobile Computing*, vol. 11, no. 6, pp. 1047–1059, 2012.

[21] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, 2018.

[22] M. W. Convolbo and J. Chou, "Cost-aware DAG scheduling algorithms for minimizing execution cost on cloud resources," *The Journal of Supercomputing*, vol. 72, no. 3, pp. 985–1012, 2016.

[23] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Annals of discrete mathematics*. Elsevier, 1979, vol. 5, pp. 287–326.

[24] P. Dass, S. Misra, and C. Roy, "T-safe: Trustworthy service provisioning for iot-based intelligent transport systems," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 9, pp. 9509–9517, 2020.

[25] T. Camp, J. Boleng, and V. Davies, "A survey of mobility models for ad hoc network research," *Wireless communications and mobile computing*, vol. 2, no. 5, pp. 483–502, 2002.