Privacy-by-Design Distributed Offloading for Vehicular Edge Computing

Weibin Ma
University of Delaware
Department of Computer and Information Sciences
Newark, DE, USA
weibinma@udel.edu

Lena Mashayekhy
University of Delaware
Department of Computer and Information Sciences
Newark, DE, USA
mlena@udel.edu

ABSTRACT

Vehicular Edge Computing (VEC) is a distributed computing paradigm that utilizes smart vehicles (SVs) as computational cloudlets (edge nodes) by virtue of their inherent attributes such as mobility, low operating costs, flexible deployment, and wireless communication ability. VEC extends edge computing services by expanding computing coverage and further improving quality-of-services (QoS) for devices. Due to limited onboard energy and computation capabilities of SV-mounted cloudlets, a single vehicle might not be able to execute a large number of tasks and guarantee their desired QoS. To address this problem, the overloaded vehicle can fulfill its overwhelming workload by offloading its tasks to other available connected vehicles. However, data privacy and accessibility are of critical importance that need to be considered for offloading. In this paper, we propose privacy-by-design offloading solutions for VEC to facilitate latency requirements of user demands and reduce energy consumption of vehicles. We formulate the Data pRotection Offloading Problem (DROP) as an Integer Program and prove its NP-hardness. To provide computationally tractable solutions, we propose three distributed algorithms by leveraging graph theory to solve this problem. We evaluate the performance of our proposed algorithms by extensive experiments and compare them to the optimal results obtained by IBM ILOG CPLEX. The results demonstrate the flexibility, scalability, and cost efficiency of our proposed algorithms in providing practical privacy-by-design offloading solutions enabling edge services along the cloud-to-thing continuum.

KEYWORDS

Vehicular Edge Computing, Smart Connected Vehicles, Internet-of-Things, Data Protection, Distributed Algorithms

ACM Reference Format:
Weibin Ma and Lena Mashayekhy. 2019. Privacy-by-Design Distributed Offloading for Vehicular Edge Computing. In *IEEE/ACM 12th International Conference on Utility and Cloud Computing (UCC'19), December 2–5, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3344341.3368804

UCC '19, December 2–5, 2019, Auckland, New Zealand

ACM ISBN 978-1-4503-6894-0/19/12…$15.00
https://doi.org/10.1145/3344341.3368804

1 INTRODUCTION

The ubiquitous penetration of smart connected devices (Internet of Things) into everyday life is projected to reach 50 billion by 2020 [23]. The growth of IoT will continue as users enjoy the convenience of mobility and with the emergence and progress of new technologies such as wearable devices, autonomous vehicles/drones, and collaborative augmented/virtual reality. To enable these IoT applications and scale over the number of participants and large geographical areas, computational capabilities of IoT devices are not sufficient due to being restricted by weight, size, battery life, and heat dissipation. To handle this challenge, offloading computation to clouds to remotely execute IoT applications is one of the promising solutions. As data proliferation increases exponentially, however, sending data from IoT devices to the cloud is not feasible for time-sensitive applications.

Edge Computing (EC) has been introduced as a new paradigm [21] that optimizes cloud computing systems to provide a distributed computing solution at the edge of the network, where IoT devices utilize the computing resources, called cloudlets, in their vicinity. Edge computing can be leveraged to bridge the gap between the increasing computational demand of IoT devices and their limited computational capabilities [24]. However, deploying cloudlet infrastructure at the edge of the network is costly and may not be feasible in many situations (e.g., disaster situations, emergency rescue, unexpected surge in user demand) and regions with sparse or no infrastructure of wireless access points such as remote rural areas [1]. Moreover, a single cloudlet has finite computing resources, which makes it hard to fulfill demand spikes (e.g., a massive number of offloading requests from IoT devices).

To overcome these problems, Vehicular Edge Computing (VEC) has recently been introduced as an emerging edge platform [4, 16, 17, 30], where smart vehicles (SVs) such as Unmanned Aerial Vehicles (UAVs) and Connected and Autonomous Vehicles (CAVs) are considered as computational cloudlets by virtue of their inherent attributes such as mobility, low operating costs, flexible deployment, and wireless communication ability. SV-mounted cloudlets can expand edge computing services and further improve quality-of-services (QoS) for IoT devices.

While VEC can bring many opportunities to avoid QoS violations and balance the load, these SV-mounted cloudlets have often limited computing capacities and energy budgets. To expand the computational capacity and coverage of a single SV-mounted cloudlet, especially when serving large computational demands, autonomous cooperation and coordination among multiple SVs are needed to form an interconnected computing system and improve

quality of edge services [11, 12, 22]. Such a perspective, on the other hand, opens new research challenges on account of the current lack of efficient data protection mechanisms for task offloading. In particular, the major categories are considered privacy and accessibility restrictions defined as follows:

***Privacy restrictions*:** If some tasks are offloaded to the same SV, they can reveal sensitive information, which could harm IoT user privacy [15, 18] (e.g., in finance and healthcare data).

***Accessibility restrictions*:** There are often some restrictions in offloading a task to a specific SV due to lack of trust, reliability, or system compatibility/preferences [14, 27] (e.g., cannot provide a proper type of VM for a task). In addition, there may exist some enforced policies that do not allow a third party to access confidential user data.

These restrictions eliminate the privacy-breach problem and enable *privacy-by-design* solutions. Many studies have been conducted to protect outsourced data by designing steganography and encryption [9]. However, these algorithms have limitations, especially in edge computing domain, due to requiring additional processing before offloading. Moreover, encryption may dramatically increase the amount of data. Privacy by design is a suitable approach for offloading in VEC, by incorporating privacy principles as early as in the design phase of systems–a proactive rather than reactive to risk.

In this paper, we consider a set of SV-mounted cloudlets cooperating to provide edge computing services for IoT devices. They are capable of both communicating with the devices and offloading computation to other SVs via the wireless communication technology in order to fulfill the overwhelming demand. We focus on the critical problem of offloading tasks from an overloaded SV to minimize computation overhead in terms of energy consumption and processing time while satisfying *privacy* and *accessibility restrictions*. We first design an optimal offloading mathematical model for this Data pRotection Offloading Problem (DROP), and we theoretically prove that this is an NP-hard problem. We then propose three distributed algorithms by virtue of graph theory to obtain efficient and computationally tractable solutions in order to minimize the total computation overhead of the offloading. We finally evaluate the performance of our proposed solutions in extensive experiments. To the best of our knowledge, this is *the first work that provides privacy-by-design offloading solutions among a set of cooperative SVs*.

Organization. The rest of the paper is organized as follows. In Section 2, we discuss the state-of-the-art research in this domain. In Section 3, we introduce the problem of optimal task offloading among cooperative SVs considering the privacy and accessibility restrictions, and we mathematically formulate the problem. In Section 4, we present our proposed computationally tractable algorithms. In Section 5, we evaluate our proposed algorithms by extensive experiments. In Section 6, we summarize our results and present possible directions for future research.

2 RELATED WORK

The key challenges of DROP lie in the combinatorial nature of offloading decisions, the necessity of data protection requirements, and the limited capacities of SVs. Due to the dynamics and unplanned deployment of SV-mounted cloudlets, centralized optimization approaches for task offloading may not be efficient in such a distributed environment. Moreover, they require each IoT user to report his/her own information including the capacity of the IoT device and the size of each task to a centralized entity, e.g., cloud, which decides the offloading decisions accordingly. Therefore, extra concerns such as high computational complexity, tremendous communication overhead, and massive data transfers, are unavoidable in centralized optimization. All these points accelerate the emergence of efficient offloading schemes by designing decentralized approaches.

Recently, a few studies [12, 13, 16], devised distributed approaches to optimize the offloading problem in EC by exploiting multi-player noncooperative games. Messous *et al*. [16] presented a game-theoretic approach to address the intensive computation offloading problem for multiple UAVs, and they showed that the formulated game admits a Nash equilibrium. Ma *et al*. [12] investigated the feasibility of offloading computational tasks in a network of capacitated UAV-mounted cloudlets in order to minimize the energy consumption of UAVs while satisfying QoS requirements of the tasks. They devised an efficient decentralized approach based on potential games. Ma *et al*. [13] proposed a distributed computation offloading approach for multiple users to offload their tasks to a single cloudlet via multiple access points. However, the game theoretic approaches present some drawbacks [5]. For example, finding the best response of each player usually requires the knowledge of other players' actions at each round, which instead increase the computational complexity of the solution. Furthermore, to find an equilibrium of a game, the objective function requires a specific structure.

With the rapid proliferation of smart vehicles, the task offloading problem in VEC has received considerable attention. Zhang *et al*. [29] proposed a contract-based offloading and computation resource allocation scheme in VEC. Zhang *et al*. in [28] studied the feasibility of combining vehicular cloudlets with the centralized cloud and proposed a flexible offloading strategy to explore underutilized resources via task migration. Yu *et al*. in [26] proposed a coalitional game model for cooperation among cloud service providers in cloud-enabled vehicular networks in order to share and utilize idle resources.

Nevertheless, none of the existing work investigated both data privacy and cloudlets accessibility in a multi-cloudlet cooperative computing system.

3 PRIVACY-BY-DESIGN OPTIMIZATION FORMULATION

In this section, we introduce the system model and the data protection offloading problem.

3.1 System Model

In this subsection, we describe the system model with a set of SVs acting as mobile computational cloudlets that provide edge computing services to IoT devices. We consider that an overloaded SV has to offload its overwhelming computation to other SVs in order to fulfill its demands and guarantee the desired QoS of its workload.

The objective is to minimize the overhead cost of offloading while satisfying the aforementioned *privacy* and *accessibility restrictions*. This is an offloading decision-making problem that requires an efficient and proactive data protection offloading solution.

We consider u_0 as the overloaded SV and $U = \{u_1, \ldots, u_m\}$ as the set of m SVs with available computational resources. The set of tasks at SV u_0 to be offloaded is denoted by $K = \{k_1, \ldots, k_n\}$, where $n \geq 1$ is the number of tasks (u_0 has at least one task to be offloaded). Let F_i be the CPU frequency (i.e., CPU cycles per second) of u_i, and e_i denotes its energy consumption per CPU cycle. The capacity of u_i is denoted by $u_i^{cap} = (C_i^{cap}, D_i^{cap})$, where C_i^{cap} is the number of idle computational cycles of u_i and D_i^{cap} is the available memory size of u_i. Each task $k_i \in K$ has some requirements defined by (C_i, D_i), where C_i represents the number of computational cycles required to obtain the outcome of k_i and D_i denotes its data size. Similar to previous studies in mobile wireless networks [2, 7], for tractability, we assume that the locations of SVs remain fixed during the offloading decision making. More details are presented in the following subsections.

SV u_0 does not have enough computational resources to complete all its incoming tasks with their required QoS, and hence this overloaded SV has to offload these tasks to other SVs. As we described in the Introduction Section, there are two restrictions to be considered when offloading tasks to other SVs: (i) the *privacy restrictions*, specifying that some tasks cannot be offloaded to the same SV; (ii) the *accessibility restrictions*, specifying that a task cannot be offloaded to a specific SV. To model these two restrictions, we visualize them in the form of a *conflict graph* and *searching matrix*, respectively.

We model the *privacy restrictions* using a conflict graph $\mathcal{G}\langle \mathbf{V}, \mathbf{E} \rangle$, where $\mathbf{V} = \{v_1, \ldots, v_n\}$ is a set of vertices representing the tasks in K [1] and $\mathbf{E}$ is a set of edges $\langle i, j \rangle$ representing a conflict between task k_i and k_j, where $i, j \in \{1, \ldots, n\}$. Specifically, if tasks k_i and k_j cannot be offloaded to the same SV, an edge $\langle i, j \rangle$ is added between v_i and v_j in the graph. If there is no edge between two vertices, their corresponding tasks can be offloaded to the same SV. Therefore, $\mathcal{G}$ is an undirected graph, and $\mathbf{E}$ is symmetric.

[1] We use the terms task and vertex interchangeably.

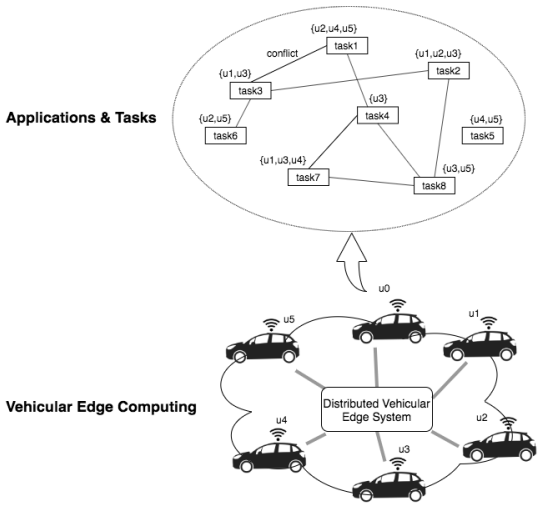


Figure 1: VEC architecture: an overloaded SV u_0 offloads its tasks to other SVs

We model the *accessibility restrictions* by a $m \times n$ matrix $\mathcal{T}_{m,n} = (t_{ij})$, where the ith row represents SV $u_i \in U$ and the jth column represents task $k_j \in K$. In particular, if SV u_i is accessible to task k_j, we set $t_{ij} = 1$; otherwise, $t_{ij} = 0$.

Fig 1 shows an overloaded SV offloads its overwhelming tasks to other SVs with enough computational resources via vehicle-to-vehicle (V2V) communications, while considering the data protection restrictions.

3.2 Data Protection Offloading Problem

In this subsection, we introduce the optimal mathematical model for the data protection offloading problem. We first describe the communication, energy consumption, and latency models, and we then present our optimization model.

1) Communication Model.
To avoid generating severe interference during offloading tasks, we consider the overloaded SV applies a Frequency-Division Multiple Access scheme (FDMA) to transmit tasks to other SVs. In FDMA, each channel between u_0 and $u_j \in U$ is unique, and u_0 can hence offload tasks to other SVs simutaneously without co-channel interference. As a result, the transmission rate of u_0 to u_j is defined as [20]:

$$R_{0j} = B_j \log(1 + \frac{P_0}{N_0 d_{0j}^h}),$$

where B_j is the total bandwidth of u_j, P_0 denotes the transmission power of u_0, and d_{0j} is the distance between u_0 and u_j. In addition, N_0 and h represent the background noise and the path-loss factor, respectively.

2) Energy Consumption Model.
The total energy consumption for completing a task via offloading (e.g., from u_0 to u_j, where $u_j \in U$) consists of three components: the transmission energy consumption from u_0 to u_j, the execution energy consumption at u_j, and the backhaul energy consumption of the outcomes of computation from u_j to u_0. Similar to many studie [2, 6, 12], the backhaul energy consumption can be ignored, since the size of output is generally much smaller than the size of input. As a result, the total energy consumption for completing task k_i via offloading from u_0 to u_j is calculated as:

$$E_j(i) = \frac{D_i P_0}{R_{0j}} + C_i e_j. \tag{1}$$

The first term represents the transmission energy, and the second term represents the energy consumption of executing the task at u_j.

3) Latency Model.
Likewise, the total delay of completing task k_i by offloading (from u_0 to u_j) is calculated as:

$$T_j(i) = \frac{D_i}{R_{0j}} + \frac{C_i}{F_j}, \tag{2}$$

where the first term represents the total transmission time of k_i from u_0 to u_j, and the second term represents the total execution time of k_i at u_j.

4) Optimization Model.
Both latency and energy consumption are important factors in offloading among SVs. One of the main limitations of SVs is their restricted battery lifetime. On the other hand, most IoT applications are sensitive to delay such as video streaming and real-time games. We hence consider an overhead cost as a combination of both latency and energy consumption. According to (1) and (2), the overhead cost of offloading task k_i to u_j in terms of the energy and latency is calculated by:

$$Z_j(i) = \alpha_i E_j(i) + \beta_i T_j(i), \tag{3}$$

where $0 \leq \alpha_i, \beta_i \leq 1$ represent the relative weights of energy consumption and latency of the objectives, and $\alpha_i + \beta_i = 1$. This provides rich modeling flexibility to prioritize the objectives (energy and latency) according to the required QoS of the applications and the current battery status of the SVs. For instance, when a task is delay-sensitive and the SVs are at a high-battery state, more weight is assigned to the latency (i.e., higher β_i). In practice, the proper weights can be determined by exploiting multiple criteria decision making (MCDM) and multi-attribute utility theory (MAUT) approaches [25].

We define an indicator variable $b_{ij}, \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}$, that characterizes the accessibility relationship between task k_i and SV u_j as follows:

$$b_{ij} = \begin{cases} 1 & \text{if task } k_i \text{ can be offloaded to } u_j, \\ 0 & \text{otherwise.} \end{cases}$$

This indicates whether SV u_j is accessible to task k_i or not. We then define a decision variable x_{ij} as follows:

$$x_{ij} = \begin{cases} 1 & \text{if task } k_i \text{ is offloaded to } u_j, \\ 0 & \text{otherwise.} \end{cases}$$

This specifies whether task k_i is offloaded to SV u_j or not. We now formulate the data protection offloading problem (DROP) as an Integer Program (IP), called IP-DROP, as follows:

$$\text{Minimize } \mathcal{Z} = \sum_{i=1}^{n} \sum_{j=1}^{m} Z_j(i) x_{ij} \tag{4}$$

Subject to:

$$\sum_{i=1}^{n} C_i x_{ij} \leq C_j^{cap}, \qquad \forall j \in \{1, \ldots, m\}, \tag{5}$$

$$\sum_{i=1}^{n} D_i x_{ij} \leq D_j^{cap}, \qquad \forall j \in \{1, \ldots, m\}, \tag{6}$$

$$\sum_{j=1}^{m} b_{ij} x_{ij} = 1, \qquad \forall i \in \{1, \ldots, n\}, \tag{7}$$

$$x_{vj} + x_{v'j} \leq 1, \qquad \forall \langle v, v' \rangle \in \mathbf{E}, \forall j \in \{1, \ldots, m\}, \tag{8}$$

$$x_{ij} \in \{0, 1\}, \qquad \forall i \in \{1, \ldots, n\}, \forall j \in \{1, \ldots, m\}. \tag{9}$$

The objective function (4) is to minimize the overhead cost of offloading all tasks in K. Constraints (5)-(6) ensure that the assignment of the offloaded tasks to each SV does not exceed the available computational cycles and memory size of that SV, respectively. Constraints (7) ensure that each task is offloaded to exactly one SV obeying the *accessibility restrictions*. Constraints (8) ensure the *privacy restrictions* such that any two conflicted tasks cannot be offloaded to the same SV. Finally, Constraints (9) guarantee that the decision variables are binary.

3.3 Computational Complexity of DROP

To verify the computational complexity of DROP, we first introduce the related preliminaries from graph theory.

Definition 1. Given a graph $G(V, E)$ and a set of colors, a *proper coloring* is an assignment of colors to vertices such that no two adjacent vertices have the same color.

Definition 2. Given a graph $G(V, E)$ and a list of colors $L(v)$ for $v \in V$, a *proper list coloring* is a choice function $c(\cdot)$ that maps every vertex $v \in V$ to a color in the list $L(v)$ such that $c(v) \in L(v)$ and $c(i) \neq c(j)$ if $\langle i, j \rangle \in E$ for all $i, j \in V$.

We now define DROP as a *list coloring problem* (LCP) considering $\mathcal{G}(\mathbf{V}, \mathbf{E})$ and $\mathcal{T}_{m,n}$. Each task $v \in \mathbf{V}$, represented by a vertex, is given a *list* of accessible SVs (permissible colors) [2] based on $\mathcal{T}_{m,n}$. For our analysis, we let each task $v \in \mathbf{V}$ have a list of permissible SVs $L(v) \subseteq \{u_1, u_2, \ldots, u_m\}$, where m is the total number of SVs. We then sort SVs in $L(v)$ based on their corresponding overhead cost in ascending order. We define the sorted list of permissible SVs of task v as $L^s(v)$ and its corresponding sorted overhead cost list is defined as $Z^s(v)$.

Lemma 1. *If all tasks have the same requirements (computation C_i and data size D_i, for all tasks $k_i \in K$) and considering consistent priorities for the objectives (α_i, β_i remains the same), then the offloading cost $Z_j(i)$ to any u_j is the same for all tasks $k_i \in K$.*

Proof. Since the offloading scenario in DROP is that an overloaded SV u_0 offloads n tasks $K = \{k_1, \ldots, k_n\}$ via FDMA scheme to m SVs, the parameters $P_0, N_0, h, \gamma, B_j, F_j, d_{0j}$ with respect to any SV u_j are hence constant. Therefore, if any two tasks $k_u, k_w \in K$ have the same requirements $C_u = C_w, D_u = D_w$ while $\alpha_u = \alpha_w, \beta_u = \beta_w$, according to Eq. (1-3), for these tasks we have $Z_j(u) = Z_j(w), \forall j \in \{1, \ldots, m\}$. □

In graph theory, a *sum coloring* of a graph is a labeling of its vertices by natural numbers (positive integers), with no two adjacent vertices having equal labels, that minimizes the sum of the labels. The minimum sum that can be achieved is called the *chromatic sum* of the graph.

Definition 3. The *chromatic sum* $\sum G$ of graph G is the smallest sum of labels (colors) among all proper colorings with natural numbers.

The chromatic sum problem is NP-complete [10]. We now prove the computational complexity of DROP. The following lemma constructs the decision version of DROP by imposing a bound on the cost value and proves its NP-completeness.

[2] From now, we use the terms SV and color interchangeably.

Lemma 2. The decision version of DROP, called D-DROP, is *NP-complete*: Given $\mathcal{G}\langle \mathbf{V}, \mathbf{E}\rangle$, $\mathcal{T}_{m,n}$, and a value F, is there a *complete* assignment such that the overhead cost of all tasks, called $\mathcal{Z}$, does not exceed F (i.e., $\mathcal{Z} \leq F$)?

Proof. The first step is to prove that D-DROP is in NP by showing that given a complete assignment $\mathcal{X}$, it can be decided in polynomial time that $\mathcal{X}$ is a solution to the problem or not. This is easy to show since a nondeterministic program guesses a complete assignment $\mathcal{X}$ for all vertices and checks it in polynomial time whether this assignment is *feasible* and $\mathcal{Z} \leq F$.

The second step is to find a polynomial-time reduction from the decision version of the *chromatic sum problem* (denoted here by CSP-D), a well known NP-complete problem [10], to D-DROP. The CSP-D problem is defined as follows: Given graph $G(V, E)$ and an integer F, is there a *feasible* coloring c of graph G such that $\sum_{v \in V} c(v) \leq F$? We construct an instance of D-DROP as follows: We use a one-to-one mapping between $\mathcal{G}(\mathbf{V}, \mathbf{E})$ and $G(V, E)$, and considering F. We let each task $v \in V$ have a list of permissible SVs $L(v) = \{u_1, u_2, \ldots, u_m\}$, where m is the total number of SVs. Following Lemma 1, we assume all tasks $k_i \in K$ have the same required computational cycles and data size for simplicity. Therefore, all tasks have the same $L^s(v)$ and $Z^s(v)$. The Yes/No answer to the D-DROP instance corresponds to the same answer as for the CSP-D instance. Is there a *feasible* complete assignment $\mathcal{L} = \{l(v) | l(v) \in L(v)\}$ such that $\sum_{v \in V} Z_{l(v)}(v) \leq F$, where $Z_{l(v)}(v) \in Z^s(v)$ is the overhead cost that task v is offloaded to its permissible SV $l(v)$? Obviously, the new constructed problem is equivalent to the CSP-D, and this construction is done in polynomial time. Therefore, we have the CSP-D $\leq_P$ D-DROP. In other words, we can transform the decision version of the *chromatic sum problem* to a special case of D-DROP in polynomial time.

Therefore, the D-DROP is NP-complete. □

The decision version of a problem is easier than (or the same as) the optimization version. We now prove the computational complexity of the optimization version of DROP.

Theorem 1. *The optimization version of DROP (O-DROP) for finding the minimum overhead cost of offloading all tasks is NP-hard.*

Proof. The proof is by contradiction. If a polynomial-time algorithm can be found to solve O-DROP, it implies that we can obtain $\mathcal{Z}$ in polynomial time. Then, we just need to check if there is a solution for D-DROP considering $\mathcal{Z}$ as F. Obviously, this comparison is done in polynomial time, and thus, the D-DROP can be solved in polynomial time, which contradicts Lemma 2. Since the D-DROP is NP-complete, then the O-DROP is NP-hard (D-DROP $\leq_P$ O-DROP).

□

4 DISTRIBUTED SOLUTIONS

In this section, we propose three algorithms for solving DROP. The reason we prefer distributed algorithms is due to their robustness and scalability. Since DROP is NP-hard, optimal solutions may only be obtained when the problem size is relatively small (a few vehicles and tasks). We design proper distributed offloading algorithms that are computationally tractable in finding complete assignments with small overhead cost for privacy-by-design task offloading.

Algorithm 1 *DIST-RAND-DROP*

```
S ← ∅        /*offloading decisions*/
repeat
    s_min ← min_{k_i ∈ K} |L(k_i)|
    V_min ← {k_i ∈ K, |L(k_i)| = s_min}
    each task k_i ∈ V_min receives a unique random number r_i in [1, |V_min|]
    if k_i has the highest r_i among its neighbors in G then
        k_i selects SV u* with minimum overhead cost
        S(k_i) ← u*
        k_i multicasts (k_i, u*) to all its neighbors
    else
        task k_i ∈ V that is adjacent to k_j receives (k_j, u*)
        V = V \ {k_j}
        T(u*, k_i) ← NaN
until all tasks are assigned
Return S
```

4.1 Distributed Randomized Algorithm

In this subsection, we design an iterative randomized distributed algorithm called DIST-RAND-DROP in which the offloading will be based on a uniform distribution. In each round, tasks with the highest value among their neighbors (i.e., conflicting tasks) select their SVs for offloading. Then, each of these tasks, called a winning task, multicasts its offloading information to all its neighbors in which the selected SV will be removed from their permissible list. Next, these winning tasks are removed, and the remaining tasks that lose this contention update their own sets for the next round.

DIST-RAND-DROP, given in Algorithm 1, works as follows. It defines integer s_{min} to be the minimum number of permissible SVs for any task $k_i \in K$. Then, a list of tasks with minimum number of permissible SVs, s_{min}, is added to $\mathcal{V}_{min}$. Each task in $\mathcal{V}_{min}$ receives a unique random number uniformly from [1, $|\mathcal{V}_{min}|$] without any replacement. Within one round, if one task has the highest random number among all its neighbors, it wins the contention and selects its permissible SV with minimum overhead cost. Then each winning task multicasts offloading information to all its neighbors in which the selected SV will be removed from their permissible SVs lists (by setting it to NaN or not available). Next, these winning tasks are removed from $\mathbf{V}$ and the remaining tasks that lose the contention update their own $\mathcal{U}^{cap}, \mathcal{G}(\mathbf{V},\mathbf{E}), \mathcal{T}(m, n)$. The algorithm continues to the next round.

Example 1 (Infeasible assignment). This algorithm can result in infeasible assignments. Consider a privacy-restriction conflict graph shown in Fig. 2, where all tasks have the same size of permissible SVs list (here 2) and all SVs have adequate capacity. According to the property of DIST-RAND-DROP, a unique random number is assigned to each task (shown in green). Task 2 has the highest number among all its neighbors, thus it selects the first SV in its list (i.e., u_1) and multicasts it to other tasks to update their lists for the next rounds. This selection results in an infeasible assignment, since both tasks 3 and 4 have only SV u_3 for offloading

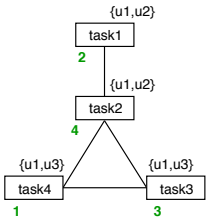


Figure 2: Infeasible assignment

while they conflict with each other and cannot be offloaded to the same SV.

This encourages us to design an efficient selection function at each round to improve the probability of finding a complete assignment.

4.2 Distributed Clique Algorithm

In this subsection, we introduce our second algorithm called DIST-CLIQUE-DROP, shown in Algorithm 2, that considers feasibility of assignments in cliques.

Given the conflict graph $\mathcal{G}(\mathbf{V}, \mathbf{E})$, accessibility matrix $\mathcal{T}_{m,n}$, and SVs capacity $\mathcal{U}^{cap}$, the algorithm first calculates s_{min} and $\mathcal{V}_{min}$. It then arbitrarily selects a task $k_i \in \mathcal{V}_{min}$, and sorts its permissible SVs in ascending order of their overhead cost, denoted by $L^s(k_i)$. Next, this task checks whether a selected SV from the list $L^s(k_i)$ leads to a feasible assignment. This step is done using cliques. First, all cliques containing any adjacent task $k_j \in \mathbf{V}$ to k_i, where $\langle i, j \rangle \in \mathbf{E}$ are found. If the number of unique SVs in each of these cliques is larger than or equal to the clique size itself, then there is a feasible assignment. Task k_i investigates if choosing the first SV from the list $L^s(k_i)$ (minimum cost) leads to a feasible assignment for all neighboring tasks. Otherwise, it chooses the next SV in the list and check the feasibility. This is a backtracking search approach, and it is halted when all the neighboring tasks are assigned to some SVs.

Task k_i sends its offloading information to all its neighbors to remove its selected SV from their permissible SVs list. Finally, this task is removed from $\mathbf{V}$ and the remaining tasks update their own $\mathcal{U}^{cap}$, $\mathcal{G}(\mathbf{V},\mathbf{E})$, $\mathcal{T}(m,n)$. The algorithm continues to next round iteratively. If none of the SVs are selected for k_i, that means the algorithm could not find a complete solution and it starts the next iteration with another task. Note that only one task is processed within one round through the algorithm.

Since there are multiple restrictions and constraints in DROP, including privacy and accessibility restrictions, and SVs capacities, we devise the next algorithm based on a *proactive renew rule* to enhance performance in finding efficient solutions.

4.3 Distributed Renew Algorithm

In this subsection, we present the distributed renew algorithm based on a *proactive renew rule* to increase the probability of finding a complete assignment.

The distributed renew algorithm, given in Algorithm 3, is called with $\mathcal{S} = \emptyset$. It first collects tasks with the minimum number of permissible SVs. If any of these tasks has a conflict with another task in the set, the task with the minimum degree is selected. In case of a tie, the task is selected randomly. Otherwise, if no edge exists among the tasks in $\mathcal{V}_{min}$, a task with the minimum degree (not including tasks with degree of zero) is selected. The selected task chooses an SV from its permissible SVs list which has the minimum overhead cost. Then, it multicasts the offloading information to all its neighbors in order to remove the selected SV from their own permissible SVs list. Finally, the remaining tasks update their $\mathcal{U}^{cap}$, $\mathcal{G}(\mathbf{V},\mathbf{E})$ accordingly. This procedure continues iteratively.

With defining these priorities for offloading, the above procedure can largely reduce the chance of finding infeasible solutions. To further boost the probability of finding a complete assignment, we then design a *proactive renew rule* as shown in Algorithm 4. In doing so, a task $k_i \in \mathbf{V}$ is selected randomly and the first SV in $L^s(k_i)$ will be selected for offloading of task k_i. If a complete solution is not obtained by DIST-RENEW-DROP, the *proactive renew rule* updates the offloading decision of task k_i to the next SV in $L^s(k_i)$. After checking all permissible SVs of a task if the *proactive renew rule*

Algorithm 2 *DIST-CLIQUE-DROP*

1: $\mathcal{S} \leftarrow \emptyset$ /*offloading decisions*/
2: $s_{min} \leftarrow \min_{k_i \in K} |L(k_i)|$
3: $\mathcal{V}_{min} \leftarrow \{k_i \in K, |L(k_i)| = s_{min}\}$
4: $k_i \leftarrow$ a task from $\mathcal{V}_{min}$
5: $L^s(k_i) \leftarrow$ sort SVs in $L(k_i)$ in ascending order of cost
6: $C^{adj} \leftarrow$ calculate all cliques containing any adjacent task $k_j \in \mathbf{V}$ to k_i
7: **repeat**
8: $u^* \leftarrow$ min cost unchecked SV in $L^s(k_i)$
9: k_i checks whether to select SV u^*
10: u^* is temporarily removed from the permissible SVs list of all neighbors of k_i
11: Flag $\leftarrow True$
12: **for** every clique $c_i \in C^{adj}$ **do**
13: $f_i \leftarrow$ the number of unique permissible SVs of tasks in clique c_i
14: **if** $f_i < |c_i|$ **then**
15: /*Infeasible assignment*/
16: Move to the next unchecked SV
17: Flag $\leftarrow False$
18: **break**
19: **if** Flag **then**
20: /*feasible assignment*/
21: k_i multicasts offloading information to all its neighbors
22: $\mathcal{S}(k_i) \leftarrow u^*$
23: **for** each task $k_j \in \mathbf{V}$ that is adjacent to k_i **do**
24: Receive (k_i, u^*)
25: $\mathbf{V} = \mathbf{V} \setminus \{k_i\}$
26: $\mathcal{T}(u^*, k_j) \leftarrow NaN$
27: **break**
28: **until** Flag or there is no unchecked SV in $L^s(k_i)$
29: **if** Flag is *False* **then**
30: No complete solution
31: $\mathbf{V} = \mathbf{V} \setminus \{k_i\}$
32: Continue to the next iteration from line 2
33: **Return** $\mathcal{S}$

Algorithm 3 *DIST-RENEW-DROP($\mathcal{S}$)*

repeat
  $s_{min} \leftarrow \min_{k_i \in K} |L(k_i)|$
  $\mathcal{V}_{min} \leftarrow \{k_i \in K, |L(k_i)| = s_{min}\}$
  $F \leftarrow$ all tasks $k_i \in \mathcal{V}_{min}$ that has an edge with any $k_j \in \mathcal{V}_{min}$
  if F is non empty **then**
    Select an unchecked $k_i \in F$ with the minimum degree
  else
    Select an unchecked $k_i \in \mathcal{V}_{min}$ with the minimum non-zero degree
  $u^* \leftarrow$ min cost SV in $L^s(k_i)$
  k_i selects SV u^*
  k_i multicasts (k_i, u^*) to all its neighbors
  $\mathcal{S}(k_i) \leftarrow u^*$
  for each task $k_j \in \mathbf{V}$ that is adjacent to k_i **do**
    Receive (k_i, u^*)
    $\mathbf{V} = \mathbf{V} \setminus \{k_i\}$
    $\mathcal{T}(u^*, k_j) \leftarrow NaN$
until all tasks in $\mathcal{V}_{min}$ are assigned
Return $\mathcal{S}$

Algorithm 4 *PR-rule: Proactive Renew Rule*

for task $k_i \in \mathbf{V}$ selected randomly **do**
  $L^s(k_i) \leftarrow$ sort SVs in $L(k_i)$ in ascending order of cost
  for each SV $u^* \in L^s(k_i)$ **do**
    $\mathcal{S} \leftarrow \emptyset$ /*offloading decisions*/
    k_i checks whether to select SV u^*
    u^* is temporarily removed from the permissible SVs list of all neighbors of k_i
    k_i is temporarily removed from $\mathbf{V}$
    $\mathcal{S}(k_i) \leftarrow u^*$
    $\mathcal{S}' \leftarrow$ DIST-RENEW-DROP($\mathcal{S}$)
    if $|\mathcal{S}'| = n$ **then**
      break
  if $|\mathcal{S}'| = n$ **then**
    break
Return $\mathcal{S}'$

cannot find a complete solution, it halts and moves to the next task that is randomly selected. Note that the *proactive renew rule* only applies when Algorithm 3 cannot achieve a complete solution. Our experimental results demonstrate that the *proactive renew rule* greatly improves the probability of finding a complete assignment.

5 EXPERIMENTAL RESULTS

In this section, we comprehensively evaluate the performance of our proposed algorithms from three aspects: finding a complete assignment, the overhead cost, and execution time. For a benchmark, we obtain the optimal solution, called DROP-OPTIMAL, using IBM ILOG CPLEX Optimization Studio for Academics Initiative (Python API). The proposed algorithms and DROP-OPTIMAL are implemented in Python 3.6, and the experiments are conducted on $2.3GHz$ Intel Core i5 with $16GB$ of RAM.

Table 1: Parameters

Parameter	Value	Description
μ_c	40×10^3 Megacycles	mean of CPU cycles of SVs
μ_s	5 MB	mean of memory size of SVs
μ_b	5.0 MHz	mean of transmission bandwidth of SVs
μ_f	5.0 GHz	mean of CPU frequency of SVs
μ_d	100 m	mean of distance between u_0 and other SVs
γ	1.0×10^{-28}	effective switched capacitance of SV
C_i	[1000 – 4000] Megacycles	computational cycles of task k_i
D_i	[100 – 500] KB	data size of task k_i
P_0	0.1 Watts	transmission power of u_0
h	3.4	path-loss factor
N_0	4×10^{-15} Watts	background noise power

5.1 Experimental Setup

We consider a set of ten SVs are scattered across an area in which they can communicate with u_0. As a result, we consider the distance d_{0i} between u_0 and u_i obeys Gaussian distribution with mean μ_d and standard deviation $\sigma_d = 0.25\mu_d$. Moreover, the CPU frequency of SVs obeys Gaussian distribution with mean μ_f and standard deviation $\sigma_f = 0.30\mu_f$.

For the wireless access, we set the transmission bandwidth B_i obeys Gaussian distribution with mean μ_b and standard deviation $\sigma_b = 0.30\mu_b$. We consider diffrenet number of tasks $n = \{20, 30, 40, 50, 60, 70, 80\}$ at u_0. For the computational tasks, similar to the previous studies [16, 19], we set (C_i, D_i) of task k_i uniformly selected from $[1000, 2000, 3000, 4000]$ Megacycles and $[100, 200, 300, 400, 500]$ KB, respectively. To experiment with a set of heterogeneous SVs, CPU cycles C_i^{cap} and memory D_i^{cap} of u_i obey Gaussian distribution with mean μ_c, μ_s and standard deviation $\sigma_c = 0.30\mu_c$, $\sigma_s = 0.30\mu_s$, respectively. In addition, e_j is calculated using $e_j = \gamma F_j^2$ [8], where γ is the effective switched capacitance.

To enable tractable analysis and useful insights, we assume that the energy cost and delay cost are equally important for each tasks k_i, i.e., $\alpha_i = \beta_i = 0.5$. The main parameters used in the experiments are summarized in Table 1.

For the *privacy restrictions*, we setup a well-known random graph model, the Erdös-Rényi model [3], as the conflict graph $\mathcal{G}(\mathbf{V}, \mathbf{E})$. An Erdös-Rényi graph (n, P) is a random graph constructed by connecting vertices randomly, where n is number of tasks and any conflict (edge between any pair of tasks) has a probability of P. Based on the

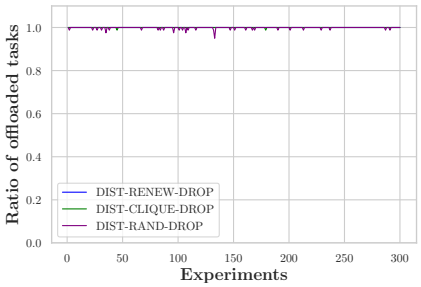
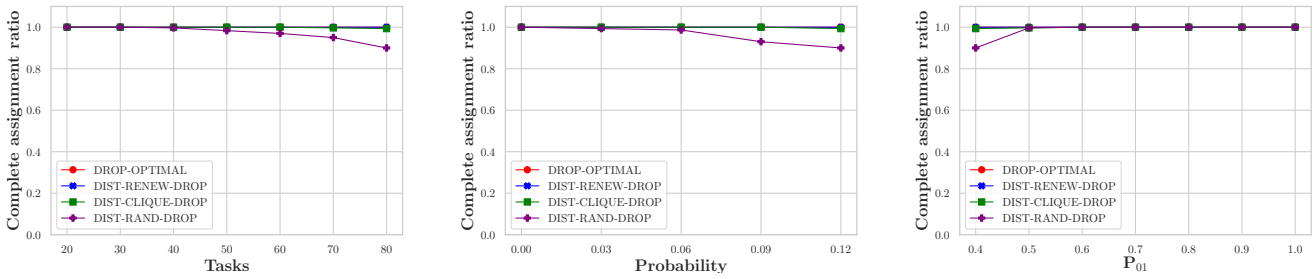


Figure 3: Complete assignment ratio

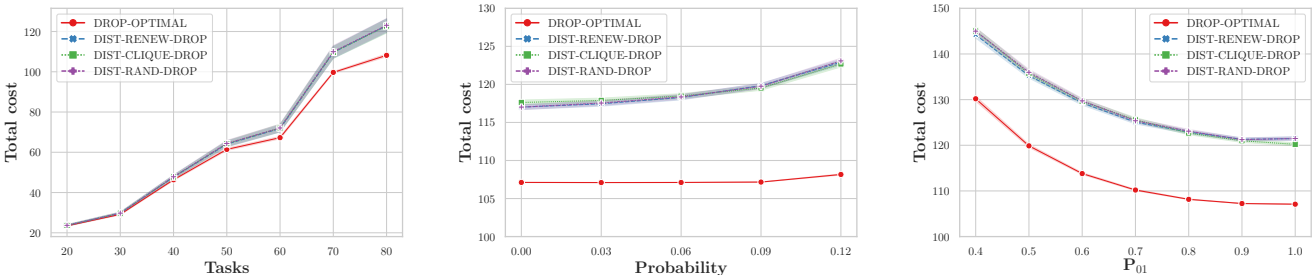


(a) different number of tasks n ($P = 0.12; P_{01} = 0.4$)

(b) different values of P ($n = 80; P_{01} = 0.4$)

(c) different values of P_{01} ($n = 80; P = 0.12$)

Figure 4: Complete Assignments: Sensitivity Analysis



(a) different number of tasks n ($P = 0.12; P_{01} = 0.8$)

(b) different values of P ($n = 80; P_{01} = 0.8$)

(c) different values of P_{01} ($n = 80; P = 0.12$)

Figure 5: Overhead Cost: Sensitivity Analysis

value of $P \in [0, 1]$, we can obtain a sparse or complex conflict graph. For the *accessibility restrictions*, the probability associated with SVs for each task (column) in matrix $\mathcal{T}$ is P_{01} (the probability of generating 1 using the Bernoulli distribution). In our experiments, we study the performance of the proposed algorithms with different values of n, P, and P_{01}.

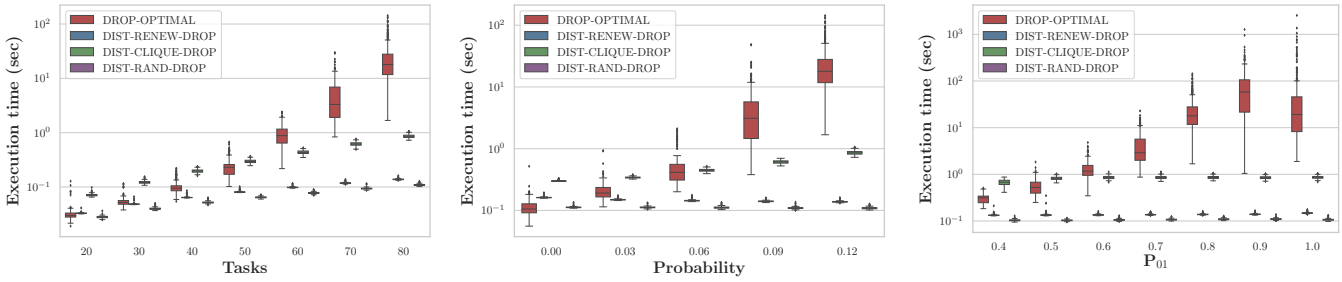
5.2 Analysis of Results

In this subsection, we evaluate the performance of the proposed algorithms in finding complete assignments, obtained overhead cost, and execution time.

We first analyze the assignments to find out the percentage of offloaded tasks in the obtained solutions of our algorithms. We fix $(n, P, P_{01}) = (80, 0.12, 0.4)$ in which both of the *privacy* and *accessibility restrictions* are constricted. As shown in Fig. 3, even though our proposed algorithms cannot always ensure complete assignments, the obtained results in 100 experiments show that the algorithms are able to achieve close to complete assignments, where at least 95.0% of tasks are offloaded.

To further verify the performance of the proposed algorithms in finding complete assignments, we perform sensitivity analysis on the number of tasks n, value of P, and P_{01} (results are shown in Fig. 4). In each set of experiments, we fix the other two parameters. We present the complete assignment ratio defined as the number of experiments with complete assignments over the total number of experiments. Fig. 4a shows the performance of the algorithms with different number of tasks, considering $P = 0.12$ and $P_{01} = 0.4$. The complete assignment ratio is at least 90.0% by DIST-RAND-DROP in 100 experiments. Fig. 4b shows the performance of the algorithms with different values of P, considering $n = 80$ and $P_{01} = 0.4$. The results show that the proposed algorithms are able to achieve a high complete assignment ratio as the probability of having conflicts increases. Fig. 4c presents the performance of the algorithms with different values of P_{01}, considering $n = 80$ and $P = 0.12$. Similarly, the proposed algorithms obtain high complete assignment ratio as P_{01} (SVs accessibility) increases. Overall, DIST-RENEW-DROP outperforms the other two proposed algorithms in terms of complete assignment ratio due to the fact that it executes a *proactive renew rule* once an unfeasible solution happens. DIST-RAND-DROP, however, performs the worst compared to the other two algorithms. Since it does not consider an iterative rule for improving the probability of obtaining a complete assignment.

We then analyze the performance of the proposed algorithms in terms of the obtained overhead cost compared to the optimal cost obtained by DROP-OPTIMAL. As shown in Fig. 5, the proposed algorithms obtain near optimal solutions with respect to different values of N, P, and P_{01}. Specifically, the dashed lines represent the average cost obtained by the proposed algorithms, and the red line indicates the average optimal cost obtained by DROP-OPTIMAL.



(a) different number of tasks n ($P = 0.12; P_{01} = 0.8$)

(b) different values of P ($n = 80; P_{01} = 0.8$)

(c) different values of P_{01} ($n = 80; P = 0.12$)

Figure 6: Execution Time: Sensitivity Analysis

Moreover, the standard deviation of the obtained solutions is presented in shaded error bands (semi-transparent areas). The results show that the the standard deviation is low and the average optimality gap remains small in all experiments. This is due to the fact that our proposed algorithms always allocate a proper SV with minimum cost to the selected task at each round in any experiment.

Finally, we verify the performance of the proposed algorithms in terms of execution time. Fig. 6a shows that the proposed algorithms obtain the results in a short amount of time and scale well as the number of tasks increases. Moreover, their execution time is stable in 100 experiments as the standard deviation is very low compared to the execution time of the optimal algorithm, DROP-OPTIMAL, obtained by CPLEX. We also study the performance of algorithms with different values of P and P_{01}. Again, Fig. 6b and Fig. 6c show that our proposed algorithms are fast. The execution time by DROP-OPTIMAL can be very unstable with respect to the three parameters N, P, and P_{01} due to its NP-hardness. It can only be the best choice for solving DROP when the problem size is very small, which is not practical.

From the above results, we conclude that the proposed algorithms are able to achieve complete assignments with a very high probability. In case that a complete assignment is not achieved, the proposed algorithms are still able to offload almost all tasks. Moreover, the proposed algorithms are scalable and obtain near optimal overhead cost for offloading in a short amount of time.

6 CONCLUSION

In this paper, we formulated the general Data pRotection Offloading Problem (DROP) in a network of capacitated SV-mounted cloudlets and proved its NP-hardness. We proposed three distributed iterative algorithms based on different updating rules. The experimental results demonstrate that the proposed algorithms are efficient in terms of overhead cost and execution time. In addition, they scale well as the system size grows. In our future work, we will study the impacts of SVs mobility on task offloading.

ACKNOWLEDGMENTS

This research was supported in part by NSF grant CNS-1755913.

REFERENCES

[1] Dixit Bhatta and Lena Mashayekhy. 2019. Generalized Cost-Aware Cloudlet Placement for Vehicular Edge Computing Systems. In *Proceedings of the 11th IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. 1–8.

[2] Xu Chen. 2014. Decentralized computation offloading game for mobile cloud computing. *IEEE Transactions on Parallel and Distributed Systems* 26, 4 (2014), 974–983.

[3] P Erdös and A Rényi. 1959. On random graphs. *Publ. Math* 6 (1959), 290–297.

[4] Jingyun Feng, Zhi Liu, Celimuge Wu, and Yusheng Ji. 2018. Mobile edge computing for the Internet of vehicles: Offloading framework and job scheduling. *ieee vehicular technology magazine* 14, 1 (2018), 28–36.

[5] Zhu Han, Yunan Gu, and Walid Saad. 2017. *Matching theory for wireless networks*. Springer.

[6] Dong Huang, Ping Wang, and Dusit Niyato. 2012. A dynamic offloading algorithm for mobile computing. *IEEE Transactions on Wireless Communications* 11, 6 (2012), 1991–1995.

[7] George Iosifidis, Lin Gao, Jianwei Huang, and Leandros Tassiulas. 2015. A double-auction mechanism for mobile data-offloading markets. *IEEE/ACM Transactions on Networking* 23, 5 (2015), 1634–1647.

[8] Seongah Jeong, Osvaldo Simeone, and Joonhyuk Kang. 2017. Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning. *IEEE Transactions on Vehicular Technology* 67, 3 (2017), 2049–2063.

[9] David Jiang, Wang-Chiew Tan, Gang Chen, David Maier, Kian-Lee Tan, Beng Chin Ooi, and Hosagrahar Jagadish. 2014. Federation in Cloud Data Management: Challenges and Opportunities. *IEEE Transactions on Knowledge and Data Engineering* (2014), 1–14.

[10] Ewa Kubicka and Allen J Schwenk. 1989. An introduction to chromatic sums. In *Proceedings of the 17th conference on ACM Annual Computer Science Conference*. ACM, 39–45.

[11] Feng Luo, Chunxiao Jiang, Shui Yu, Jingjing Wang, Yipeng Li, and Yong Ren. 2017. Stability of cloud-based UAV systems supporting big data acquisition and processing. *IEEE Transactions on Cloud Computing* (2017).

[12] Weibin Ma, Xuanzhang Liu, and Lena Mashayekhy. 2019. A Strategic Game for Task Offloading among Capacitated UAV-Mounted Cloudlets. In *Proceedings of the IEEE International Congress on Internet of Things (ICIOT)*. 61–68.

[13] Xiao Ma, Chuang Lin, Xudong Xiang, and Congjie Chen. 2015. Game-theoretic analysis of computation offloading for cloudlet-based mobile cloud computing. In *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*. ACM, 271–278.

[14] Lena Mashayekhy, Mark Nejad, and Daniel Grosu. 2019. A Trust-Aware Mechanism for Cloud Federation Formation. *IEEE Transactions on Cloud Computing (in press)* (2019).

[15] Lena Mashayekhy, Mahyar Movahed Nejad, and Daniel Grosu. 2014. A framework for data protection in cloud federations. In *Proceedings of the 43rd International Conference on Parallel Processing*. 283–290.

[16] Mohamed-Ayoub Messous, Hichem Sedjelmaci, Noureddin Houari, and Sidi-Mohammed Senouci. 2017. Computation offloading game for an UAV network in mobile edge computing. In *Proceedings of the IEEE International Conference on Communications (ICC)*. 1–6.

[17] Mohammad Mozaffari, Walid Saad, Mehdi Bennis, and Mérouane Debbah. 2016. Unmanned aerial vehicle with underlaid device-to-device communications: Performance and tradeoffs. *IEEE Transactions on Wireless Communications* 15, 6 (2016), 3949–3963.

[18] Siani Pearson. 2009. Taking account of privacy when designing cloud computing services. In *Proceedings of ICSE Workshop on Software Engineering Challenges of Cloud Computing*. IEEE, 44–52.

[19] Lingjun Pu, Xu Chen, Jingdong Xu, and Xiaoming Fu. 2016. D2D fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted D2D collaboration. *IEEE Journal on Selected Areas in Communications* 34, 12 (2016), 3887–3901.

[20] Theodore S Rappaport et al. 1996. *Wireless communications: principles and practice*. Vol. 2. prentice hall PTR New Jersey.

[21] Mahadev Satyanarayanan. 2017. The emergence of edge computing. *Computer* 50, 1 (2017), 30–39.

[22] Nafiseh Sharghivand, Farnaz Derakhshan, and Lena Mashayekhy. 2018. QoS-Aware Matching of Edge Computing Services to Internet of Things. In *Proceedings of the 37th IEEE International Performance Computing and Communications Conference*. 1–8.

[23] Neha Sharma, Madhavi Shamkuwar, and Inderjit Singh. 2019. The History, Present and Future with IoT. In *Internet of Things and Big Data Analytics for Smart Generation*. Springer, 27–51.

[24] Olena Skarlat, Vasileios Karagiannis, Thomas Rausch, Kevin Bachmann, and Stefan Schulte. 2018. A framework for optimization, service placement, and runtime operation in the fog. In *Proceedings of the IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*. 164–173.

[25] Jyrki Wallenius, James S Dyer, Peter C Fishburn, Ralph E Steuer, Stanley Zionts, and Kalyanmoy Deb. 2008. Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead. *Management science* 54, 7 (2008), 1336–1349.

[26] Rong Yu, Xumin Huang, Jiawen Kang, Jiefei Ding, Sabita Maharjan, Stein Gjessing, and Yan Zhang. 2015. Cooperative resource management in cloud-enabled vehicular networks. *IEEE Transactions on Industrial Electronics* 62, 12 (2015), 7938–7951.

[27] Daniel Zhang, Yue Ma, Chao Zheng, Yang Zhang, X Sharon Hu, and Dong Wang. 2018. Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing. In *Proceedings of the IEEE/ACM Symposium on Edge Computing (SEC)*. 243–259.

[28] Hongli Zhang, Qiang Zhang, and Xiaojiang Du. 2015. Toward vehicle-assisted cloud computing for smartphones. *IEEE Transactions on Vehicular Technology* 64, 12 (2015), 5610–5618.

[29] Ke Zhang, Yuming Mao, Supeng Leng, Alexey Vinel, and Yan Zhang. 2016. Delay constrained offloading for mobile edge computing in cloud-enabled vehicular networks. In *2016 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*. IEEE, 288–294.

[30] Zhenyu Zhou, Pengju Liu, Junhao Feng, Yan Zhang, Shahid Mumtaz, and Jonathan Rodriguez. 2019. Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach. *IEEE Transactions on Vehicular Technology* 68, 4 (2019), 3113–3125.