This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2872436, IEEE Internet of Things Journal

Chimera: An Energy-efficient and Deadline-aware Hybrid Edge Computing Framework for Vehicular Crowdsensing Applications

Lingjun Pu, Xu Chen, *Member, IEEE,* Guoqiang Mao, *Fellow, IEEE,* Qinyi Xie and Jingdong Xu.

***Abstract*—In this paper, we propose *Chimera*, a novel hybrid edge computing framework, integrated with the emerging edge cloud radio access network, to augment network-wide vehicle resources for future large-scale vehicular crowdsensing applications, by leveraging a multitude of cooperative vehicles and the virtual machine (VM) pool in the edge cloud via the control of the application manager deployed in the edge cloud. We present a comprehensive framework model and formulate a novel multi-vehicle and multi-task offloading problem, aiming at minimizing the energy consumption of network-wide recruited vehicles serving heterogeneous crowdsensing applications, and meanwhile reconciling both application deadline and vehicle incentive. We invoke Lyapunov optimization framework to design *TaskSche*, an online task scheduling algorithm, which only utilizes the current system information. As the core components of the algorithm, we propose a task workload assignment policy based on graph transformation and a knapsack-based VM pool resource allocation policy. Rigorous theoretical analyses and extensive trace-driven simulations indicate that our framework achieves superior performance (e.g., $20\%-68\%$ energy saving without overstepping application deadlines for network-wide vehicles compared with vehicle local processing) and scales well for a large number of vehicles and applications.**

I. Introduction

With the rapid development of autonomous driving technologies, an increasing number of vehicles are equipped with various kinds of sensors and advanced processing units. In the meanwhile, a variety of vehicle-to-vehicle (V2V) and vehicle-to-infrastructure (V2I) communication technologies such as dedicated short range communication (DSRC) and long-term-evolution-vehicle (LTE-V) [1] become increasingly mature. As a result, the Internet of Vehicles (IoV) is emerging and has great potential to boost many attractive applications [2].

Vehicular crowdsensing is one of the most promising IoV applications, which takes advantage of vehicle sensing and processing capability as well as vehicle mobility to provide location-based services such as environmental monitoring and parking lot detection in large areas [3]. In general, a vehicular crowdsensing application requires a crowd of vehicles to periodically sense data from immediate surroundings, in-situ process them such as data fusion and image/video processing, and transmit the processed results within a specific deadline to the centralized application manager (e.g., traffic control center) for post-processing such as spatial and big data analyses [4].

As the era of artificial intelligence and mobile Internet is evolving, future vehicles require to execute many local processing tasks for autonomous driving and user entertainment in terms of QoS, privacy and security issues, and therefore they may be feeble to match the emerging features (e.g., variety and complexity) and requirements (e.g., data volume and processing deadline) of vehicular crowdsensing applications. Also, it should be energy-efficient for vehicles (e.g., electric cars) to serve vehicular crowdsensing applications from drivers' points of view (e.g., economy issue). Therefore, in parallel with the design of vehicle recruitment policy, how to extend vehicle available resources to satisfy the ever-increasing application requirements and meanwhile reduce the corresponding vehicle processing energy is also of significance for the development of vehicular crowdsensing industry.

Task offloading, a promising technology to augment device resources and conserve device energy, has been extensively investigated in both academia and industry. In the last decade, many researchers studied mobile cloud computing [5], where mobile devices can offload computation-intensive tasks to the remote resource-rich clouds. Nevertheless, this paradigm is inadequate in supporting latency-aware IoV applications, due to the large volume of data exchange (e.g., images and videos) and the long network distance between vehicles and clouds. In comparison, mobile edge computing [6] is a novel paradigm aiming to exploit a multitude of devices and facilities (e.g., base stations and edge clouds) at the network edge to cooperatively provide low-latency and elastic resource augmenting services, which is promising to serve the emerging IoV applications.

With this motivation, we propose *Chimera* as illustrated in Fig. 1, a hybrid edge computing framework integrated with the emerging edge cloud radio access network to augment vehicle resources for crowdsensing applications. In this framework, vehicles are enabled to utilize their local resources (i.e., local computing), "fragmented resources" from nearby vehicles (i.e., V2V-based computing), and "chunk resources" from the virtual machine (VM) pool in the edge cloud (i.e.,

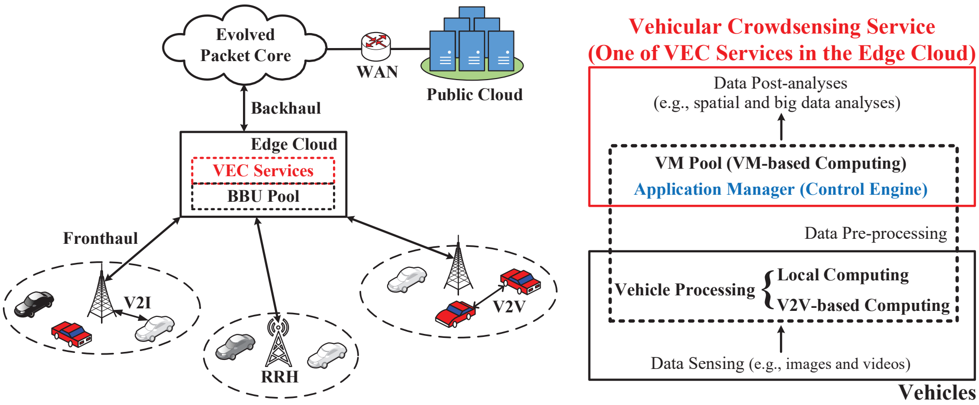
Lingjun Pu is with the College of Computer Science, Nankai University, Tianjin, China, and also with the Guangdong Key Laboratory of Big Data Analysis and Processing, Sun Yat-sen University, Guangzhou, China. (e-mail: pulj@nankai.edu.cn).

Xu Chen is with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. (e-mail: chenxu35@mail.sysu.edu.cn).

Guoqiang Mao is with the School of Computing and Communications, University of Technology Sydney, Sydney, Australia. (e-mail: Guoqiang.Mao@uts.edu.au).

Qinyi Xie and Jingdong Xu are with the College of Computer Science, Nankai University, Tianjin, China. (e-mail: {xieqy, xujd}@nankai.edu.cn).

The corresponding authors are Xu Chen and Jingdong Xu.


Evolved Packet Core
WAN
Public Cloud
Backhaul
Edge Cloud
VEC Services
BBU Pool
Fronthaul
V2I
RRH
V2V
Vehicular Crowdsensing Service
(One of VEC Services in the Edge Cloud)
Data Post-analyses
(e.g., spatial and big data analyses)
VM Pool (VM-based Computing)
Application Manager (Control Engine)
Data Pre-processing
Vehicle Processing
Local Computing
V2V-based Computing
Data Sensing (e.g., images and videos)
Vehicles

(a) The edge cloud radio access network architecture

(b) The vehicular crowdsensing service flow

Figure 1: An illustration of the *Chimera* framework (One of Vehicular Edge Computing (VEC) services).

VM-based computing) to facilitate sensed data processing, via the control of the application manager deployed in the edge cloud. Compared with the existing RSU-based edge computing frameworks (e.g., [7], [8], [9]), our framework does not require additional resource deployment at RSUs, makes full use of vehicle resource sharing, and supports a variety of RAN functionalities such as vehicle handover management and V2V connection control. Besides, compared with the existing V2V-based edge computing frameworks (e.g., [10], [11], [12]), our framework takes advantage of both the powerful computing capacity and the global system information in the edge cloud to make optimal task offloading decisions.

In order to reap the profound benefits of our framework for vehicular crowdsensing applications, we need to fulfill the following requirements:

***Chimera* should be aware of the application heterogeneity.** With the development of vehicular crowdsensing industry, vehicles will support multiple crowdsensing applications simultaneously [3], and the specific features (e.g., processing density) and requirements (e.g., latency) of those applications are distinct, which requires our framework to provide suitable management policy and task offloading scheme for heterogeneous applications in vehicles.

***Chimera* should be online and adaptive to system dynamics.** Due to the dynamic nature of network conditions, vehicle mobility as well as the available resources of vehicles and VM pool in the edge cloud [13], it is difficult to predict the future system information, and hence our framework should operate using the current information only.

***Chimera* should provide a proper incentive scheme to encourage vehicle cooperation.** Our framework will exploit cooperative opportunities among nearby “strangers”, and hence a good incentive scheme that can effectively prevent vehicle free-riding and overloading is highly desirable.

***Chimera* should be energy-efficient, deadline-aware and scalable.** Our framework should achieve optimal task processing, in order to conserve the energy of network-wide vehicles while satisfying the latency requirements of crowdsensing applications, and moreover it should be scalable to accommodate a large number of vehicles and applications.

To address these challenges, we present a comprehensive framework model, formulate a novel multi-vehicle and multi-task offloading problem and devise an optimal online task scheduling algorithm accordingly. The main contributions of this paper are summarized as follows:

We propose a hybrid edge computing framework including an innovative task queueing model with three kinds of task processing modes to manage heterogeneous crowdsensing applications in vehicles (**Section III**). In this context, we formulate a novel multi-vehicle and multi-task offloading problem aiming at minimizing the energy consumption of network-wide recruited vehicles serving heterogeneous crowdsensing applications, and meanwhile reconciling both application latency and vehicle incentive in the long term[1] (**Section IV**).

We invoke Lyapunov optimization framework to realize *TaskSche*, an efficient online task scheduling algorithm which only utilizes the current system information per time slot (**Section V**). As the core components of the algorithm, we devise a task workload assignment policy based on graph transformation and a knapsack-based VM pool resource allocation policy for each time slot. Besides, we make a series of discussions about the framework extensions (**Section VI**).

Rigorous theoretical analyses and extensive trace-driven simulation results corroborate that our framework achieves superior performance without overstepping the deadline of each application, such as 20% – 68% energy saving and 1.5x – 3.6x speedup for network-wide vehicles compared with vehicle local processing, 56% – 130% and 28% – 60% performance gain compared with two state-of-the-art works (i.e., *DualControl* [14] and *D2D-Fogging* [15]), respectively. In addition, our framework scales well to support a large number of vehicles and applications. For example, the running time of the *TaskSche* algorithm is less than 150 ms and 16 ms for 2000 vehicles and 200 applications, respectively (**Section VII**).

[1] We consider the long-term perspective in this paper, since most of the vehicular crowdsensing applications (e.g., environmental monitoring and parking lot detection) require to operate over a period.

II. Related Work

Existing task offloading researches in the mobile edge computing domain can be broadly classified into the following three categories.

Device-centric edge cloud computing (e.g., [16], [17], [18]), where each mobile device can dynamically adjust its CPU frequency and/or cellular speed in terms of QoS requirements (e.g., application latency) to achieve energy-efficient task processing. However, most of the works in this category only study the task offloading problem for a single device, and hence their proposed task offloading schemes cannot be applied to the practical vehicular crowdsensing applications where multiple recruited vehicles compete for limited edge cloud resources. Although our previous work [19] and other game theory based works (e.g., [20], [21]) consider the multi-device task offloading problem, they only work for the single-task case and still do not take into account the competition for edge cloud resources among mobile devices.

Operator-controlled edge cloud computing (e.g., [14], [22], [23]), where an authorized task offloading controller is deployed in the edge cloud to facilitate energy-efficient task processing for network-wide devices regarding joint device and edge cloud resources. However, since the proposed task offloading schemes in most works target the single-task case, they are unsuitable for the heterogeneous vehicular crowdsensing applications. Recently, some efficient task offloading schemes are designed for the multi-task case (e.g., [24], [25]). However, they work in an offline manner (i.e., the task workload is known in advance) and hence cannot benefit crowdsensing applications with uncertain sensed data arrival encountered in practice. In addition, this kind of works normally assumes the device-oriented edge cloud model (i.e., multiple devices share a given amount of computation resources), which may adversely impact the task offloading performance with the increase in the number of recruited vehicles.

Device-centric cooperative computing (e.g., [23], [26], [27]), where each mobile device can opportunistically exploit the under-utilized resources from nearby devices via M2M communications for energy-efficient task processing. This kind of framework can benefit from the massive number of mobile devices, while the proposed task offloading schemes highly rely on the device mobility prediction (e.g., contact duration and frequency) and only consider the single-device optimization rather than the network-wide optimization. Although our previous work [15] devises an energy-efficient cooperative computing scheme achieving network-wide optimization, it merely exploits a best-effort task admission model. That is, the admitted task workload of different applications in a time slot cannot be larger than the device local computing capacity (i.e., no queueing model). In addition, it considers the binary task offloading setting where the admitted task workload is regarded as a whole task and will be either processed locally or offloaded to process at a nearby device, which restricts the throughput of task workload processing.

There are also task offloading works that consider component partition and placement of mobile applications to achieve minimum completion time or maximum workload processing throughput [28], [29]; incorporate wireless power transfer and energy harvesting to extend the lifetime of mobile devices [30], [31]; study energy-efficient workload scheduling in multi-cell or multi-clouds scenarios [32], [33]; design feasible data dissemination and device cooperation in vehicular scenarios [34], [35] or discuss a map-reduce case where the workload of a "huge" task is divided and processed by a crowd of cooperative devices in proximity to achieve low latency and/or energy consumption [36], [37]. Compared with the existing works, **the main novelty of the *Chimera* framework is not only combining edge cloud computing with cooperative computing from the operator-controlled perspective but also revealing the relationship among energy, latency and incentive from a holistic perspective for heterogeneous vehicular crowdsensing applications in network-wide recruited vehicles.** There are also a few researches [24], [38], [39] on the multi-device and multi-task task offloading problem. However, they do not take device cooperative computing into account and only touch on one-shot optimization (i.e., not in the online perspective). Although our recent work [40] also considers the coexistence of cooperative and edge cloud computing, the system model and the problem formulation are entirely different (e.g., the single-task case without queueing model and the time-average constraints in terms of application latency and vehicle incentive). We should emphasize that, the *Chimera* framework is not a simple merge of distinct models from the existing works but exhibits an innovative and comprehensive framework by taking many practical features of the vehicles, crowdsensing applications and edge cloud into account, which presents novel challenges in analyses.

III. Framework Model

As illustrated in Fig. 1, we consider an edge cloud radio access network scenario, in which mobile network providers cooperate with enterprises (e.g., auto companies) or governments (e.g., traffic control center) to build a vehicular crowdsensing platform. Specifically, they deploy an application manager in the edge cloud to supervise a pool $\mathcal{M} = \{1, 2, \dots, M\}$ of crowdsensing applications, and also deploy a series of functional modules in the edge cloud for post-analyzing the processed results uploaded from the recruited vehicles. In addition, we consider that a set $\mathcal{N} = \{1, 2, \dots, N\}$ of vehicles has registered on the crowdsensing platform (i.e., they are willing to participate the crowdsensing service), and some vehicles are recruited to run a suite of crowdsensing applications from the application pool $\mathcal{M}$. For simplicity, we assume that each vehicle $i \in \mathcal{N}$ installs and supports each application $k \in \mathcal{M}$ in advance. In order to augment vehicle resources to facilitate those crowdsensing applications in the network-wide perspective, we consider that the mobile network providers (1) set up a specialized VM pool in the edge cloud; (2) design an incentive scheme to inspire vehicles to cooperatively share resources with each other; (3) propose a control scheme in the application manager to provide the vehicle, application and VM pool management. We assume that our framework operates in a time-slotted pattern and each time slot t has a unit time length[2] for simplicity.

A. Vehicle Resource Model

Computing Capacity: We consider that the available computing capacity $s_i(t)$ (in cycles) of a vehicle i is indicated by the stabilized CPU working frequency $S_i(t)$ together with the current CPU load $\delta_i(t)$ (i.e., percentage of occupied processing capacity by other vehicular applications) in a time slot [15]. That is, $s_i(t) = (1 - \delta_i(t))S_i(t)$ in the context of the time slot with a unit length. Note that, $S_i(t)$ and $\delta_i(t)$ can be estimated by the online frequency-history window method [41].

Cellular Uplink Capacity: Each vehicle can establish a cellular link with the nearest radio remote head (RRH). We assume a quasi-static case where the wireless channels of vehicles are stable during a time slot (e.g., a few seconds in urban scenarios), while they may change across different time slots (e.g., due to vehicle mobility) [42]. At the beginning of a time slot, each vehicle i will choose a cellular transmission power level $p_i^n(t)$ in terms of modulation schemes and power control algorithms [15], and then it can achieve cellular uplink capacity $d_i(t)$ (in bits) as $B_i(t)\log_2\left[1+\frac{p_i^n(t)H_i(t)}{N_0(t)}\right]$. Here, $H_i(t)$ is the channel gain between the vehicle i and the RRH, $B_i(t)$ is the bandwidth allocated for the vehicle i (e.g., several resource blocks), $N_0(t)$ is the background noise. Note that our framework can support other kinds of V2I schemes (e.g., LTE-V-cell [1]), and can also capture the cellular interference among different vehicles by introducing a maximum interference temperature value [43], which will not affect our algorithm design and analysis.

V2V Link Capacity: Although many kinds of V2V communication technologies (e.g., WiFi-direct, LTE-D2D, 802.11p-based DSRC, LTE-V-direct [1]) have been discussed in the literature, we believe only one of them will be used as the standard in the future. In this context, we consider a resource-constrained setting, in which each vehicle will establish and maintain only one stable V2V link in a time slot, due to the limited communication resource and time [15], [27]. Note that, we will discuss the framework performance if the above restriction of V2V link establishment is released in Section VI-D. In practice, each vehicle i can transmit data via both cellular uplink and V2V link simultaneously (i.e., different transmission modes), and similar to the cellular uplink capacity, the V2V link capacity $d_{ij}(t)$ with a nearby vehicle j is predictable by online or offline estimation exploiting RSSI and/or historical data rates [17], [44].

In our framework, we consider that the application manager in the edge cloud cannot control vehicle computing capacity, since the CPU working mode (e.g., frequency adjustment and multi-core scheduling) is generally determined by the vehicle operating system, and also it will not change vehicle transmission management algorithms (e.g., power control schemes) to cater for task offloading, due to the modification overhead and privacy issue. In other words, the cellular and V2V transmission rates are time-varying but are not the control variables in our framework, which is similar to that assumed in [14], [15], [17]. Besides, the application manager in practice will only consider the cooperation between vehicles traveling along the same direction on the road, since they can achieve a more stable and prolonged V2V link.

B. Application Model

We consider that each crowdsensing application requires a crowd of vehicles to periodically sense data from immediate surroundings, in-situ process them such as data fusion and image/video processing (e.g., object recognition and feature detection), and transmit the processed results within a specific deadline (i.e., the expired time of the sensed data in each sensing and pre-processing period) to the centralized application manager for post-processing such as spatial and big data analyses [4]. In our framework, we adopt $\mathbf{1}_{ik} = 1$ to represent a vehicle i is recruited to run an application k by the application manager, and take the pre-processing of sensed data at the vehicle side (e.g., one or several functional modules) as a whole task for ease of heterogeneous application management. In other words, similar to that considered in many existing works [14], [17], [18], [22], [27], our framework will concentrate on the task workload scheduling (i.e., sensed data offloading) for crowdsensing applications.

We denote the processing density of the whole task of a crowdsensing application k by γ_k (i.e., the average amount of CPU cycles to process a bit of sensed data), and consider the processed result size equals to a fraction ρ_k of the input sensed data (e.g., video compression or data fusion). Besides, we denote the average data sensing rate required by an application k by $\varPsi_k$, which is given by the application manager in advance. Then, we assume that $W_i^k(t)$ size (in bits) of new data is sensed by a vehicle i for an application k in each time slot (i.e., $\mathbb{E}[W_i^k(t)] = \varPsi_k$, for all the recruited vehicles $\{i|\mathbf{1}_{ik} = 1, \forall i \in \mathcal{N}\}$ in any a time slot t), since the sensing capacity of vehicles may be occupied by other vehicular applications at any time.

C. VM Pool Model

Besides deploying a series of functional modules for post-analyzing the processed results uploaded from the recruited vehicles, mobile network providers also set up a specialized VM pool in the edge cloud, in order to assist the task processing for the recruited vehicles. In the context of a large number of vehicles, we consider an application-oriented rather than the existing device-oriented model [14], [22], [23] for the VM pool to facilitate resource management. Specifically, each application k is assigned to a dedicated virtual machine with computing capacity v_k (in cycles), and to capture the dynamic and restricted physical resources $P(t)$ of the VM pool (e.g., competing for the resources of the edge cloud with BBU pool and other VEC services) in a time slot, we introduce the following resource constraint:

$$\textstyle\sum_{k=1}^{M} v_k \pi_k(t) \leq P(t), \tag{1}$$

[2] Since most vehicular crowdsensing applications such as parking lot detection and traffic surveillance do not have stringent latency requirement and the sensed data for them are generally queued in the recruited vehicles, the time length of our framework could be at the timescale of several seconds, in order to make full use of computation resources at the network edge and reduce the control overhead of the application manager.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2872436, IEEE Internet of Things Journal

where $\pi_k(t) \in \{0,1\}$ is a binary indicator decided by the application manager. $\pi_k(t)=1$ if the VM pool resources are allocated to the VM of the application k, and 0 otherwise.

D. Task Queueing Model

With the development of vehicular crowdsensing industry, vehicles will support multiple crowdsensing applications simultaneously [3], and the specific features (e.g., processing density) and requirements (e.g., latency) of those applications are often distinct. As such, our framework should provide adaptive management policy for heterogeneous applications in vehicles. To this end, for each application k in a vehicle i we introduce a task workload queue $Q_i^k(t)$ to describe the residual amount of unprocessed sensed data and a task output queue $D_i^k(t)$ to maintain the processed results to be uploaded, as shown in Fig. 2. In addition, we introduce M task workload queues at the VM pool side denoted by $L_1(t), L_2(t), \dots, L_M(t)$ where $L_k(t)$ indicates the residual amount of unprocessed sensed data of an application k offloaded from all the recruited vehicles (i.e., $\{i|\mathbf{1}_{ik}=1, \forall i \in \mathcal{N}\}$) in our framework.

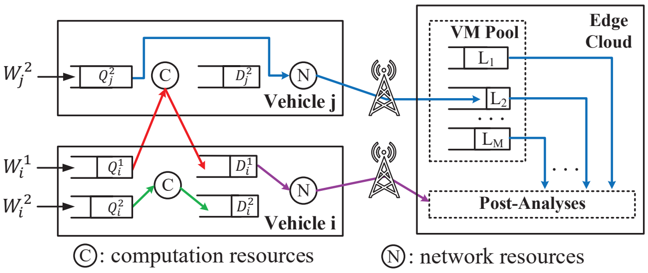


Figure 2: Task queueing model in our framework.

Our framework provides three kinds of task processing modes (i.e., local computing, V2V-based computing and VM-based computing), and hence we respectively introduce three kinds of control variables to indicate them. Specifically, $x_i^k(t)$ represents how much task workload from the $Q_i^k(t)$ is locally processed by the vehicle i, and the processed results are pushed into the $D_i^k(t)$ (i.e., the green arrows in Fig. 2). $y_{ij}^k(t)$ represents how much task workload from the $Q_i^k(t)$ is offloaded to a nearby resource-free vehicle j which after processing will transmit the processed results back to the $D_i^k(t)$ (i.e., the red arrows in Fig. 2), since only the recruited vehicles of each application should upload the processed results[3] (e.g., due to the vehicle recruitment and incentive policies such as monetary reward). $z_i^k(t)$ represents how much task workload from the $Q_i^k(t)$ is offloaded to the dedicated VM of the application k in the VM pool which after processing will forward the processed results to the module for post-analyses in the edge cloud (i.e., the blue arrows in Fig. 2). Besides, we introduce another control variable $u_i^k(t)$ indicating how many processed results from the $D_i^k(t)$ are uploaded to the post-analyses module in the edge cloud (i.e., the purple arrows in Fig. 2). In this context, we can update the queue dynamics (in bits) as follows:

$$Q_i^k(t+1)=\Big[Q_i^k(t)-x_i^k(t)-\textstyle\sum_{j=1}^N \lambda_{ij}(t)y_{ij}^k(t)-z_i^k(t)\Big]^+ +W_i^k(t),$$

$$D_i^k(t+1)=\Big[D_i^k(t)-u_i^k(t)\Big]^+ +\rho_k x_i^k(t) + \rho_k \textstyle\sum_{j=1}^N \lambda_{ij}(t)y_{ij}^k(t),$$

$$L_k(t+1) = \Big[L_k(t)-\pi_k(t)\frac{v_k}{\gamma_k}\Big]^+ +\textstyle\sum_{i=1}^N z_i^k(t),$$

where $[x]^+ \triangleq \max\{x,0\}$ and the binary control variable $\lambda_{ij}(t) \in \{0,1\}$ indicates whether vehicle i and j will establish a V2V link. Since the unit of VM computing capacity is cycle and that of queue backlog is bit, we use the value of computing capacity (e.g., v_k) divided by the processing density (e.g., γ_k) to represent the amount of processed workload from the queue at VM side (e.g., $L_k(t)$) for unit agreement.

E. Energy Consumption Model

In the local computing mode, only vehicle local computation resources are utilized, and hence the vehicle energy consumption can be expressed as follows:

$$E_i^l(t) = p_i(t)\frac{\sum_{k=1}^M x_i^k(t)\gamma_k}{s_i(t)},$$

where $p_i(t)$ is the CPU working power according to the stabilized CPU working frequency in a time slot [14], [41], and the rest part is the time required for task workload processing.

In the V2V-based computing mode, both vehicle computation and V2V transmission resources are utilized, and we can obtain the energy consumption expression as follows:

$$E_{ij}^o(t)=\lambda_{ij}(t)\Big[p_j(t)\frac{\sum_{k=1}^M y_{ij}^k(t)\gamma_k}{s_j(t)}+p_{ij}(t)\frac{\sum_{k=1}^M (1+\rho_k)y_{ij}^k(t)}{d_{ij}(t)}\Big],$$

where $p_{ij}(t)$ is the V2V transmission power, and we assume it and the V2V transmission rate are symmetric between each pair of vehicle i and j for simplicity. In this context, the first part is the computation energy consumption of vehicle j, and the second part is the transmission energy consumption of offloading some workloads to vehicle j plus forwarding the processed results back to vehicle i.

In the VM-based computing mode or in the case of processed result uploading, only cellular transmission resources are utilized, and we can have the energy consumption:

$$E_i^n(t)=p_i^n(t)\frac{\sum_{k=1}^M \big[z_i^k(t)+u_i^k(t)\big]}{d_i(t)},$$

where $p_i^n(t)$ as mentioned in Section III-A is the cellular transmission power. Note that, as our main purpose is to reduce the energy consumption of network-wide recruited vehicles for crowdsensing applications, we neglect the energy cost in the edge cloud which has persistent energy supply.

IV. Problem Formulation

On the basis of the above framework model, we next describe a novel multi-vehicle and multi-task offloading problem formulation with a series of necessary constraints.

[3] Note that we can easily modify our model to let the cooperative vehicles upload the processed results rather than forwarding them back to the recruited vehicles (i.e., substituting $\sum_{j=1}^N \lambda_{ij}(t)y_{ij}^k(t)$ in $D_i^k(t+1)$ expression with $\sum_{j=1}^N \lambda_{ji}(t)y_{ji}^k(t)$). In this case, we also need to introduce another incentive constraint for network resources. We will consider them in the future work.

A. Scheduling Constraint

Considering the vehicle resource limitation, we have the following task workload scheduling constraints:

$$\textstyle\sum_{k=1}^{M}\left[x_i^k(t)+\sum_{j=1}^{N}\lambda_{ji}(t)y_{ji}^k(t)\right]\gamma_k \le s_i(t), \tag{2}$$

$$\textstyle\sum_{k=1}^{M}\left[z_i^k(t)+u_i^k(t)\right] \le d_i(t), \tag{3}$$

$$\textstyle\sum_{k=1}^{M}\lambda_{ij}(t)y_{ij}^k(t) \le d_{ij}(t). \tag{4}$$

Constraint (2) ensures the amount of local task workload plus that offloaded from other vehicles are processed within vehicle available computing capacity in each time slot, constraint (3) ensures that the amount of offloaded task workload plus the amount of uploaded processed results do not exceed vehicle cellular uplink capacity, and constraint (4) ensures that the amount of task workload offloaded to a nearby vehicle is not more than the V2V link capacity between them.

In addition, since the task workload of an application required to process in a time slot cannot exceed the backlog of the corresponding task workload queue, we have the following scheduling constraints:

$$\textstyle x_i^k(t)+\sum_{j=1}^{N}\lambda_{ij}(t)y_{ij}^k(t)+z_i^k(t) \le Q_i^k(t), \tag{5}$$

$$u_i^k(t) \le D_i^k(t). \tag{6}$$

Next, taking vehicle mobility into account, we introduce a time-varying V2V connectivity graph $G(t)=\{\mathcal{N},\mathcal{E}(t)\}$. Specifically, the set of vehicles $\mathcal{N}$ is the vertex set and $\mathcal{E}(t)=\{(i,j): e_{ij}(t)=1,\ \forall i,j\in\mathcal{N}\}$ is the edge set in a time slot, where $e_{ij}(t)=1$ if the vehicle i can establish a feasible V2V link with another vehicle j. Note that, we specify that $e_{ii}(t)\equiv 0$ for each vehicle i in each time slot. Besides, each vehicle cannot establish and maintain more than one V2V link in a time slot, due to the resource and time constraints as mentioned in Section III-A. To summarize, we also have the following scheduling constraints:

$$\lambda_{ij}(t)=0,\ \text{if } e_{ij}(t)\notin\mathcal{E}(t), \tag{7}$$

$$\textstyle\sum_{j=1}^{N}\lambda_{ij}(t)+\sum_{u=1}^{N}\lambda_{ui}(t)\le 1. \tag{8}$$

B. Latency Constraint

Although many existing works [14], [17], [18], [22] assume mobile applications are delay-tolerant, we consider that many vehicular crowdsensing applications in practice are deadline-sensitive (e.g., traffic flow prediction and environmental monitoring). In other words, the incoming sensed data of a crowdsensing application k should be processed, and its processed results should be uploaded before an application-specific deadline T_k. To jointly capture the deadline-sensitive, large-scale (i.e., multiple recruited vehicles) and long-term characteristics of crowdsensing applications, we define the following application latency constraints:

$$\frac{\lim\limits_{T\to\infty}\frac{1}{T}\sum\limits_{t=0}^{T-1}\Big[\sum\limits_{i=1}^{N}\big[Q_i^k(t+1)+\frac{D_i^k(t+1)}{\rho_k}\big]+L_k(t+1)\Big]}{\sum_{i=1}^{N}\mathbf{1}_{ik}\Psi_k}\le T_k, \tag{9}$$

$$\text{All queues } Q_i^k(t),\ D_i^k(t) \text{ and } L_k(t) \text{ are stable.} \tag{10}$$

Constraint (10) ensures our framework is a stable queueing system. Only when the constraint (10) is satisfied, the left-hand side of the constraint (9), in which the numerator is the long-term average amount of *residual* task workloads[4] of an application k in the system and the denominator is the average workload arrival rate of that application in the system, indicates the average application latency in terms of the Little's law principle [45]. Note that as we take the task processing in vehicles and the edge cloud as a serial queueing system, the workload processing and transmission delay in it will be incorporated in the average waiting time. In addition, the exact application latency in a queueing system is the average waiting time plus the average service time (e.g., the transmission time of vehicles uploading processed results). Since our framework operates in a time-slotted manner, the fixed and limited slot length can be roughly viewed as the service time. Therefore, we neglect the service time here for simplicity. To facilitate next description, we introduce two auxiliary parameters $C_k=\sum_{i=1}^{N}\mathbf{1}_{ik}\Psi_k T_k$ and $U_k(t+1)=\sum_{i=1}^{N}(Q_i^k(t+1)+D_i^k(t+1)/\rho_k)+L_k(t+1)$, which enables us to rewrite the constraint (9) as follows:

$$\lim_{T\to\infty}\frac{1}{T}\textstyle\sum_{t=0}^{T-1}U_k(t+1)\le C_k.$$

According to the task queueing model as mentioned in Section III-D, we can easily obtain the following expression:

$$U_k(t+1)=U_k(t)-\sum_{i=1}^{N}\frac{u_i^k(t)}{\rho_k}-\pi_k(t)\min\{\frac{v_k}{\gamma_k},L_k(t)\}+\sum_{i=1}^{N}W_i^k(t).$$

An intuitive interpretation of the above expression is that the first two subtracted items can be viewed as the workload departure of the queueing system, and the last added item can be viewed as the workload arrival of the queueing system. Note that, we substitute "$\min\{u_i^k(t),D_i^k(t)\}$" with "$u_i^k(t)$" in the above expression in terms of the constraint (6).

C. Incentive Constraint

To expand vehicle cooperation in our framework, mobile network providers should design a proper incentive scheme to prevent vehicle free-riding and overloading in the long run.

For vehicle free-riding issue, similar to the prevalent scheme in P2P systems, we introduce a tit-for-tat constraint for computation resources which ensures that a vehicle exploits computation resources from other vehicles only if it contributes sufficient resources to the others. To proceed, we denote by $X_i(t)$ to represent the resource amount that other vehicles contribute to the vehicle i, and $Y_i(t)$ to represent the resource amount the vehicle i contributes to the others in a time slot. According to the task workload scheduling, we can obtain:

$$X_i(t)=\textstyle\sum_{j=1}^{N}\lambda_{ij}(t)\sum_{k=1}^{M}y_{ij}^k(t)\gamma_k,$$
$$Y_i(t)=\textstyle\sum_{j=1}^{N}\lambda_{ji}(t)\sum_{k=1}^{M}y_{ji}^k(t)\gamma_k.$$

[4] Since the physical meaning of the workloads in $D_i^k(t)$ (i.e., the processed results) is different from that in $Q_i^k(k)$, $L_k(t)$ and Ψ_k (i.e., the sensed data), we exploit $D_i^k(t)$ divided by ρ_k for unit agreement. In addition, we exploit $Q_i^k(t+1)$, $D_i^k(t+1)$ and $L_k(t+1)$ to indicate the residual task workloads in the constraint (9), since we take $t=0$ as the starting point.

Let $\overline{X_i}$ and $\overline{Y_i}$ stand for the time averages of $X_i(t)$ and $Y_i(t)$. We then define the resource balance constraint as follows:

$$\alpha_i \overline{X_i} \le \overline{Y_i}, \tag{11}$$

where α_i is within [0,1] for each vehicle $i \in \mathcal{N}$.

For vehicle overloading issue, we introduce an energy budget constraint which ensures that each vehicle will not sacrifice excessive energy (i.e., computation and transmission energy) to benefit others. Specifically, we define K_j^o as the average energy budget of each vehicle j for helping other vehicles and $K_j(t)$ as the real energy consumption contributed to the others in a time slot including workload computation cost and processed result transmission cost, i.e.,

$$K_j(t)=\textstyle\sum_{i=1}^{N}\lambda_{ij}(t)\Big[p_j(t)\dfrac{\sum_{k=1}^{M}y_{ij}^k(t)\gamma_k}{s_j(t)}+p_{ij}(t)\dfrac{\sum_{k=1}^{M}\rho_k y_{ij}^k(t)}{d_{ij}(t)}\Big].$$

Then, we define the energy budget constraint as follows:

$$\overline{K_j}=\lim_{T\to\infty}\frac{1}{T}\textstyle\sum_{t=0}^{T-1}K_j(t)\le K_j^o. \tag{12}$$

In practice, the value of α and K^o are decided by mobile network providers. We should emphasize that, the vehicle incentive information (i.e., $X_i(t)$, $Y_i(t)$ and $K_i(t)$) is maintained by the application manager in our framework, and the above incentive schemes are feasible but not the only choice. In practice, mobile network providers can also design social-based, effort-based or monetary-based incentive schemes.

D. Problem Formulation

In our framework, the objective is to design a joint control algorithm for workload assignment and VM pool resource allocation (i.e., $x_i^k(t)$, $\lambda_{ij}(t)$, $y_{ij}^k(t)$, $z_i^k(t)$, $u_i^k(t)$, $\pi_k(t)$, $\forall i,j\in\mathcal{N}$ and $\forall k\in\mathcal{M}$) in the application manager, so as to minimize the energy consumption of network-wide vehicles for task processing of heterogeneous crowdsensing applications, and meanwhile reconciling both application latency and vehicle incentive in a long-term perspective. To this end, we formulate the following optimization problem:

$$\begin{aligned}&\text{minimize} && \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{N}\mathbb{E}\Big[E_i^l(t)+\sum_{j=1}^{N}E_{ij}^o(t)+E_i^n(t)\Big] \quad (13)\\ &\text{subject to} && (1)-(12).\end{aligned}$$

Here, we take the expectation form in the objective function due to the random variable $W_i^k(t)$ as mentioned in Section III-B. In practice, the vehicle information in terms of available resources and task queues can be piggybacked on the cellular control messages (e.g., uplink training) or the control messages shared with other VEC services to the edge cloud for making task scheduling decision, which will not lead to much energy consumption for vehicles. More specific analyses have been given in our previous works [15], [19]. Therefore, we do not incorporate it into the problem formulation for simplicity.

V. TaskSche Algorithm Design

The main challenge of solving the problem in (13) lies in the time-average objective and constraints which require the unpredictable future system information such as available vehicle resources and V2V connectivity, due to dynamic local application usages in vehicles and vehicle mobility. In this paper, we design *TaskSche*, an online task scheduling algorithm which only utilizes the current system information per time slot for the problem in (13), by invoking the Lyapunov drift-plus-penalty framework [46].

A. Problem Transformation

The basic idea of this framework has two-fold. First is to use the stability of virtual queues to ensure that the time average constraints are satisfied. Second is to transform the original problem with the drift-plus-penalty expression so as to stabilize the virtual queues while optimizing the objective. In this following, we will discuss how to adopt this framework for our problem in detail. In order to capture the time-average application latency as well as vehicle incentive constraints, we introduce the following virtual queues:

$$\begin{aligned}A_k(t+1)&=A_k(t)-C_k+U_k(t+1),\\ H_i(t+1)&=H_i(t)-Y_i(t)+\alpha_i X_i(t),\\ Z_i(t+1)&=Z_i(t)-K_i^o+K_i(t),\end{aligned}$$

where $U_k(t+1)$, $\alpha_i X_i(t)$ and $K_i(t)$ can be viewed as arrival rates, while C_k, $Y_i(t)$ and K_i^o can be viewed as departure rates. We assume the initial backlogs of these virtual queues are all 0. According to the queueing theory, if these virtual queues are stable (i.e., the average arrival rate is no more than the average departure rate), then the constraints (9), (11) and (12) can be satisfied. With this principle in mind, we adopt the Lyapunov drift-plus-penalty framework to jointly consider the objective optimization and the stability of both the virtual queues (i.e., the constraints (9), (11), (12)) and the real queues (i.e., the constraint (10)). Specifically, we first define a quadratic Lyapunov function:

$$\begin{aligned}&R(\mathbf{\Theta}(t))=\frac{1}{2}\Big\{R_1(\mathbf{\Theta}(t))+R_2(\mathbf{\Theta}(t))\Big\},\\ &R_1(\mathbf{\Theta}(t))=\textstyle\sum_{i=1}^{N}\sum_{k=1}^{M}\omega_k^2\big[Q_i^k(t)^2+D_i^k(t)^2\big]+\sum_{k=1}^{M}\omega_k^2 L_k(t)^2,\\ &R_2(\mathbf{\Theta}(t))=\textstyle\sum_{k=1}^{M}\omega_k^2 A_k'(t)^2+\sum_{i=1}^{N}\big[H_i'(t)^2+Z_i'(t)^2\big].\end{aligned}$$

Here, $A_k'(t)$ equals to $A_k(t)$ if $A_k(t)>0$, and 0 otherwise. In other words, $A_k'(t)\ge A_k(t)$ in each time slot. So do $H_i'(t)$ and $Z_i'(t)$. $\mathbf{\Theta}(t)$ is the vector of all real and virtual queue backlogs in our framework. We introduce R_1, which is widely used to guarantee queue stability in the Lyapunov function [14], [15], [17], [18], [46], for the constraint (10), and introduce R_2 for the application latency constraint (9) and the incentive constraints (11), (12). We should emphasize that, we do not need to take those virtual queues with negative backlogs into account, since they do not violate the time-average constraints at that time. In addition, we introduce a set of weights $\omega_k, \forall k\in\mathcal{M}$ to reveal the “importance” of applications, which is decided by the application manager in advance. Intuitively, a larger weight implies that the application manager has a higher preference on prioritizing the sensed data processing of that application. We will discuss the impact of different weights on application performance in Section VII.

Note that, since the unit of queue Q, D, L, A is bit, the unit of queue H is cycle and the unit of queue Z is joule, we should respectively normalize their backlogs with the maximum sensed data amount, the maximum shared computing capacity and the maximum shared energy consumption among all vehicles for unit agreement in the above Lyapunov function. These maximum values can be specified or learned by the application manager in practice. Then, we define the following one-slot Lyapunov drift-plus-penalty expression:

$$\Delta(\boldsymbol{\Theta}(t)) + V \textstyle\sum_{i=1}^{N} \mathbb{E}\Big[E_i^l(t) + \sum_{j=1}^{N} E_{ij}^o(t) + E_i^n(t)\Big],$$
$$\Delta(\boldsymbol{\Theta}(t)) \triangleq \mathbb{E}\big[R(\boldsymbol{\Theta}(t+1)) - R(\boldsymbol{\Theta}(t)) \mid \boldsymbol{\Theta}(t)\big],$$

where V is a tradeoff parameter between objective optimality and queue stability, and its value is set by the application manager in advance (e.g., according to some small-scale experiments). We will study the relationship between the V's value and the framework performance by simulation in Section VII. In terms of the Lyapunov drift-plus-penalty expression, the problem in (13) can be transformed as follows (note that, the time-average constraints (9) – (12) have been captured in the $\Delta(\boldsymbol{\Theta}(t))$):

$$\text{minimize} \quad \Delta(\boldsymbol{\Theta}(t)) + V \sum_{i=1}^{N} \mathbb{E}\Big[E_i^l(t) + \sum_{j=1}^{N} E_{ij}^o(t) + E_i^n(t)\Big]$$
$$\text{subject to} \quad (1), (2), (3), (4), (5), (6), (7) \text{ and } (8).$$

Since the new objective function includes quadratic terms which are difficult to tackle, we instead minimize an upper bound of the objective function which is given in Lemma 1.

Lemma 1: Under any possible value of $\boldsymbol{\Theta}(t)$ and any feasible scheduling policy, the drift-plus-penalty expression satisfies:

$$\Delta(\boldsymbol{\Theta}(t)) + V \textstyle\sum_{i=1}^{N} \mathbb{E}\Big[E_i^l(t) + \sum_{j=1}^{N} E_{ij}^o(t) + E_i^n(t) \Big| \boldsymbol{\Theta}(t)\Big]$$
$$\leq \mathbb{E}\Big[F^* + f(t) + \sum_{i=1}^{N}\sum_{k=1}^{M} c_i^k(t) x_i^k(t) + \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{M} c_{ij}^k(t)\lambda_{ij}(t) y_{ij}^k(t)$$
$$+ \sum_{i=1}^{N}\sum_{k=1}^{M} a_i^k(t) z_i^k(t) + \sum_{i=1}^{N}\sum_{k=1}^{M} b_i^k(t) u_i^k(t) + \sum_{k=1}^{M} e_k(t)\pi_k(t) \Big| \boldsymbol{\Theta}(t)\Big],$$

where the factors are expressed as follows:

$$c_i^k(t) = \omega_k^2\Big[D_i^k(t)\rho_k - Q_i^k(t)\Big] + V p_i(t)\frac{\gamma_k}{s_i(t)},$$
$$c_{ij}^k(t) = \omega_k^2\Big[D_i^k(t)\rho_k - Q_i^k(t)\Big] + V\Big[\frac{p_j(t)\gamma_k}{s_j(t)} + \frac{p_{ij}(t)(1+\rho_k)}{d_{ij}(t)}\Big]$$
$$+ \Big[\alpha_i H_i'(t)\gamma_k - H_j'(t)\gamma_k\Big] + Z_j'(t)\Big[\frac{p_j(t)\gamma_k}{s_j(t)} + \frac{p_{ij}(t)\rho_k}{d_{ij}(t)}\Big],$$
$$a_i^k(t) = \omega_k^2\Big[-Q_i^k(t) + L_k(t)\Big] + V p_i^n(t)\frac{1}{d_i(t)},$$
$$b_i^k(t) = \omega_k^2\Big[-D_i^k(t) - \frac{A_k'(t) + U_k(t)}{\rho_k}\Big] + V p_i^n(t)\frac{1}{d_i(t)},$$
$$e_k(t) = \omega_k^2\Big[-L_k(t) - A_k'(t) - U_k(t)\Big]\min\{L_k(t), \frac{v_k}{\gamma_k}\}.$$

The factor F^* is a constant value across all time slots and $f(t)$ involves no scheduling indicators. Therefore, we do not consider them in the task scheduling algorithm design. Due to the page limit, we leave their specific expressions and the detailed proof of Lemma 1 to the online technical report [47].

B. TaskSche Algorithm

According to the transformed problem and Lemma 1, we propose *TaskSche*, an online task scheduling algorithm to minimize the drift-plus-penalty upper bound (i.e., the last five terms on the right-hand-side), subject to the constraints (1) – (8), which can be proved to achieve a good performance for the problem in (13). We formally state this algorithm as follows:

- **Step 1**. Find the optimal workload assignment for the task queue of each crowdsensing application in each vehicle:
$$\min_{\{\boldsymbol{x},\boldsymbol{\lambda},\boldsymbol{y},\boldsymbol{z},\boldsymbol{u}\}} \quad \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{M} c_{ij}^k(t)\lambda_{ij}(t) y_{ij}^k(t)$$
$$+ \sum_{i=1}^{N}\sum_{k=1}^{M}\Big[c_i^k(t)x_i^k(t) + a_i^k(t)z_i^k(t) + b_i^k(t)u_i^k(t)\Big]$$
$$\text{subject to} \quad (2), (3), (4), (5), (6), (7) \text{ and } (8).$$
- **Step 2**. Find the optimal VM pool resource allocation:
$$\min_{\{\boldsymbol{\pi}\}} \quad \sum_{k=1}^{M} e_k(t)\pi_k(t) \quad \text{subject to} \quad (1).$$
- **Step 3**. Update the real and virtual queues in the framework based on the results in Step 1 and Step 2.

We should emphasize that, although the Lyapunov drift-plus-penalty framework is a general optimization tool, the design of Lyapunov function and the upper bound of drift-plus-penalty expression are problem-specific, which also leads to significant differences in the corresponding algorithm design. We next elaborate on the implementation for the *TaskSche* algorithm, which is the core building block of our framework.

VI. Algorithm Implementation

In this section, we respectively provide the algorithm implementation for Step 1 and Step 2, then analyze the performance and complexity of the *TaskSche* algorithm, and discuss framework practicality and some extensions in the end.

A. Algorithm Implementation for Step 1

The problem in Step 1 has both linear constraints (2) – (6) and integer constraints (7), (8), and its objective function includes "quadratic" term (i.e., the linear variable $y_{ij}^k(t)$ × the integer variable $\lambda_{ij}(t)$), which is non-trivial to be solved directly. To this end, we will decouple the integer and linear variables in the workload assignment policy design. It is ready to observe that under any workload assignment policy, each recruited vehicle has either no V2V links (i.e., **single vehicle case**) or exactly one V2V link with a nearby vehicle (i.e., **coupled vehicle case**), in terms of the integer constraints. Therefore, our basic idea is to first study the optimal workload assignment in these two cases for each recruited vehicle with only the linear constraints, and then consider the **network-wide vehicle case** with the integer constraints.

Specifically, we first study the task workload assignment in the single vehicle case or called self-assignment case (i.e., $\lambda_{ij}(t) = 0, \forall i, j \in \mathcal{N}$). In this context, we require to solve the following problem:

$$\min_{\{\boldsymbol{x},\boldsymbol{z},\boldsymbol{u}\}} \quad \sum_{k=1}^{M}\Big[c_i^k(t)x_i^k(t) + a_i^k(t)z_i^k(t) + b_i^k(t)u_i^k(t)\Big]$$
$$\text{subject to} \quad (2), (3), (5), (6).$$

This is a linear programming problem with $3\times M$ variables and $4\times M$ constraints, which can be optimally and efficiently solved by many mature algorithms (e.g., simplex method).

We next consider the coupled vehicle case or called V2V-assignment case. Using the notation m to indicate task workload offloader and n to indicate task workload offloadee respectively, then we require to solve the following subproblem twice for each pair of vehicles i, j with an available V2V link (i.e., $i\!=\!m$, $j\!=\!n$ and $i\!=\!n$, $j\!=\!m$):

$$\min_{\{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z},\boldsymbol{u}\}} \sum_{k=1}^{M} \Big[c_m^k(t)x_m^k(t)+c_{mn}^k(t)y_{mn}^k(t)+a_m^k(t)z_m^k(t)$$
$$+b_m^k(t)u_m^k(t)+c_n^k(t)x_n^k(t)+a_n^k(t)z_n^k(t)+b_n^k(t)u_n^k(t)\Big]$$
$$\text{subject to} \qquad (2), (3), (4), (5), (6).$$

Note that, this is also a linear programming problem with $7\times M$ variables and $5\times M$ constraints, which can be optimally solved by the similar algorithms for the above single vehicle case.

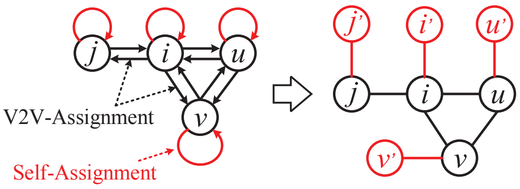


Figure 3: A graph model for the task workload assignment.

At last, we consider the network-wide vehicle case (i.e., the problem in Step 1) such that each recruited vehicle will conduct either self-assignment or V2V-assignment. We propose a graph transformation model, as shown in Fig. 3. On the left-hand-side of the figure, a self-loop indicates the self-assignment of the vehicle and its weight is the optimal result of the problem in the single vehicle case; a directed edge between two nodes indicates two vehicles have a V2V link for cooperation (i.e., the constraint (7)), the starting point of the edge is the task workload offloader and the edge weight is the optimal result of the problem in the coupled vehicle case. Since each vehicle can choose either self-assignment or V2V-assignment, we can transform the problem in Step 1 into a matching problem over the transformed graph on the right-hand-side of the figure. Specifically, we remove the self-loops in the graph on the left-hand-side, duplicate each node i (the duplicated node is denoted as i'), and add an edge between each node and its duplicated node in the new graph. Then we substitute the directed edges between each pair of nodes with an undirected edge, and define the weight of the new edge as the smaller weight of those two directed edges. By doing so, we can obtain a new graph as shown on the right-hand-side of Fig. 3. According to the constraint (8), we require to find a matching over the new graph, such that a node can either match up with its duplicated node (i.e., choosing self-assignment) or another node (i.e., choosing the V2V-assignment). In order to solve this problem, we can apply the Edmonds's Blossom algorithm to obtain the minimum weighted graph matching solution, which is also the optimal solution for the problem in Step 1.

B. Algorithm Implementation for Step 2

In terms of solving the problem in Step 2, we can easily observe that the value of factor $e_k(t)$ is non-positive across all time slots. Since minimizing the objective with non-positive factors is equivalent to maximizing the objective with non-negative factors, we can treat the problem in Step 2 as a standard $0-1$ knapsack problem, which can be optimally or approximately solved by many mature algorithms (e.g., dynamic programming and greedy methods).

After the application manager in the edge cloud implemented the aforementioned algorithms to obtain the workload assignment and resource allocation results in Step 1 and Step 2, it will notify these results to each recruited vehicle and the VM pool to conduct task workload processing, and they subsequently update their real queues. Note that, the data size of the notified results is limited (i.e., $\boldsymbol{x},\boldsymbol{y},\boldsymbol{u},\boldsymbol{z}$), and therefore it will not introduce much overhead to the edge cloud. Besides, the application manager will supervise the V2V-based computing, and if the V2V cooperation is successful, it will update its maintained virtual queues.

C. Performance and Complexity Analysis

The following theorem proclaims the minimum expected time-average energy consumption of network-wide vehicles running crowdsensing applications, and the backlog bound of the time-average real and virtual queues achieved by the *TaskSche* algorithm.

Theorem 1: For each recruited vehicle i, suppose its average workload arrival rate of each crowdsensing application k denoted by $\overline{W_i^k}$ is strictly within the system capacity region[5] Ω (i.e., $\overline{W_i} + \epsilon_i^k \in \Omega$). Here, ϵ_i^k is a small positive value, and without loss of generality, we denote $\epsilon = \min \epsilon_i^k$, $\forall i \in \mathcal{N}$, $\forall k \in \mathcal{M}$. In this context, we can derive

$$\lim_{T\to\infty}\frac{1}{T}\sum\nolimits_{t=0}^{T-1}\mathbb{E}\big[J(t)\big] \le \sum\nolimits_{i=1}^{N}\sum\nolimits_{k=1}^{M}E^*(\overline{W_i^k}+\epsilon_i^k)+\frac{F^*}{V},$$
$$\lim_{T\to\infty}\frac{1}{T}\sum\nolimits_{t=0}^{T-1}\mathbb{E}\Big[\sum\nolimits_{i=1}^{N}\sum\nolimits_{k=1}^{M}\omega_k^2\big(Q_i^k(t)+D_i^k(t)\big)$$
$$+\sum\nolimits_{k=1}^{M}\omega_k^2\big(L_k(t)+A_k(t)+U_k(t)\big)+\sum\nolimits_{i=1}^{N}\big(H_i(t)+Z_i(t)\big)\Big]$$
$$\le \lim_{T\to\infty}\frac{1}{T}\sum\nolimits_{t=0}^{T-1}\mathbb{E}\Big[\sum\nolimits_{i=1}^{N}\sum\nolimits_{k=1}^{M}\omega_k^2\big(Q_i^k(t)+D_i^k(t)\big)$$
$$+\sum\nolimits_{k=1}^{M}\omega_k^2\big(L_k(t)+A'_k(t)+U_k(t)\big)+\sum\nolimits_{i=1}^{N}\big(H'_i(t)+Z'_i(t)\big)\Big]$$
$$\le \frac{1}{\epsilon}\Big(F^*+V\sum\nolimits_{i=1}^{N}\sum\nolimits_{k=1}^{M}E^*(\overline{W_i^k}+\epsilon_i^k)\Big).$$

Here, $J(t)$ is the energy cost of network-wide vehicles running crowdsensing applications in terms of the *TaskSche* algorithm, and $E^*(\overline{W_i^k}+\epsilon_i^k)$ is the offline minimum energy cost of a vehicle i for executing average $\overline{W_i^k}+\epsilon_i^k$ amount of task workload of an application k. The detailed proof is provided in our online technical report [47]. Theorem 1 shows that the *TaskSche* algorithm can achieve the offline optimum approximately as V increases. In addition, it shows that all the

[5] This is a reasonable assumption, since the average workload arrival rate of an application k served by each vehicle i (i.e., $\overline{W_i^k}$) should be close to the required data sensing rate of that application (i.e., Ψ_k) which is an empirical value given by the application manager in advance and therefore it should be within the system capacity region.

real and virtual queues in the framework are rate mean stable, which indicates our framework is a stable queueing system and the time-average constraints (i.e., the application latency and vehicle incentive) can be guaranteed (i.e., the time-average arrival rate is no more than the time-average service rate when the queue is stable).

Note that the *TaskSche* algorithm includes a task workload assignment policy, a VM pool resource allocation policy plus a queue updating policy. Then, the complexity of the first policy is made up of solving the linear programming problem in the single vehicle case for all recruited vehicles (i.e., $N\times\mathcal{O}(M^2)$), solving the linear programming problem in the coupled vehicle case for all pairs of vehicles with an available V2V link (i.e., $N^2\times\mathcal{O}(M^2)$), and running the Edmonds's Blossom algorithm for the optimal graph matching (i.e, $\mathcal{O}(N^3)$). In general, since the amount of crowdsensing applications is far less than that of recruited vehicles and the coupled vehicle case can be finished in parallel, we consider that the complexity of the first policy is dominated by $\mathcal{O}(N^3)$. The complexity of the second policy is a pseudo polynomial time $\mathcal{O}(M\times P(t))$ in each time slot if the classic dynamic programming policy is adopted, and that of the last policy is intuitively $\mathcal{O}(N)$. To sum up, we can have the complexity of the *TaskSche* algorithm is $\mathcal{O}(N^3)+\mathcal{O}(M\times P(t))$ in each time slot.

In practice, we consider that the application manager can adopt many techniques to accelerate the algorithm execution. First, it can only consider the control variables with negative values per time slot, in terms of the problem form in Step 1. In addition, it can utilize cluster partition for vehicles, and executes the optimal graph matching for each cluster in parallel. Moreover, it can adopt some approximate implementations (e.g., greedy methods with 0.5 approximation ratio) for both graph matching and knapsack problem to reduce the complexity. At last, it can also deploy many powerful CPUs or GPUs to facilitate the algorithm execution.

D. Framework Practicality and Extensions

As for the framework practicality, we consider that our framework can be viewed as a functional component of the emerging edge cloud radio access network, which is proposed as a novel and promising architecture for future cellular networks by combining RAN with cloud computing to tackle the ever-increasing cellular traffic (e.g., mobile multimedia traffic). In addition, we assume that our framework has lower priority compared with those RAN-based functional components such as RRH sleeping scheduling and BBU resource allocation, and will operate adaptively in terms of their decisions. Besides, our framework is established on the basis of the cooperation of mobile network operators, vehicles, enterprises and governments, and makes full use of their resources (e.g., vehicle sensing and processing, edge cloud processing and controlling) at the network edge. At last, our framework enables the application manager deployed in the edge cloud to watch and control all participants serving crowdsensing applications (e.g., information collection and scheduling decision notification). Although the algorithm introduces some transmission and control overhead to the edge cloud, it achieves security, robustness and efficiency.

We next discuss some extensions of our framework.

(1) Cellular transmission rate controlling: Our framework can easily incorporate the control of vehicle cellular transmission rate (i.e., $d_i(t)$). Specifically, we consider a selection of cellular transmission rates corresponding to Modulation and Coding Scheme levels. For a certain target bit error rate (BER), different transmission rates require different SNRs, which are determined by transmission power and wireless channel gain. Since wireless channel gain is time-varying, we introduce a set of feasible transmission modes (i.e., $M_i(t)$) for each vehicle (i.e., each mode is a combination of transmission rate and power for a certain BER target and a given wireless channel gain) in each time slot. In this context, the application manager requires to choose an adaptive transmission mode index $m \in M_i(t)$ for each recruited vehicle i to facilitate task offloading. As such, the linear programming problem in the single (coupled) vehicle case becomes a mix-integer programming problem due to the constraint (3). Although this problem is NP-hard in general, it might be still efficiently solved by some optimization tools (e.g., GLPK optimizer), due to the limited number of variables and constraints.

(2) Batch processing model: Although our framework currently considers the streaming processing model, in which we exploit one bit as the unit size as mentioned in Section III, it can also support batch processing model by substituting the unit "bit" with the unit "task block" and introducing the task block size s_k of each application k. In this context, $W_i^k(t)$ indicates how many task blocks are generated by vehicle i for application k, and $\lfloor\frac{d_i(t)}{s_k}\rfloor$ indicates how many task blocks of application k can be offloaded from vehicle i in a time slot. Besides, the linear programming problem in the single (coupled) vehicle case becomes an integer programming problem, since the control variables x, y, u, z are required to be integers. Note that we can also leverage some optimization tools (e.g., GLPK optimizer) to solve it.

(3) Fine-grained VM queueing model: Our framework currently assumes that the application VM has one task workload queue to maintain the residual amount of unprocessed sensed data offloaded from all the recruited vehicles. Nevertheless, different recruited vehicles in practice may be given different priorities to leverage the VM resources (e.g., due to some reputation or incentive schemes), and may be required to sense data with different rates or upload the processed results with different deadlines (e.g., due to different geographical locations). These individualized features cannot be captured by the current VM queueing model. To this end, we can consider a more fine-grained VM queueing model, where each application VM creates a task workload queue and a profile with individualized features for each recruited vehicle. In this context, the optimal VM pool resource allocation will include two stages in each time slot. The first stage is to allocate the resources of each application VM (i.e., the computing capacity v_k) to the recruited vehicles, in terms of their queue backlogs and individualized features, which can be modeled as a linear programming problem. The second stage is to allocate the VM pool resources to the application VM, in terms of the optimal VM resource allocation results in the first stage, which is a $0-1$ knapsack problem. Note that, each application VM can solve

the linear programming problem in the first stage individually, without leading to much time consumption.

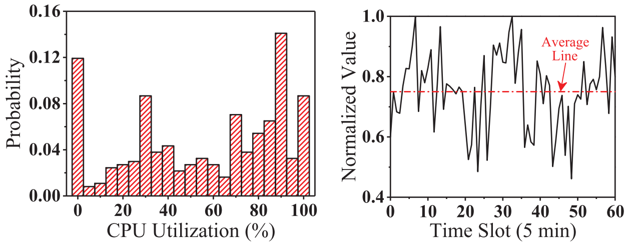
(4) Comparison with multiple simultaneous V2V link establishment: Our framework currently considers that each vehicle will only establish and maintain one stable V2V link, due to the communication resource and time constraints in a time slot (i.e., single V2V link establishment at a time). If we release this assumption and allow each vehicle i to establish at most λ_i^* V2V links in a time slot, then the constraint (8) will become $\sum_{j=1}^{N} \lambda_{ij}(t) + \sum_{u=1}^{N} \lambda_{ui}(t) \leq \lambda_i^*$. It is not hard to see that this new constraint invalidates our task workload scheduling policy in Section VI-A, and we have to solve the problem in Step 1 (i.e., a mixed-integer quadratic programming problem with $\mathcal{O}(N \times M)$ variables and constraints) directly. Although some optimization tools (e.g., CPLEX optimizer) can solve this NP-hard problem to obtain the optimal solution, they will consume substantial amounts of time (e.g., tens or hundreds of seconds), which is inapplicable for our framework in practice. Alternatively, we can search for some fast heuristic methods to solve the problem, while they will impact the performance analysis of *TaskSche* algorithm (i.e., Theorem 1 does not hold anymore), and degrades the long-term framework performance. In order to draw a conclusion, we compare the framework performance in the current resource-constrained setting with that in the resource-infinite setting (i.e., the ideal case) where each vehicle can establish an arbitrary number of V2V links in a time slot (i.e., the control variable $\lambda_{ij}(t)$ and the constraint (8) can be removed and the problem in Step 1 becomes a linear programming problem) by simulation. Numerical results in Fig. 7 show that the performance gap between them is roughly $5\% - 10\%$ in most cases, which indicates that the single V2V link establishment is sufficient for task workload processing in our framework from the perspective of both performance and control complexity.

VII. Performance Evaluation

In this section, we evaluate the performance of the *Chimera* framework through extensive trace-driven simulations.

A. Real Measurements

To facilitate our simulations, we conduct real measurements on Android smartphones to capture the dynamic vehicle computing capacity and the processing density of typical data processing modules, and perform a simple statistic in terms of the Google cluster data trace to capture the dynamic VM pool resources. Specifically, to simulate the available vehicle computing capacity, we first pin the CPU frequency of all tested smartphones to 2 GHz, and then adopt the “Another Monitor” application [48] to record the realtime CPU utilization ratio every time window (i.e., 1 minute in our setting). In this context, we can calculate the available computing capacity as the fixed CPU frequency × time window size × (1−utilization ratio). With a one-week measurement on 8 smartphones of different owners (i.e., diverse behaviors on application executions), we depict the samples as a PDF graph in Fig. 4(a).



(a) CPU utilization ratio distribution (b) Load Dynamics of data cluster

Figure 4: Real measurements for trace-driven simulations.

Functional Modules	Processing Density
Image compression (Android Bitmap)	250∼453
Audio transcoder (Android MediaCodec)	330∼860
Activity recognition (Gyro and Accelerometer)	1258∼1947
Objective recognition (Neural networks)	2250∼2374
Face recognition (OpenCV)	2339∼2566
Parking lot detection (CNN+SVM)	5000∼12800
Video transcoder (Android FFmpeg)	21500∼24483

Table I: Processing density of typical data processing modules

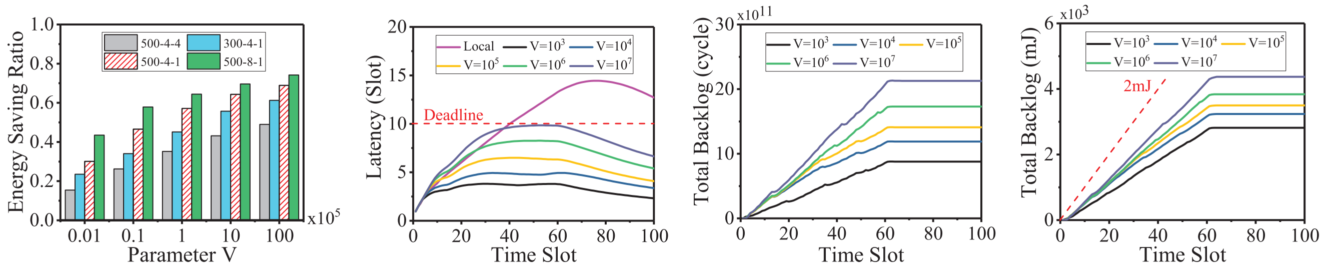
In addition, we create several Android applications including typical data processing modules and measure their processing densities. In detail, we pin the CPU frequency of smartphones to 2 GHz, process a given amount of data (e.g., images), and record the processing time. As a result, we can roughly get the processing density of each functional module by dividing the utilized CPU cycles (i.e., the CPU frequency × the sum of CPU utilization ratio during the processing time) with the input data size. Note that, the processing density of each functional module is measured with different input data sizes, and the value range is given in Table I.

To capture the dynamic VM pool resources, we perform a simple statistic in terms of the Google cluster data trace [49], which records the number of CPU cores (in fractional units) used by each task every five minutes over a 7 hour period. We simply sum up the total number of cores used by all tasks in each five-minute period as the samples (i.e., indicating the cluster load), and take all the samples in the middle 5 hours to model the load dynamics as depicted in Fig. 4(b), where each point is normalized by the maximum value in the samples.

B. Simulation Setup

We exploit the Opportunistic Network Environment (ONE) simulator [50] to create the simulation scenario. Specifically, we consider the built-in Helsinki road map, in which 500 vehicles move on the road in terms of the built-in working day movement model which has been shown to well capture the exponential distribution property of vehicle encounter frequency in many real-world mobility traces. Then, we create a “god” node to obtain the information from all vehicles as the application manager (i.e., running the *TaskSche* algorithm).

For the vehicle resources in each time slot (i.e., one minute in the simulation), we consider that all the vehicles have one CPU core with the fixed 2 GHz working CPU frequency, and their current loads are generated from the measured distribution of CPU utilization ratio in Fig. 4(a). In addition,



(a) Energy saving ratio (b) Time-average application latency (c) Resource balance queue (H) (d) Energy budget queue (Z)

Figure 5: The impact of different values of parameter V on the *Chimera* framework performance.

we adopt the similar settings from our previous work [15] to generate vehicle cellular uplink capacity (i.e., randomly choose from 1 M to 5 M) and V2V link capacity (i.e., a function of V2V distance based on the Shannon equation). Besides, we respectively set the CPU, cellular and V2V power to 1.6, 0.6 and 0.2 mW, according to various power-profiler.xml files in Android smartphones and related works [17], [18], [22]. For the VM pool resources, we leverage c4.large plan (with 3.0 GHz and 4 vCPUs) in Amazon EC2 as the computing capacity of each application VM, and set the total physical resources to 20 vCPUs with the load dynamics in Fig. 4(b). In this setting, if the current load is lower than 80%, the available vCPUs are more than $20{\times}(1-80\%) = 4$ (i.e., at least one application VM can be activated). For the vehicle incentive, we set $\alpha = 0.5$ and $K^o = 0$ mJ across all vehicles.

To begin with, we randomly choose 100 vehicles to run three applications (i.e., Audio transcoder, Face recognition and Parking lot detection) with different processing density level (i.e., 400, 2500 and 10000). For each application, the deadline $T = 10$ time slots, the arrival data size is randomly generated with the average size $\Psi = 10^5$ bits, and the task output ratio $\rho = 0.2$. In addition, all the selected vehicles will generate new data for each application in the first 50 time slots and the simulation expires when all the generated data is processed and uploaded entirely. Besides, we set the application weight $\omega = 1$ for each application. As for the performance metric, we introduce the *Energy Saving Ratio*, which equals to $\frac{E_{Loc}-E_{Alg}}{E_{Loc}}$. Here, E_{Loc} is the energy cost achieved by vehicle local processing, where we assume each vehicle schedules each application to utilize vehicle resources proportionally (i.e., $\frac{Q_i^k(t)\gamma_k}{\sum_k Q_i^k(t)\gamma_k} s_i(t)$ for computation resources and $\frac{D_i^k(t)}{\sum_k D_i^k(t)} d_i(t)$ for network resources). E_{Alg} is the energy cost achieved by a comparable task scheduling algorithm (e.g., our *TaskSche* algorithm).

C. Simulation Results

Impact of different V on framework performance: We first discuss the framework performance in terms of different values of the tradeoff parameter V. In the simulation, we take V's value from 10^3 to 10^7 into account, in terms of the normalized backlog of queue Q, D, L, A and the expression of factor c, b, a as mentioned in Section V-A. For example, since the order of magnitude of vehicle available processing capacity is 10^9 and that of application processing density is 10^2-10^4, we should give V a large value to make the energy consumption (i.e., objective) and the queue backlog (i.e., constraints) be in the similar order of magnitude. Note that, our V settings guarantee the generated input data of all the three applications can be processed and uploaded entirely.

The simulation results are given in Fig. 5. Specifically, Fig. 5(a) shows that the energy saving ratio of each legend[6] experiences a fast in the beginning and a slightly increase in the end with V increasing, which is consistent with Theorem 1. When V is over 10^5, the *Chimera* framework can save more than 40% energy, which indicates its good performance. Fig. 5(b) only presents the time-average latency of the application with the highest processing density in each time slot (calculated by the left-hand-side of the constraint (9)), since that of the other two applications are no more than 3 slots in all V cases. We can see that, the bound of application latency is less than the required deadline (i.e., the red dash line) in all V cases, which indicates that the application latency constraint (9) is satisfied, and the latency bound is linear proportion to V approximately, which is also consistent with Theorem 1. Besides, our framework achieves 1.5x – 3.6x speedup at the peak value compared with vehicle local processing.

Fig. 5(c) depicts the sum of resource balance queue backlog in terms of those 100 vehicles which are chose to run applications, since the rest 400 vehicles in the simulation are resource contributors, and they intuitively do not violate the resource balance constraint (11). We can see that, the slope as well as the peak of total backlog increases with V increasing. The reason is that a large V will result in a large backlog of task workload queue Q in terms of the Lyapunov drift-plus-penalty expression, and consequently will highlight the effectiveness of V2V cooperation. Note that, although the *Chimera* framework enables the 100 chose vehicles to "free ride" some resources from the other vehicles in the short term (i.e., generating data 50 slots + application deadline 10 slots = 60 time slots), it will enforce them to complement such a resource gap in the long term.

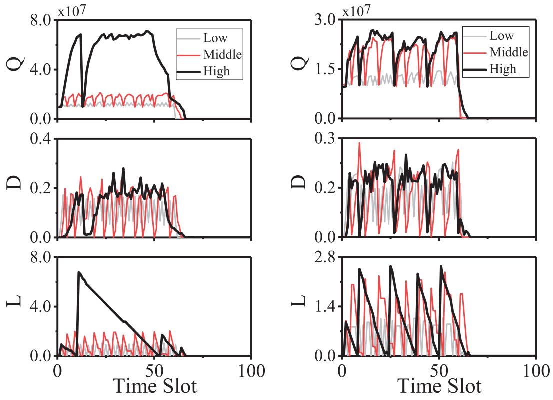
Fig. 5(d) depicts the sum of energy budget queue backlog in terms of all vehicles in the simulation. We can see that, the slope of the total backlog shares a similar trend with that

[6] The legend "500−4−1" indicates the baseline setting as mentioned in the simulation setup (i.e., 500 vehicles, 4 vCPUs for each application VM, and 1 CPU core for each vehicle).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2872436, IEEE Internet of Things Journal

in Fig. 5(c), and is smaller than that of the linear function $f(t) = 0.2t$ (i.e., the red dash line). This observation suggests that if we set $K^o = 0.2$ mJ across all vehicles, the energy budget constraint (12) can be satisfied. For example, when $t = 20$ the total energy budget is 0.2 mJ × 20 slots × 500 vehicles = 2000 mJ, which is greater than the total backlog in all V cases. Note that, it also indicates that the vehicles joining in the *Chimera* framework will not sacrifice a large amount of energy to benefit others.

Impact of different system settings on framework performance: We next discuss the framework performance in terms of different cooperative vehicle amount, VM resource amount and vehicle processing capacity. The simulation results are depicted in Fig. 5(a), where the legend "300−4−1" indicates there are 300 vehicles (also 100 chose vehicles) in the simulation, "500−8−1" indicates the computation capacity of each application VM is 8 vCPUs and the total physical resources is 40 vCPUs, and "500−4−4" indicates all the vehicles have 4 CPU cores. As we can see that, the framework performance is directly proportional to the cooperative vehicle amount and cloudlet resource amount. This phenomenon indicates that both V2V-based computing and VM-based computing play a critical role in our framework. That is, combining them together in our framework is meaningful. In addition, in the context of the improved vehicle processing capacity, our framework can still save more than 30% energy when V is over 10^5, and this phenomenon indicates that our framework always achieves a positive effect on the sensed data processing for vehicular crowdsensing applications.

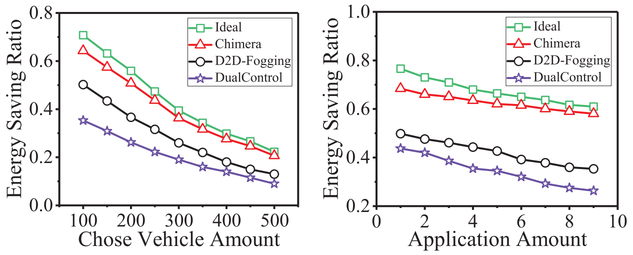


(a) Queue length (ω=1:1:1) (b) Queue length (ω=1:1:2)

Figure 6: The impact of processing densities and application weights.

Impact of different processing densities and application weights on application performance: Fig. 6(a) presents the queue length of three individual applications at the vehicle side (i.e., Q and D) and the VM side (i.e., L) when $V = 10^6$. We can see that, the application with the highest processing density (denoted by "High") accumulates more unprocessed data at the queue Q and L. The reasons are two-fold. First, processing its data at the vehicle side will consume a lot of energy in terms of the expression of factor c, and hence the framework prefers to offload its data to the VM side for energy saving. Second, in terms of the expression of factor e, only when its queue length reaches a high level, the framework will schedule resources to its VM. Indeed, we can find that the peak value of the "High" application is roughly 4 times larger than that of the "Middle" application in the queue Q and L, which is similar to the ratio of their processing densities (i.e., 10000:2500=4). For the application with the lowest processing density, it will be scheduled to utilize both vehicle resources and VM resources, and therefore it can be processed more quickly. In other words, our framework implicitly introduces "priorities" to the crowdsensing applications in terms of their processing densities, and it should complement those applications with high processing densities such as increasing their application weights to achieve fairness.

In order to study the impact of different application weights on application performance, we change the weight of the "High" application to 2, and conduct the simulation again to generate the results as shown in Fig. 6(b). We can see that, the queue length of the "High" application is significantly reduced and achieves the similar trend with that of the "Middle" application at both vehicle and VM side. This is because that increasing the application weight not only speeds up its scheduling at vehicle side, but also increases the "importance" of its corresponding VM to obtain VM pool resources. This observation justifies that by increasing the weights of the applications with high processing densities properly, multiple crowdsensing applications can achieve fairness in our framework. In addition, we should emphasize that, increasing the weights of some applications apparently reduces their time-average application latencies, while has little influence on the overall energy saving ratio.



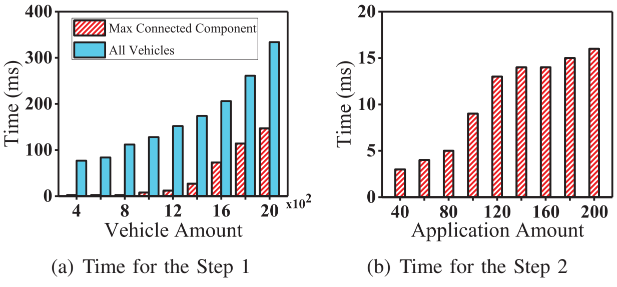
(a) Different chose vehicle amount (b) Different application amount

Figure 7: Performance comparisons among different frameworks.

Performance comparisons: We compare our framework with two state-of-the-art works: (1) *DualControl* [14], a user-operator cooperative task scheduling policy where the user adjusts its CPU speed and the operator allocates the VM pool resources to adaptive user VM (i.e., user-oriented) optimally, in order to maximize the cooperative utility. In the simulation, we substitute the single task workload queue in this work with the processing cycle queue (i.e., the input workloads×processing density) to tolerate heterogeneous applications, set an exclusive vCPU to each vehicle as the VM plan, and allow each vehicle to select three kinds of CPU frequency (i.e., 0.5, 1 and 2 GHz). (2) *D2D-Fogging* [15], an operator-controlled binary task offloading policy which optimally selects a task helper for a task owner (i.e., V2V matching) as mentioned in Section II. Note that, we consider these comparable works since they operate by the centralized control (i.e., mobile network providers), and also support the long-term and network-wide optimization by the Lyapunov theory. Besides, we also compare our framework with the *Ideal* case as mentioned in Section VI-D, where each vehicle can establish an arbitrary number of V2V links in a time slot.

As for the simulation settings, we fix the parameter V to 10^5 and conduct the performance comparisons in terms of the number of chose vehicles (total vehicle amount is 500 and each chose vehicle runs three apps with 400, 2500 and 10000 processing density, respectively) and the number of crowdsensing applications (100 chose vehicles from 500 total vehicles and the processing density of each application is 2500). The other simulation parameters remain unchanged, for instance, we consider the physical resources of VM pool is 20 vCPUs with the load dynamics in Fig. 4(b).

The results are depicted in Fig. 7. Fig. 7(a) shows that, as the number of chose vehicle increases, all the framework performances degrade rapidly, due to the increase in the amount of overall arrival workloads and the decrease in the amount of resource-free cooperative vehicles. In addition, our framework supporting both VM-based and V2V-based computing can achieve better performance especially when the chose vehicle amount is small. For example, ours can obtain 82% – 130% performance gain compared with *DualControl*, and 28% – 58% performance gain compared with *D2D-Fogging*. The reasons are two-fold. First, our framework can make full use of resource-free cooperative vehicles to facilitate the chose vehicles. Second, our framework exploits the application-oriented model for the VM pool, which is more suitable for the crowdsensing application scenario compared with the device-oriented model in *DualControl*, since the vehicular crowdsensing campaign generally will recruit a large number of vehicles to cover large-scale areas. Fig. 7(b) shows that, as the number of application increases, all the framework performances degrade slowly, which indicates that they can effectively cope with multiple applications simultaneously. Despite of this, our performance still achieves 56% – 120% performance gain compared with *DualControl*, and 37% – 60% performance gain compared with *D2D-Fogging*. To sum up, our framework can benefit from both cooperative vehicle and VM pool resources, and thus its performance is much better.



(a) Time for the Step 1

(b) Time for the Step 2

Figure 8: The execution time of the *TaskSche* algorithm.

Framework complexity: We numerically evaluate the running time of the *TaskSche* algorithm with a large vehicle and application amount. Fig. 8 presents the results using an Intel Core i5-2400 Processor@3GHz. First, with the vehicle amount increasing, the execution time for the linear programming using the Simple method plus the minimum weighted graph matching using the Blossom algorithm (i.e., Step 1) increases greatly, and the time consumption is around 320ms for 2000 vehicles (half of them are chose to run applications). Nevertheless, if we partition the whole vehicle graph into several connected components (i.e., using Tarjan algorithm) and apply the Simple method and the Blossom algorithm for each component in parallel, then the overall execution time can reduce at least 50%. That is, our framework should adopt this divide-conquer principle to support a larger number of vehicles in practice. Second, with the application amount increasing, the execution time for the knapsack-based resource allocation solution at the VM pool (i.e., Step 2) keeps small with a slight increase (i.e., less than 16ms). To sum up, our framework scales well for a large number of vehicles and applications.

VIII. Conclusion

In this paper, we proposed *Chimera*, a novel hybrid edge computing framework, integrated with the emerging edge cloud radio access network, to augment network-wide vehicle resources for future large-scale vehicular crowdsensing applications, by leveraging a multitude of cooperative vehicles and the VM pool via the application manager control in the edge cloud. We presented a comprehensive framework model and formulated a novel multi-vehicle and multi-task offloading problem aiming at minimizing the energy consumption of network-wide recruited vehicles serving heterogeneous crowdsensing applications, meanwhile considering a series of practical constraints. We designed *TaskSche*, an online task scheduling algorithm, in which an efficient task workload assignment policy based on graph transformation and a knapsack-based VM pool resource allocation policy were devised as the core components. We also discussed some practical extensions of the framework. Rigorous theoretical analyses and extensive trace-driven simulations showed that it could save substantial energy for network-wide vehicles and scale well for a large number of vehicles and applications.

IX. Acknowledgement

This work was supported in part by the National Key Research and Development Program of China under grant No.2017YFB1001703; the National Natural Science Foundation of China (No.61702287, No.U1711265); the Natural Science Foundation of Tianjin (No.18JCQNJC00200); the Fundamental Research Funds for the Central Universities (No.7933, No.17lgjc40); the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (No.2017ZT07X355), and the Open Project Fund of Guangdong Key Laboratory of Big Data Analysis and Processing (No.2017003).

References

[1] S. Chen, J. Hu, and Y. Shi, “LTE-V: A TD-LTE-Based V2X solution for future vehicular network,” *IEEE Internet of Things Journal*, 2016.

[2] X. Cheng, C. Chen, W. Zhang, and Y. Yang, “5g-enabled cooperative intelligent vehicular (5genciv) framework: When benz meets marconi,” *IEEE Intelligent Systems*, 2017.

[3] J. Ni, X. Lin, and X. Shen, "Security, privacy, and fairness in fog-based vehicular crowdsensing," *IEEE Communications Magazine*, 2017.

[4] E. Baccarelli, N. Cordeschi *et al.*, "Energy-efficient dynamic traffic offloading and reconfiguration of networked data centers for big data stream mobile computing: review, challenges, and a case study," *IEEE Network*, 2016.

[5] H. T. Dinh, C. Lee, D. Niyato, and P. Wang, "A survey of mobile cloud computing: architecture, applications, and approaches," *WILEY Wireless Communications and Mobile Computing (WCMC)*, 2013.

[6] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys & Tutorials*, 2017.

[7] M. A. Salahuddin, A. Al-Fuqaha, and M. Guizani, "Software-defined networking for rsu clouds in support of the internet of vehicles," *IEEE Internet of Things Journal*, 2015.

[8] M. Shojafar, N. Cordeschi, and E. Baccarelli, "Energy-efficient adaptive resource management for real-time vehicular cloud services," *IEEE Transactions on Cloud Computing (TCC)*, 2016.

[9] R. Kim, H. Lim, and B. Krishnamachari, "Prefetching-based data dissemination in vehicular cloud systems," *IEEE Transactions on Vehicular Technology (TVT)*, 2016.

[10] X. Hou, Y. Li *et al.*, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Transactions on Vehicular Technology (TVT)*, 2016.

[11] Z. Jiang, S. Zhou, X. Guo, and Z. Niu, "Task replication for deadline-constrained vehicular cloud computing: Optimal policy, performance analysis and implications on road traffic," *IEEE Internet of Things Journal*, 2017.

[12] M. Sookhak, F. R. Yu *et al.*, "Fog vehicular computing: Augmentation of fog computing using vehicular cloud computing," *IEEE Vehicular Technology Magazine*, 2017.

[13] X. Cheng, L. Yang, and X. Shen, "D2d for intelligent transportation systems: A feasibility study," *IEEE Transactions on Intelligent Transportation Systems*, 2015.

[14] Y. Kim, J. Kwak, and S. Chong, "Dual-side optimization for cost-delay tradeoff in mobile edge computing," *IEEE Transactions on Vehicular Technology (TVT)*, 2018.

[15] L. Pu, X. Chen, J. Xu, and X. Fu, "D2d fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2016.

[16] W. Zhang, Y. Wen *et al.*, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Transactions on Wireless Communications (TWC)*, 2013.

[17] J. Kwak, Y. Kim, J. Lee, and S. Chong, "Dream: Dynamic resource and task allocation for energy minimization in mobile cloud systems," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2015.

[18] Z. Jiang and S. Mao, "Energy delay tradeoff in cloud offloading for multi-core mobile devices," *IEEE Access*, 2015.

[19] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Transactions on Networking (ToN)*, 2016.

[20] M.-H. Chen, M. Dong, and B. Liang, "Multi-user mobile cloud offloading game with computing access point," in *IEEE International Conference on Cloud Networking (Cloudnet)*, 2016.

[21] V. Cardellini, V. D. N. Personé *et al.*, "A game-theoretic approach to computation offloading in mobile cloud computing," *Springer Mathematical Programming*, 2016.

[22] Y. Mao, J. Zhang, S. Song, and K. B. Letaief, "Power-delay tradeoff in multi-user mobile-edge computing systems," in *IEEE Global Communications Conference (Globecom)*, 2016.

[23] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Transactions on Wireless Communications (TWC)*, 2017.

[24] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading decision and resource allocation for multi-user multi-task mobile cloud," in *IEEE International Conference on Communications (ICC)*, 2016.

[25] S. Guo, B. Xiao, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[26] C. Shi, V. Lakafosis, M. Ammar, and E. Zegura, "Serendipity: enabling remote computing among intermittently connected mobile devices," in *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*, 2012.

[27] M. Xiao, J. Wu *et al.*, "Multi-task assignment for crowdsensing in mobile social networks," in *IEEE Conference on Computer Communications (INFOCOM)*, 2015.

[28] L. Yang, J. Cao, H. Cheng, and Y. Ji, "Multi-user computation partitioning for latency sensitive mobile cloud applications," *IEEE Transactions on Computers (TOC)*, 2015.

[29] X. Lyu, H. Tian, C. Sengul, and P. Zhang, "Multiuser joint task offloading and resource optimization in proximate clouds," *IEEE Transactions on Vehicular Technology (TVT)*, 2017.

[30] C. You, K. Huang, and H. Chae, "Energy efficient mobile cloud computing powered by wireless energy transfer," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2016.

[31] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2016.

[32] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Transactions on Signal and Information Processing over Networks (TSIPN)*, 2015.

[33] R. Urgaonkar, S. Wang *et al.*, "Dynamic service migration and workload scheduling in edge-clouds," *Elsevier Performance Evaluation*, 2015.

[34] R. Zhang, X. Cheng, L. Yang *et al.*, "A novel centralized tdma-based scheduling protocol for vehicular networks," *IEEE Transactions on Intelligent Transportation Systems*, 2015.

[35] X. Shen, X. Cheng, L. Yang, R. Zhang, and B. Jiao, "Data dissemination in vanets: A scheduling approach," *IEEE Transactions on Intelligent Transportation Systems*, 2014.

[36] Z. Sheng, C. Mahapatra *et al.*, "Energy efficient cooperative computing in mobile wireless sensor networks," *IEEE Transactions on Cloud Computing (TCC)*, 2015.

[37] Y. Cui, J. Song *et al.*, "Software defined cooperative offloading for mobile cloudlets," *IEEE/ACM Transactions on Networking (ToN)*, 2017.

[38] X. Liu, Y. Li, and H.-H. Chen, "Wireless resource scheduling based on backoff for multiuser multiservice mobile cloud computing," *IEEE Transactions on Vehicular Technology (TVT)*, 2016.

[39] W. Liu, X. Wen *et al.*, "Efficient computation offloading for various tasks of multiple users in mobile edge clouds," in *Springer International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP)*, 2017.

[40] X. Chen and J. Zhang, "When d2d meets cloud: Hybrid mobile task offloadings in fog computing," in *IEEE International Conference on Communications (ICC)*, 2017.

[41] J. M. Kim, Y. G. Kim, and S. W. Chung, "Stabilizing cpu frequency and voltage for temperature-aware dvfs in mobile devices," *IEEE Transactions on Computers (TC)*, 2015.

[42] F. Zeng, R. Zhang, X. Cheng, and L. Yang, "Channel prediction based scheduling for data dissemination in vanets," *IEEE Communications Letters*, 2017.

[43] D. W. K. Ng and R. Schober, "Resource allocation and scheduling in multi-cell ofdma systems with decode-and-forward relaying," *IEEE Transactions on Wireless Communications (TWC)*, 2011.

[44] X. Cheng, C. X. Wang, H. Aggoune, and H. Aggoune, "Envelope level crossing rate and average fade duration of nonisotropic vehicle-to-vehicle ricean fading channels," *IEEE Transactions on Intelligent Transportation Systems*, 2014.

[45] J. D. Little and S. C. Graves, "Little's law," in *Springer Building intuition*, 2008, pp. 81–100.

[46] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[47] "Online technical report," Available in: https://www.dropbox.com/s/8j5mo768dp8tcky/technical%20report.pdf?dl=0.

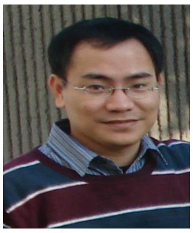
[48] "Anothermonitor," Available in: https://github.com/AntonioRedondo/AnotherMonitor.

[49] "Google cluster data trace," Available in: https://github.com/google/cluster-data/blob/master/TraceVersion1.md.

[50] "Opportunistic network environment (one) simulator," Available in: https://www.netlab.tkk.fi/tutkimus/dtn/theone.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2018.2872436, IEEE Internet of Things Journal

Lingjun Pu received the Ph.D. degree from Nankai University, China, in 2016. He was a joint Ph.D. student with University of Göttingen, Germany, from 2013 to 2015. He is currently an Assistant Professor with the College of Computer Science, Nankai University, China. His research focuses on mobile cloud/edge computing, edge caching, IoT, 5G C-RAN, SDN/NFV and opportunistic routing.



Xu Chen (M'12) is a Full Professor with Sun Yat-sen University, Guangzhou, China, and the vice director of National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He received the Ph.D. degree in information engineering from the Chinese University of Hong Kong in 2012, and worked as a Postdoctoral Research Associate at Arizona State University, Tempe, USA from 2012 to 2014, and a Humboldt Scholar Fellow at Institute of Computer Science of University of Goettingen, Germany from 2014 to 2016. He received the prestigious Humboldt research fellowship awarded by Alexander von Humboldt Foundation of Germany, 2014 Hong Kong Young Scientist Runner-up Award, 2016 Thousand Talents Plan Award for Young Professionals of China, 2017 IEEE Communication Society Asia-Pacific Outstanding Young Researcher Award, 2017 IEEE ComSoc Young Professional Best Paper Award, Best Paper Runner-up Award of 2014 IEEE International Conference on Computer Communications (INFOCOM), and Best Paper Award of 2017 IEEE Intranational Conference on Communications (ICC). He is currently an Associate Editor of IEEE Internet of Things Journal and IEEE Journal on Selected Areas in Communications (JSAC) Series on Network Softwarization and Enablers.

Guoqiang Mao (S'98-M'02-SM'08-F'18) joined the University of Technology Sydney in February 2014 as Professor of Wireless Networking. Before that, he was with the School of Electrical and Information Engineering, the University of Sydney. He has published about 200 papers in international conferences and journals, which have been cited more than 6000 times. He is an editor of the IEEE Transactions on Wireless Communications (since 2014), IEEE Transactions on Vehicular Technology (since 2010) and received "Top Editor" award for outstanding contributions to the IEEE Transactions on Vehicular Technology in 2011, 2014 and 2015. He is a co-chair of IEEE Intelligent Transport Systems Society Technical Committee on Communication Networks. He has served as a chair, co-chair and TPC member in a number of international conferences. He is a Fellow of IET. His research interest includes intelligent transport systems, applied graph theory and its applications in telecommunications, Internet of Things, wireless sensor networks, wireless localization techniques and network performance analysis.

Qinyi Xie is currently a Master student with the College of Computer Science, Nankai University, China. Her research interests include Low-Power Wide-Area Networks (LPWAN), mobile edge computing and 5G C-RAN.

Jingdong Xu is currently a Professor with the College of Computer Science, Nankai University, China. She is also the Head of the Computer Networks and Information Security Group. Her research interests include sensor networks, vehicle ad-hoc networks, network security and management, and opportunistic network and computing.