Offloading Autonomous Driving Services via Edge Computing

Mingyue Cui, Shipeng Zhong, Boyang Li, Xu Chen, *Member, IEEE*, and Kai Huang, *Member, IEEE*

***Abstract*—A key challenge for autonomous driving is to process a massive amount of sensor data and make safe and reliable decisions in real time. However, autonomous vehicles often have insufficient onboard resources to provide the required computation capacity. To address this problem, this article advocates a novel approach to offload computation-intensive autonomous driving services to roadside units and cloud for swift executions. Our approach combines an integer linear programming (ILP) formulation for offline optimization of the scheduling strategy and a fast heuristics algorithm for online adaptation. We verify our technique with both synthetic task graphs and real-world deployment. The experimental results show that our approach can improve system performance effectively.**

***Index Terms*—Autonomous driving, computation offloading, edge computing, Internet of Things (IoT), simultaneous localization and mapping (SLAM).**

I. Introduction

AUTONOMOUS driving systems are required to achieve guaranteed performance and safety through intensive sensing from various sensors, such as cameras, radar, and LiDAR [1]–[3]. Such a large amount of sensor data requires a huge amount of computing power to process in real time. To cope with this computing demand, state-of-the-art onboard units (OBUs) integrate expensive computing units. For example, the Nvidia Drive PX 2, used by Audi Q7 and Tesla Model X sells for $15 000 [4]. Even the Mobileye's EyeQ4 costs $1500 [5]. The price of OBU limits a wider range of applications of autonomous driving.

To reduce the computing demand on OBU, the summary of the 2007 DARPA Urban Challenge [6] called for a hybrid mechanism for autonomous driving, i.e., to share information among vehicles, roadside infrastructure, and cloud. For example, Kumar *et al.* [7] proposed a computational model based on vehicular cloud computing (VCC) [8]. This model builds a vehicle computing resource cloud by using the processing and storage capabilities of nearby vehicles and assigns some computation tasks to the cloud to reduce latency. Due to the long and unpredictable latency between OBU and cloud, combining only OBU and cloud fails to serve as a solid solution for autonomous driving.

The emerging edge-computing paradigm, which computes programs in the edge zone in the proximity of the data source, can be much more promising. Compared to VCC, edge computing can provide shorter response time, better reliability, and less overall bandwidth. There is related work [9], [10] in the literature that utilizes mobile-edge computing (MEC) [11], [12] and offloads vehicular computation tasks to shared resources on edge servers. They focus on the performance of cellular networks and the value of multiaccess edge computing for supporting multiple services. Nevertheless, how to offload vehicle computation tasks and deploy them in real scenarios has not been solved.

Sharing workload among OBU, edge, and cloud is not straightforward. First, finding which portions of the workload to offload to edge or cloud is not obvious. Although offloading more tasks to edge and cloud will result in lighter weight OBU, more sensor data need to be sent out from OBU and more result data need to be sent back, consequently requiring more bandwidth and having longer latency. Second, the network status may not always be stable and subsequently, the bandwidth between the OBU, edge, and cloud may change over time, resulting in fluctuation on the latency of the transmitting data. The asynchronous data between computation tasks may thereof miss their deadlines and consequently lead to functional errors, e.g., fail to localize the current position of the vehicle and miss the braking point. Last but not least, the schedule needs to be lightweight, simple, and practical, so as to reduce any additional overhead as much as possible. Therefore, a sophisticated method is needed to guarantee QoS by elastic offloading of tasks.

This article proposes an approach to offloading vehicular computation services on a three-layer vehicular-edge-cloud structure. Our approach consists of an offline scheduling strategy and an online dynamic adaptive (ODA) scheduling algorithm. Given computation services modeled as a direct acyclic graph (DAG), the offline scheduling strategies are computed by an integer linear programming (ILP) model to obtain the optimal solutions for a set of given network parameters. The ODA scheduling algorithm takes the offline scheduling strategies as a basis and computes a new offloading strategy for online use, taking the actual network status into consideration.

Manuscript received October 28, 2019; revised March 25, 2020 and May 1, 2020; accepted May 29, 2020. Date of publication June 9, 2020; date of current version October 9, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1802405, in part by the Science and Technology Planning Project of Guangzhou, China, under Grant 202007050004, in part by the National Science Foundation of China under Grant 61872393, in part by the Scholarship from 19lgpy229, Fundamental Research Funds for the Central Universities, and in part by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams under Grant 2017ZT07X355. *(Corresponding author: Kai Huang.)*

The authors are with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 400100, China (e-mail: cuimy@mail2.sysu.edu.cn; zhongshp5@mail2.sysu.edu.cn; liby3@mail2.sysu.edu.cn; chenxu35@mail.sysu.edu.cn; huangk36@mail.sysu.edu.cn).

Digital Object Identifier 10.1109/JIOT.2020.3001218

To demonstrate the effectiveness of our approach, extensive evaluations with synthetic tasks and real-world deployment are conducted. The case study in real scenarios shows that: compared with using OBU only: 1) the average latency of localization for autonomous vehicles based on edge computing is reduced by 34% and 2) the average accuracy of lateral and longitudinal localization is improved about 5.7 *times* and 7.6 *times*, respectively. For worst case accuracy, the localization error is increased by about 15 *times* in lateral and 14 *times* in longitudinal. The detailed contributions of this article are as follows.

1) We investigate the possibility of low-cost electronics for autonomous driving while achieving QoS. For offloading autonomous driving services on a three-layer Internet structure, an ILP formulation is developed for the latency minimization problem and an efficient lightweight heuristic algorithm is designed to online cope with the network fluctuation.
2) A distributed LiDAR simultaneous localization and mapping (SLAM) algorithm is designed, enabling task-level parallelism, remote data exchange via the cellular network, and point cloud (de)compression. In this manner, the algorithm can be cooperatively deployed on both OBU, edge, and cloud.
3) We verified our technique with both simulations on synthetic task graphs and realistic deployment. We used the combination of inertial measurement unit (IMU) mileage estimation and real-time kinematic (RTK) to construct the environmental map for experimental LiDAR SLAM verification. Results illustrate that our approach can shorten the response time of computation tasks for autonomous driving and achieve high-quality performance.

The remainder of this article is organized as follows. Section II reviews the related work. In Section III, we present the architecture of system and data transmission problems. Our system models are introduced in Section IV. Then, we perform schedulability analysis and get constraints that will be integrated into our formulation approach in Section V. Section VI presents and analyzes the experimental simulation and its results. In Section VII, we verify the feasibility of our approach by conducting a case study. Finally, Section VIII concludes this article.

II. Related Work

In order to effectively reduce the system response time, Borrmann *et al.* [15] proposed a runtime resource management in a heterogeneous system for autonomous driving. Their approach can effectively reduce the idle time of the classic frame-slotted schedule. The limitation is that only local resources can be used and sometimes that might not be enough. Jiang *et al.* [16] proposed a task offloading model based on VCC. This model effectively solves the problem of successful transmission between vehicles and infrastructures during encounters but does not take into account the limitations of the computation resources for onboard units of vehicles.

In recent years, many proofs of concepts [17]–[19] have been made to incorporate edge computing into the vehicular computation. Mach and Becvar [20] divided the edge-computing problem of computation offloading to three key areas: 1) decision on computation offloading; 2) allocation of computation resource within the MEC; and 3) mobility management. Zhang *et al.* [21] provided a theoretical framework of energy-optimal mobile cloud computing under a stochastic wireless channel. They derived an optimal threshold policy for computation offloading decisions, i.e., mobile execution or cloud execution. The study [22] implemented dynamic voltage scaling techniques with computation offloading for different objectives. Hao *et al.* [23] proposed a scheme to optimize MEC by caching task data at the edge. This approach reduces the total energy consumed by mobile devices in the MEC. All these works generally focused on energy conservation, which is not suitable for offloading low-latency high-safety tasks.

Chen *et al.* [24] proposed a game-theoretic approach for the computation offloading decision among multiple mobile device users for mobile-edge cloud computing. Fan *et al.* [25] proposed a method based on the cooperation of multiple MEC enabled (MEC-BCs) to enhance the computing offload service of the MEC. However, the former focuses on load balancing among multiple users, while the latter is the cooperative communication mechanism of multiple MECs. Mao *et al.* [26] jointly optimized the task scheduling and wireless power allocation in a single-user single-core MEC system, which reduces the execution latency and device energy consumption. Shahzad and Szymanski [27] used the "dynamic programming with Hamming distance termination" method to offload tasks and reduced the energy use of the mobile device. But it is more used for noncritical tasks, not suitable for sensor-driven autonomous driving services. Many research works [28]–[30] are used to solve the problem of tasks offloading with edge computing or cloud, but they often bring unacceptable computing costs for cheap semiconductor computing devices.

Up till today, there have not been many thorough prior studies on applying edge computing to low-cost OBU for autonomous vehicles while fulfilling rigorous requirements of low latency, especially evaluated with real-life experiments on the vehicle, which prompted this article. A recent document [31] proposed a complete edge-computing framework for autonomous robots and vehicles. However, it lacks experimental data and rigorous verification. As an initial thrust, in this article, we only focus on how to provide better QoS for a single autonomous vehicle with edge computing, not considering the impact of multivehicles.

Multivehicle scenario based on edge computing will be more complex and diverse than the single autonomous vehicle. There are many questions that need to be considered. For instance, whether coordinated resource management and load management are needed, if yes how. How to make full use of network bandwidth with enormous amounts of raw data sent by multiple vehicles. In the case of SLAM, cooperative SLAM itself is currently a hot topic in the field.

III. Background for Autonomous Driving

In order to consider offloading autonomous driving services, we first present the classical autonomous driving solutions

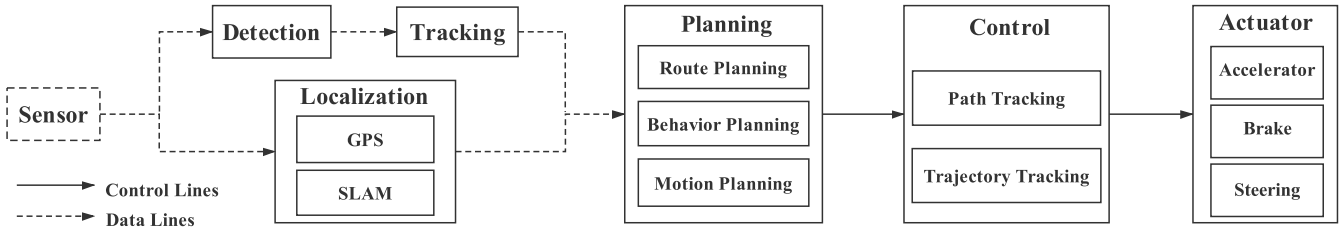


Fig. 1. Overview of the general autonomous driving system [13], [14], which achieves autonomous navigation through acquiring sensor data and a series of algorithms.

in the industry and analyze their performance constraints. In addition, the network architecture used in this article will be introduced to facilitate the subsequent mathematical modeling.

A. Autonomous Driving Pipeline

Autonomous vehicles sense the surrounding environment by carrying various sensors and classify various objects, such as nonmotor vehicles and motor vehicles. Subsequently, it is necessary to track the identified objects and keep an eye on the moving objects on the road. Additionally, the vehicle needs sensor data to complete its own high-precision positioning. Perception and localization information is fused as input, which is transformed into the same 3-D coordinate space for transmission to the planning module. After route planning, behavior planning, and motion planning, the planning module generates a series of sequential point path information from the starting position to the end position. The path information generated by the planning module is used for vehicle self-control to meet the needs of various vehicle behaviors. Finally, the control instructions are sent to the acceleration, braking, and steering wheel components of the car operating system (car-OS) for action. The software structure of the whole system is shown in Fig. 1.

According to industry standards and design specifications recently issued by Mobileye [32] and Udacity [33], autonomous driving systems should be able to sense traffic conditions in real time and respond quickly enough. As Lin *et al.* mentioned in [34], the latency of autonomous driving systems handling traffic conditions should be less than 100 ms, faster than manual driving latency. The limited computing capacity of OBUs of commercial vehicles imposes high challenges to fulfill this latency requirement. One of the solutions may be to offload computation tasks by means of edge and cloud computing.

B. Three-Tier Network Architecture

As shown in Fig. 2, we introduce an edge-computing architecture for offloading autonomous driving tasks, which includes OBU, MEC server, and cloud. OBU mounted on autonomous vehicles is connected to MEC servers and cloud by wireless edge network (WEN) and cellular network connections through the wireless base station (BS), respectively. Data transmission between MEC servers and cloud is carried out by the wired carrier backbone network. The introduction of each platform in the three-tier network architecture lists as follows.

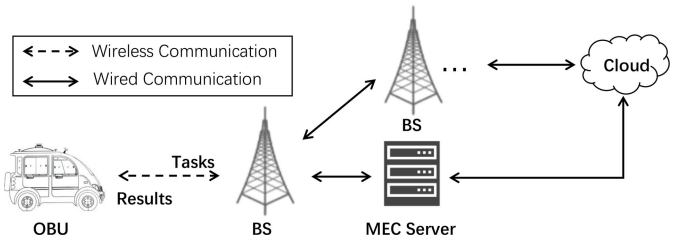


Fig. 2. Network architecture with tasks offloading based on edge computing.

1) *Autonomous Driving System Network:* It is the basis of the framework. The vehicle itself has a certain local processing capability and can offload computation tasks to the corresponding MEC servers and cloud. It connects to the WEN and cloud, and efficiently completes computation tasks according to available resources, current bandwidth, and computing capacity requirements.
2) *WEN:* It is a distributed open platform composed of a series of scattered computing units, data storage devices, network devices, and so on. Because it is closer to users or data sources, it has high bandwidth and low latency. When the vehicle is in the network scope, the computation task is assigned to the edge server to achieve faster response time and more accurate results and to alleviate the pressure on local computing and storage.
3) *Cloud Platform:* It is composed of several interconnected virtual machines (VMs) or containers that can provide computing and storage resources. By using the Platform-as-a-Service (PaaS) or Containers-as-a-Service (CaaS) model, autonomous driving systems, and edge servers can access the cloud. In addition to network routing and resource information, the configuration information of edge servers is also stored in the cloud.

C. Data Transmission of Point Cloud

Network bandwidth is a challenge to the real-time transmission of LiDAR point cloud data. Taking Velodyne VLP-16 LiDAR as an example, point cloud data with an average size of 426 kB are generated at the rate of 10 Hz. WiFi networks can only often reach 5–100 Mb/s [35], while uplink-constrained cellular networks can only maintain 1–10 Mb/s [36], [37] due to complicated Internet surroundings. In addition, there are uncertainties when the robot operating system (ROS) is used to process large persistent streams, and the data synchronization between the sender and the receiver cannot be guaranteed, as

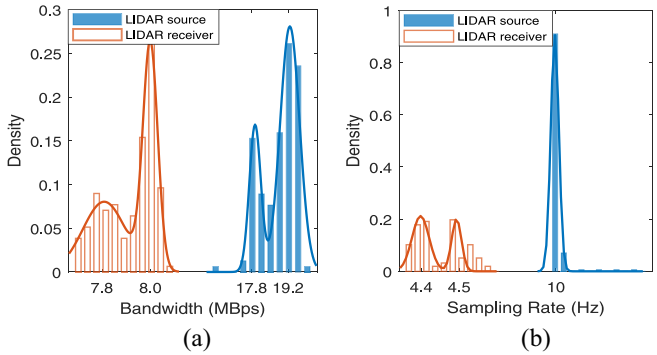


Fig. 3. Comparison of data rates between the sender and the receiver. (a) Data transmission bandwidth. (b) Data transmission frequency.

also mentioned in [38]. Here is an experiment we conducted. The NVIDIA Jetson TX2 and a server with a processor of Intel Xeon Platinum 8163 are connected through WiFi connection by using the Xiaomi 4A router which supports the IEEE 802.11n protocol. Then, we measured the effective data rate when sending raw data of the Velodyne HDL-32E LiDAR. The results are shown in Fig. 3. At the source, we measured a median 18.76 MB/s data rate at 9.99 Hz, as expected from our measured average data size of 1.70 MB at 10 Hz. However, at the receiver, we measured a median date rate of 8.01 MB/s (less than half of the sender's) with a received sampling frequency of only 4.43 Hz. In our experiments, we can observe that there are severe latency, message loss, and network interruption in point cloud data transmission based on ROS. The lack of real-time support of ROS may be one of the reasons. Therefore, it is necessary for point cloud data to en/decode before transmission.

Draco [39] is an open-source library developed by Google for compressing and decompressing 3-D geometric meshes and point clouds. The overall idea is to encode and store grid connection information and geometric information separately. We modified it as an ROS node and tested it with the Stanford 3-D scanning repository [40]. The compression effect can reach one-tenth of the original data, and the peak signal-to-noise ratio (PSNR)[1] obtained by the *pc_error* MPEG tool can be maintained at about 31. With data quality guaranteed, Draco can significantly improve the performance of the point cloud data transmission.

IV. Models and Problem Statements

In this section, we propose several models and define them, then provide the problem formulation. The constants and variables used are listed in Table I.

A. System Model

The autonomous driving system with access to a set $P = \{V,E,C\}$ of three platforms, i.e., OBU, edge server, and cloud, respectively. Computational devices on different platforms can provide different performances. The capability of OBU on vehicles is denoted as F^V, (i.e., CPU cycles per second). Similarly, the cloud and each edge server offer F^C and F^E of computation capability, respectively. As the vehicle runs on the road, autonomous driving tasks are offloaded to the nearby edge server, cloud, or a combination of both through cellular connection with some BSs. In the model, we assume cloud providers can offer consistent, sufficient, and reliable computation resources, and all available edge servers during each ride or trip are deployed with the same configuration and availability.

[1]The point-to-point geometry PSNR [41] is an objective criterion for evaluating image quality, which is used to measure the degree of change between the processed image and the original image.

TABLE I
Constants and Variables

Constants	Explaination
P	the set of three platforms
G	task graph
T	the set of all tasks in DAG G
E	the set of all edges in DAG G
F^p	computation capacity on platform p
t_i	task t_i
ω_i	computation resource used to execute task t_i
δ_i	deadline of execution of task t_i
τ_i^p	execution time of task t_i if it is executed on platform p
e_{ij}	indicating that task t_j needs output from t_i
c_{ij}	communiation task of task t_i and task t_j
ϵ_{ij}	the size of data need to be transmitted in c_{ij}
τ_{ij}	execution time of computation task c_{ij}
$\varphi_{u/d}$	the upload/download wireless data transmission rate
Variables	**Explanation**
x_i^p	indicating that task t_i is executed on platform p
r_i	release time of execution of task t_i
r_{ij}	release time of communication task c_{ij}
f_i	terminal time of execution of task t_i

B. Task Model

Most of the nonfeedback autonomous driving services can be modeled as periodic-dependent tasks. These periodic-dependent tasks are represented by a DAG, $G = (T, E)$. $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of $|T|$ individual computation tasks. We use a tuple $t = (id, \omega, \delta)$ to represent each computation task, where id is the identifier of the task, ω is the amount of computation resources needed to execute task t, and δ is the deadline of the task. The set of directed edges $E \subseteq T \times T$ defines data dependency relations of all tasks in T. Each directed edge e_{ij} represents the existence of data dependency between tasks t_i and t_j, i.e., the result of executing t_i is used for execution of t_j in the same period. The data size is denoted by ϵ_{ij} (in bits). It is worthwhile noting that no loops exist in DAG. Meanwhile, no mutual data dependencies exist in the graph, i.e., $\forall e_{ij},\ e_{ij} \in E \implies e_{ji} \notin E$.

There is an example shown in Fig. 4, which is the execution model and scheduling strategy of the SLAM algorithm. In Fig. 4 (a), the computation tasks of SLAM are modeled by a DAG. In the DAG, there are six nodes and five directed edges. Each node denotes one computation task and every directed edge is associated with a communication task. The average car already has more than 150 million lines of code, according to a 2018 KPMG report [42], including six major functional modules, such as object detection, object tracking, localization, etc. The division of tasks is not necessarily the same. Here, we divide the task according to the functionality.

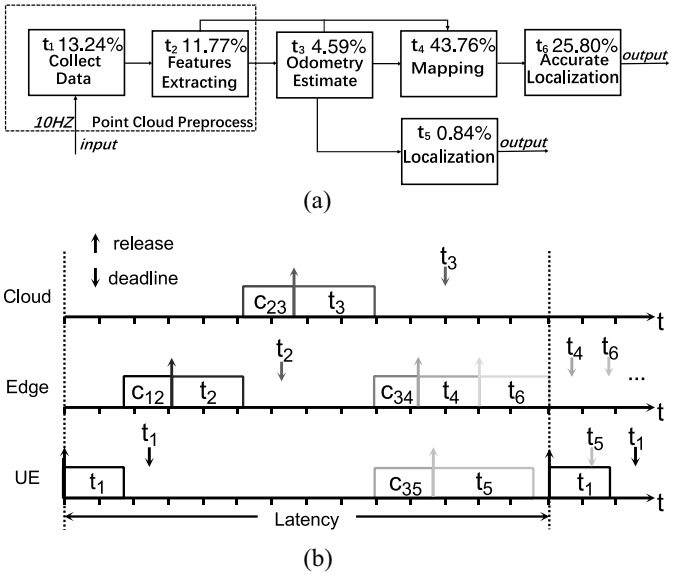


Fig. 4. Execution model and scheduling strategy for the SLAM algorithm. (a) SLAM pipeline. (b) Possible schedule of computation tasks.

Usually, the task t_i is allocated to the platform which gives the maximum gain, i.e., the minimal cost of execution time and more accurate results. The total time of t_i is composed of execution time and communication cost. For a task $t_i,\ i = 1, 2, \ldots, n$, let the set $X_i = \{x_i^p\}$ be the allocation of platforms when t_i is executed. x_i^p is a binary variable representing whether t_i is executed on platform p, namely, x_i^V, x_i^E, and x_i^C, or respectively, OBU, edge, and cloud. Possible values for each x_i are

$$x_i^p = \begin{cases} 1, \text{ if task } t_i \text{ is executed on platform } p, \\ 0, \text{ otherwise} \end{cases} \tag{1}$$

where $p \in P$.

C. Computation and Communication Model

Most of the computation services for autonomous driving are driven by sensors on the car body, which provide initial data input for independent algorithms. For each task t_i in the task graph G, its complete lifespan starts from the start of the earliest transmission of its input to the finishing of execution. The execution time of a task on the vehicle can be defined as $\tau_i^V = (\omega_i / F^V)$. Similarly, the execution time of a task on the edge server is $\tau_i^E = (\omega_i / F^E)$, assuming edge servers with an identical configuration in a given period of the task graph. The case on the cloud can be defined as $\tau_i^C = (\omega_i / F^C)$.

We employ a typical MEC deployment, where the worst case network bandwidth of both uplink and downlink is reliably provided by lower level services and can be queried from MEC APIs. In this deployment, it considered a standard operation to query bandwidth information and request sensible bandwidth occupation. For our model, the uplink data rate φ_u and the downlink data rate φ_d are given from MEC APIs. During the prologue, data may need to be offloaded to edge or cloud and it may also be downloaded to the vehicle. Each data dependency relation e_{ij} between tasks $t_i,\ t_j \in T$ has a one-to-one counterpart, defined by communication task c_{ij}. It will take $\tau_{ij} = (\epsilon_{ij} / \varphi_{u/d})$ of time to offload autonomous driving services (i.e., upload tasks to the edge and download back to vehicle).

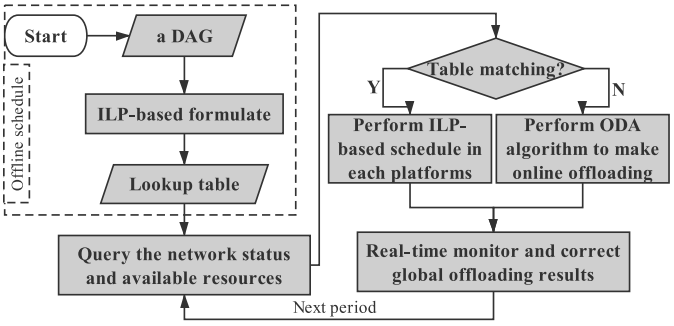


Fig. 5. Overview of our approach.

D. Static Schedule

A static schedule of a given DAG denotes a repeated pattern of executing computation tasks in a single period. It contains control step assignments (i.e., when each task starts) and computation tasks' allocations (i.e., where each task is performed), with compliance to all data dependencies in the DAG. τ_{ij} is determined by the assignment of tasks t_i and t_j. If t_i and t_j are executed on the same side, we assume the execution time of the computation task c_{ij} equals *zero*. In Fig. 4(b), we give a possible schedule for the DAG, which is repeatedly executed in each period. The earliest deadline first (EDF) strategy is adopted for the schedule of tasks that have no data dependencies and are executed on the same platform. The sequence of task execution is sorted in the ascending order of the difference between the task deadline and the arrival of the new request.

E. Problem Statement

Given autonomous driving services modeled as a DAG $G = (T, E)$, three platforms, including OBU on the vehicle, edge, and cloud, we will study the problem: what is the minimum response time for autonomous driving services to guarantee all computation and communication tasks schedulable?

V. Our Approach

In this section, we introduce our framework of offloading autonomous driving services via edge computing. Then, we perform schedulability analysis and modeling for computation tasks.

A. Framework

Fig. 5 presents an overview of our approach, which consists of an offline ILP solution and a fast heuristic for online tuning. The reasons for such a hybrid approach are twofold. On the one hand, although the ILP solution can provide the optimal results, the timing overhead to obtain a result is significant and is not allowed for online updates. On the other hand, the network status may not always be stable and subsequently, the bandwidth between the OBU, edge, and cloud may change

over time. There is a need to adjust the strategy during runtime to adapt to the change of the network status.

For the offline part, an ILP formulation is designed to compute the optimal offloading strategy, including partition, allocation, and scheduling for the DAG on all three platforms, according to the given network parameters. The optimal strategies for a set of possible network parameters are computed and stored in a lookup table for online use. In this manner, the timing overheads for the ILP computation will not affect the runtime.

The ODA scheduling algorithm takes the offline scheduling strategy as a basis and greedily computes a new offloading strategy, taking the actual network state into consideration. If the current network state matches those in the lookup table, the strategy stored in the lookup table is chosen. If not, the strategy closest to the network state is used as the initial state of our greedy algorithm. This strategy takes the minimum Euclidean distance of two groups of networks as the selection rule. When the Euclidean distance is equal, the smaller distance of the cellular networks will be preferred. This process will be activated for each network status update during runtime.

B. Problem Formulation

According to the data dependency relation in the DAG and the deadline of each task, we analyze the schedulability of each computation task and obtain a series of constraints. The objective of the ILP model is to find an offloading allocation and scheduling strategy, which assigns tasks to different platforms, such that each task can meet constraints of data dependency in DAG and the response time of tasks is the shortest. For task t_j in a given schedule, its release time of t_j is affected by the tasks which have data dependence with t_j or the tasks executing on the same platform, but whose deadline is earlier than t_j. Let us set $\mathcal{T}_j$ be the set of all tasks which are executed on the same platform p as t_j, but other than t_j, that is

$$\mathcal{T}_j = \left\{ t_i \neq t_j \mid t_i \in T,\ x_i^p + x_j^p = 2 \right\} \tag{2}$$

where $x_i^p + x_j^p = 2$ means t_i and t_j are executed on the same platform p. Let us set D_j be the set of all tasks needed to execute task t_j, uniting all tasks whose deadlines are less than task t_j

$$D_j = \left\{ t_i \mid t_i \in T, e_{ij} = 1 \right\} \cup \{ t_i \mid t_i \in \mathcal{T}_j, \delta_i \leq \delta_j \}. \tag{3}$$

Then, the release time r_j of t_j can be written as

$$r_j = \max \left\{ (f_k + \sum_{i=1}^{|T|} \tau_{ik}) \mid t_k \in D_j \right\}. \tag{4}$$

The terminal time f_j of t_j can be written as

$$f_j = r_j + \sum_{p \in P} x_j^p \tau_j^p \tag{5}$$

where $\sum_{p \in P} x_j^p \tau_j^p$ is the execution time of t_j.

Therefore, the optimization problem that finding the minimum response time L with offloading scheduling $X = \{X_i\}$ can be described as

$$\textbf{Minimize } L = \max(f_j) \tag{6}$$

subject to

$$\forall\ i, \quad \sum_{p \in P} x_i^p = x_i^V + x_i^E + x_i^C = 1 \tag{7}$$

$$L - \sum_{p \in P} x_i^p \tau_i^p \geq r_i \geq 0 \tag{8}$$

$$L - \tau_{ij} \geq r_{ij} \geq 0 \tag{9}$$

$$\forall \sum_{p \in P} x_i^p + x_j^p = 2, r_j \geq r_i \quad r_j - r_i \geq \sum_{p \in P} x_i^p \tau_i^p \tag{10}$$

$$\forall \sum_{p \in P} x_i^p + x_m^p = 2, r_{mn} \geq r_{ij}, \quad r_{mn} - r_{ij} \geq \tau_{mn} \tag{11}$$

$$\forall \sum_{p \in P} x_i^p + x_j^p = 2, r_j \geq r_i, \quad \delta_j \geq \delta_i \tag{12}$$

$$\delta_i \geq r_i + \tau_i \tag{13}$$

$$r_j - \tau_{ij} \geq r_{ij} \geq r_i + \sum_{p \in P} x_i^p \tau_i^p. \tag{14}$$

In order to generate the optimal solution, the ILP model considers the following constraints.

1) *Schedulability:* Constraint (7) represents that each task can and only can execute on one of the platforms in $P = \{V, E, C\}$. Constraints (10) and (11) represent that the difference of the release time r_i and r_j/r_{ij} and r_{mn} of computation or communication tasks without data dependency executed on the same platform should be at least the execution time of the task performed previously. In other words, once a computation or communication task has been scheduled to start execution, no other computing or communication tasks can start execution until the communication task has completed its execution. Besides, the release time r_j/r_{ij} of computation or communication task should be larger than the beginning of the current period and its terminal time should be less than the finishing of the period. Each computation or communication task should be scheduled during the period, shown as (8) and (9).
2) *Data Dependency:* Constraint (14) shows that communication task c_{ij} should wait until the finish of task t_i and be executed before the release of task t_j.
3) *EDF Policy:* As shown in (12) and (13), the tasks without data dependency executed on the same platform should be scheduled with EDF, which means that the task with a smaller release time has a smaller deadline and the deadline for each task should be larger than its terminal time.

C. ODA Algorithm

Usually, limited by extra high computation costs, the offline ILP model is only deployed on edge or cloud. In addition, in order to ensure reliability and security, the online scheduling method must be initiated locally and lightweight, especially for low-cost electronics of autonomous driving. During the runtime of autonomous driving, we use a heuristics algorithm to offload tasks, i.e., allocation of t_i on a platform. Based on the heuristics algorithm, the chosen option for offloading of each task should be the optimum among all possible schedulings of this task. Assume that the task results are returned to

the original platform by default, let Cost_i^p be the sum of computation and communication time of the task t_i performed on platform p, that is

$$\text{Cost}_i^p = \tau_i^p + \sum_{j=1}^{|T|} \tau_{ij} + \sum_{j=1}^{|T|} \tau_{ji}. \tag{15}$$

Assume that task t_i is offloaded on the platform p, then the cost of t_i on p must be less than or equal to the minimum of the cost of t_i on the other platforms, which gives

$$\text{Cost}_i^p \leq \min\{\text{Cost}_i^q\} \tag{16}$$

where q is the platform that $p \neq q$.

Such a method is more suitable for serial cases, which makes the best gain only for the current task and does not consider the parallelism between tasks. Therefore, we add the parallel module to the offline schedule module, which will greatly shorten the period of tasks in some cases. Setting a penalty coefficient to the platforms on which t_{i-1} and $t_{i-2}(i > 2)$ are executed, task t_i is performed in parallel with task t_{i-1} and task t_{i-2}, when possible. When a new task t_i comes, it prefers to be offloaded to a platform with available resources. If there are available resources on more than one platform, task t_i should be scheduled based on the heuristics algorithm mentioned above. In other words, to a certain extent, we can make the degree of parallelism for computation tasks higher.

The pseudocode of the adaptive dynamic scheduling method is shown in Algorithm 1. First, the current network status of the vehicle is queried and the closest item in the ILP lookup table is matched to get a reference schedule (line 1). Starting from the first task t_i that is not assigned to any platform (line 2), the algorithm checks the data dependency relations of t_i in the DAG and calculates the communication cost from other tasks to t_i (lines 3–5). First, t_i is offloaded to the platform p that has the minimum cost (lines 6–9). Next, all tasks with existing data dependency with t_i or executing on the same platform with t_i will be found to ensure the release time and finishing time of t_i (lines 10–14). Meanwhile, it will check whether p has available resources and executes tasks in parallel with the tasks that do not have data dependencies, if possible (line 15). After that, we find the best platform with such a greedy strategy (lines 16 and 17). Then, we calculate the latency of the reference schedule with the current network status (line 18). Finally, we get the latency with maximum release time and compare it with the latency of the reference schedule to get a better result (lines 19 and 20). Iteratively, a schedule based on the ILP lookup table is found.

Algorithm 1: ODA Algorithm

Input: DAG= $G(T, E)$, *LookUpTbl*, *NetkSta*, *AvaRes*
Output: Latency L, Objective task schedule S

1 $CloNetSta \leftarrow ClosestNetkSta(NetSta)$
$RefS \leftarrow LookUpTbl[CloNetSta]$
2 **for** *each* $t_i \in T$ **do**
3 **for** *each* $t_j \in T$ **do**
4 **if** $e_{ji} = 1$ **then**
5 $UpTime_i^p \leftarrow Cal(i, AvaRes, NetkSta)$
$DownTime_i^p \leftarrow Cal(i, AvaRes, NetkSta)$
$TransTime_i^p \leftarrow UpTime_i^p + DownTime_i^p$
6 $Cost_i^p \leftarrow \tau_i^p + TransTime_i^p$
7 **while** $x_i^v + x_i^e + x_i^c = 0$ **do**
8 **if** $Cost_i^p = min(Cost_i^v, Cost_i^e, Cost_i^c)$ **then**
9 $x_i^p \leftarrow 1$
10 **for** *each* $t_j \in T$ && $i \neq j$ **do**
11 **if** $\sum_{p \in P} x_i^p x_j^p = 1 \parallel e_{ji} = 1$ **then**
12 $D_i \leftarrow D_i + \{j\}$
13 $D_i \leftarrow D_i - \{i\}$; $r_i \leftarrow \max(f_{D_i}) + UpTime_i^p$
14 $f_i \leftarrow r_i + \sum_{p \in P} x_i^p \tau_i^p$
15 $(f_i', x') \leftarrow parallel(G, x, i)$
16 **if** $f_i > f_i'$ **then**
17 $f_i \leftarrow f_i'$; $x \leftarrow x'$
18 $f_i^{ILP} \leftarrow CalLatency(i, RefS)$
19 **if** $f_i > f_i^{ILP}$ **then**
20 $f_i \leftarrow f_i^{ILP}$; $x \leftarrow RefS$
21 $S \leftarrow x$; $L \leftarrow \max(f_j)$
22 **return** (L, S)

VI. Performance Evaluation

We evaluate our approach with randomly generated DAGs. The DAG is generated by a random generator. While generating the adjacency relations in the DAG, i.e., the data dependency relations, the generator also gives complete information about each task $t = (\text{id}, \ \delta, \ \omega)$. For other relevant network parameters, we use a model presented in [43]. The wireless data transmission rate (Tx) is calculated from data channel characteristics [44]

$$\varphi_{p_1,\ p_2} = B \log_2 (1 + \frac{PH}{\sigma^2}) \tag{17}$$

which is a fundamental tradeoff between transmission rate, bandwidth, and signal-to-noise ratio.

The Tx rate is simulated with predefined network configurations to represent common situations: transmitting power $P = 0.1$ W, data channel bandwidth $B = 10$ MHz, and noise power is $\sigma^2 = 10^{-13}$ W. The wireless channel gain is given by the inverse power law [45], $H^{(u)} = H^{(d)} = A_0 l^{-2}$, where l is the distance between the vehicle and BS, and $A_0 = -17.8$ dB [46]. As defined in the system model, $\varphi_{V,E}$ and $\varphi_{E,V}$ can be calculated by the definition of the Tx rate. For $\varphi_{V,C}$ and $\varphi_{C,V}$, although the BS and core cellular network can offer a very high data rate and throughput, it is common for the cellular carrier to impose a limit on each user's bandwidth allocation. In the simulation, we set the maximum allowed usage for a single user as 30 Mb/s. Both $\varphi_{V,C}$ and $\varphi_{C,V}$ are set to be equal in the simulation to authentically imitate regular scenarios for commercial applications, used for autonomous driving services. The $\varphi_{E,C}$ and $\varphi_{C,E}$ pairs also transfer data via the backhaul network, therefore they are also simulated using the 30 Mb/s data rate. Computation capacities

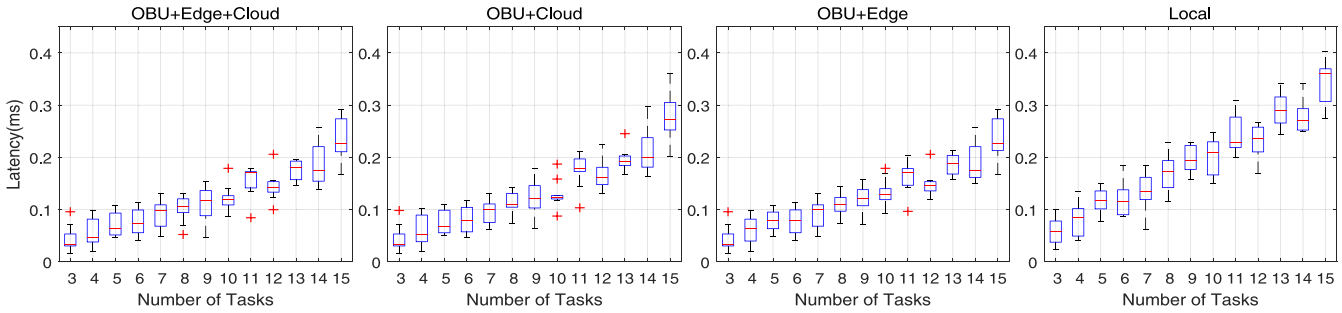


Fig. 6. Latency of DAG with different numbers of tasks. For any given number of tasks, identical generated DAG with tasks of the same specifications is used as the input for different configurations and schedule methods.

TABLE II
EVALUATION PARAMETERS

Parameter	Description	Value
$\varphi_{V,E}$, $\varphi_{E,V}$	Up-, down-link data rate between V and E (*Mbps*)	153
$\varphi_{V,C}$, $\varphi_{C,V}$	Up-, down-link data rate between V and C (*Mbps*)	30
$\varphi_{E,C}$, $\varphi_{C,E}$	Up-, down-link data rate between C and E (*Mbps*)	30
$F^{\{V,E,C\}}$	Computation capacity on each platform (GHz)	{1.4, 2.0, 2.5}

$F^{\{V,E,C\}}$ on each platform are set in the simulation as 1.4, 2.0, and 2.5 GHz, respectively. The simulation parameters used in the experiment are shown in Table II.

We first evaluate the obtained latency by comparing cases of only OBU, OBU with the edge, OBU with cloud, and OBU with both edge and cloud. DAG with different numbers of tasks is generated for the comparisons. For each number of task sets, in total, 30 sets are randomly generated. The results are shown in Fig. 6. From this figure, we have the following observations. First, the latency increases as the number of tasks increases. The reason is that the computation demands increase. Second, compared to the results of offloading tasks to edge or cloud, the latency is the shortest when utilizing all three. Third, the gap between OBU and others enlarges as the number of tasks increases. That is because as the number of tasks increases, OBU that cannot obtain more computation resources will become more laborious.

We also compare the results with and without online adaption. For a given set of tasks and network status, the results of the ODA algorithm, ILP optimal, and the ILP lookup table (the shortest Euclidean distance) are reported. The results are shown in Fig. 7. In the case of the ILP model, the delays are computed for each network parameter with ILP. Thus, the results from ILP can be considered as the ground truth. From the figure, first, the ILP lookup table performs always the worst, as it only uses offline information to compute the schedule, which overestimates the network condition. For example, when the network status is [1.1357, 16.2159], the reason for the high delay in the case of the ILP lookup table is that it incorrectly offloads some critical tasks that require a large amount of bandwidth to the edge or cloud for execution, which brings high transmission costs. Therefore, the latency can be even double than the actual optimal. Second, our ODA algorithm can adapt to the network changes, and the obtained results are closed to the optimal. This result shows that the

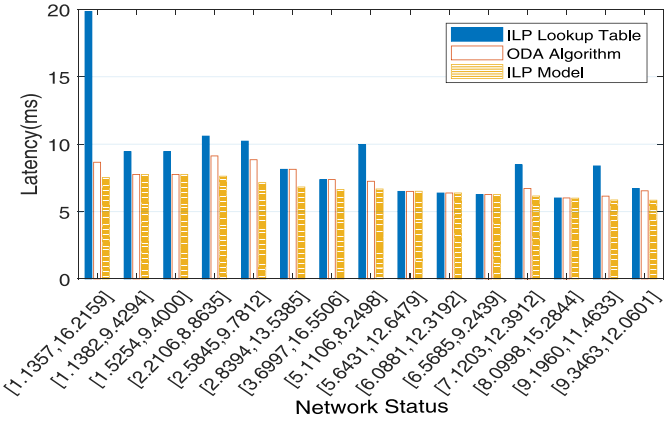


Fig. 7. Latency of a single DAG with a different network status. The network status with a form of [Cellular, WEN] is generated randomly.

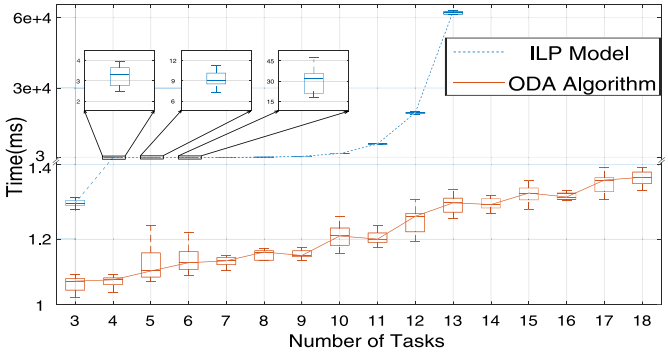


Fig. 8. Time overheads of ILP and ODA algorithm for different numbers of tasks.

ODA algorithm based on the ILP lookup table can effectively reduce task response time and improve system performance.

Fig. 8 shows the timing overheads of the ODA and ILP model. For each number of tasks, we report 30 runs of randomly generated task sets. Our ODA algorithm increases slowly with the number of tasks within the time range of millisecond. When the number of tasks increases from 3 to 18, the average time overheads by the ODA algorithm only increases by about 0.4 ms. On the other hand, the time overheads of the ILP model increases exponentially, which is unacceptable for real-time systems. In addition, in practice, the number of tasks that need to be scheduled is often large [42], and it is also not practical to use ILP for scheduling. For autonomous driving

systems, the task offloading algorithm should not only have good performance but also be lighter weight to ensure real time and reduce additional computation costs. This justifies the necessity of our online–offline hybrid approach.

VII. Real Deployment

In this section, we conduct a real-world case study using SLAM in an autonomous vehicle to verify the effectiveness of our approach.

A. Case Study

As shown in Fig. 1, accurate localization information is required for autonomous driving for path planning and behavior decision. Generally, autonomous driving vehicles need localization accuracy at the range of decimeters to maintain within a lane [47]. Traditional global position system (GPS) sensors cannot provide such accuracy. In scenarios, such as tunnels or roads surrounded by trees or buildings, where the signal is obscured, the deviation will be metered. SLAM is used for autonomously constructing a descriptive map of passed area and localization by 3-D LiDAR scanners which can launch and receive lasers to get environment data [48], [49]. The front part in SLAM uses sensor data to estimate the relationship between frames, while the back part uses filters to optimize the result and get final accurate pose and map. For autonomous driving vehicles, SLAM based on LiDAR needs a lot of calculation to get accurate positioning results to make up for the lack of GPS localization.

The SLAM algorithm used here is based on the work of laser odometry and mapping (LOAM) [50], which includes four core parts: 1) feature extracting; 2) odometry estimate; 3) mapping; and 4) localization. The pipeline is shown in Fig. 4 (a). T1 collects raw data from LiDAR, the physical device. In T2, raw data from LiDAR will be classified and features are extracted according to their spatial information. T1 and T2 are considered as the preprocessing phase. Then, T3 analyzes features in two consecutive LiDAR frames from T2 and estimates the relationship between them using the Levenberg–Marquardt method [51]. We can get the correct position and orientation of each frame under the global coordinate system and map them in global coordinates by accumulating frames in T4. Finally, we can get the precise localization information in T6. During serial computing, percentages of each module in the total processing time are 13.24%, 11.77%, 4.59%, 43.76%, 0.84%, and 25.8%, respectively.

B. Deployment

In our deployment, a VM with Skylake CPU Intel Xeon Platinum 8163 from Alibaba Cloud is chosen as the cloud server. A Tegra X2 and a Raspberry Pi III are used as the edge server and OBU, respectively. The Tegra X2 communicates with Pi III by a wireless bridge, and they are both connected to Alibaba Cloud through the cellular network. The detailed hardware specifications are listed in Table III.

The experiment environment is deployed on an automated vehicle modified from a gasoline-powered Dongfeng Fengshen AX7 SUV. The modified vehicle will have the drive-by-wire ability, which means we can use electrical signals to control (including throttle, brake, steering, etc.) rather than traditional hydraulic and mechanical techniques. The architecture of the vehicle is shown in Fig. 9. This vehicle is loaded with multiple sensors, e.g., Velodyne HDL-32 LiDAR, radar, and camera, which provide crucial information not only for scene understanding but also for later on decision making and planning. Additionally, a series of IMU devices are onboard, including the external differential global positioning system (DGPS) for recording RTK data. The IMU device records the latitude and longitude to get the vehicle odometry data which will be regarded as part of the ground truth in our experiment.

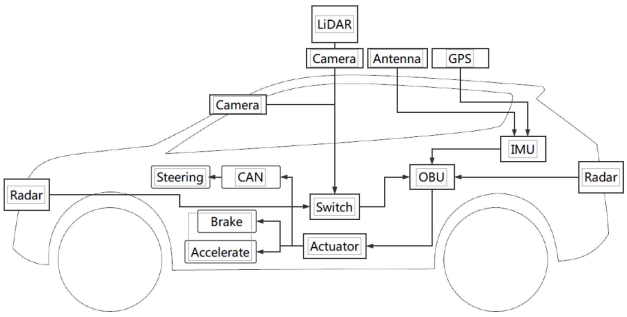
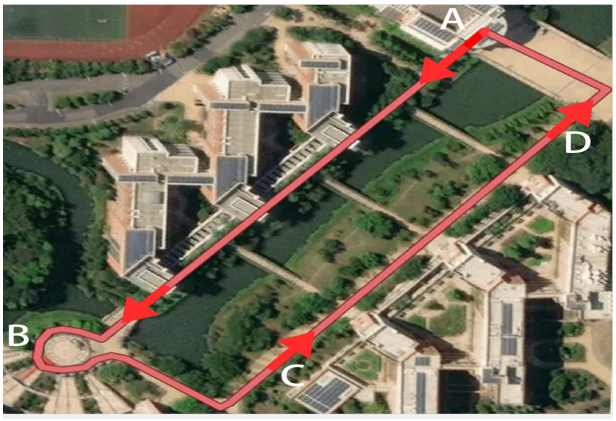
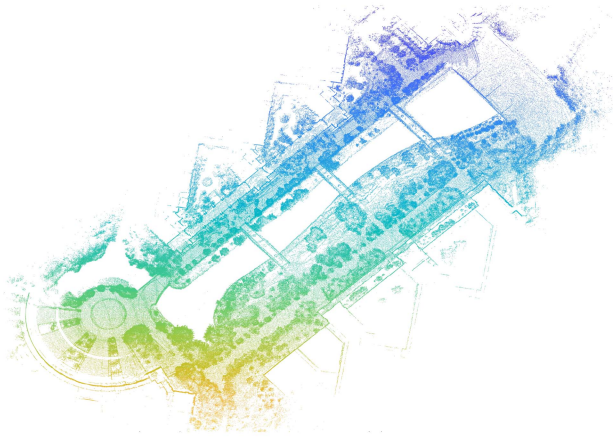


Fig. 9. Architecture for the Dongfeng AX7 vehicle on which an autonomous driving environment is deployed in the experiment.



(a)



(b)

Fig. 10. Realistic path and mapping result. (a) Guide line. (b) Mapping generated by LiDAR.

TABLE III
HARDWARE SPECIFICATION OF THE TEST PLATFORM

Platform	Vehicular	Edge	Cloud
Device	Raspberry Pi III B+	Tegra X2	Alibaba Cloud
Host CPU	Broadcom BCM2837B0	ARM A57 quad-core	Intel Xeon Platinum 8163
Architecture	Kepler	Pascal	Skylake
Clock Freq (GHz)	1.4	2.0	2.5
Memory Type	LPDDR2	LPDDR4	LPDDR4
Memory Size (GB)	1	8	8

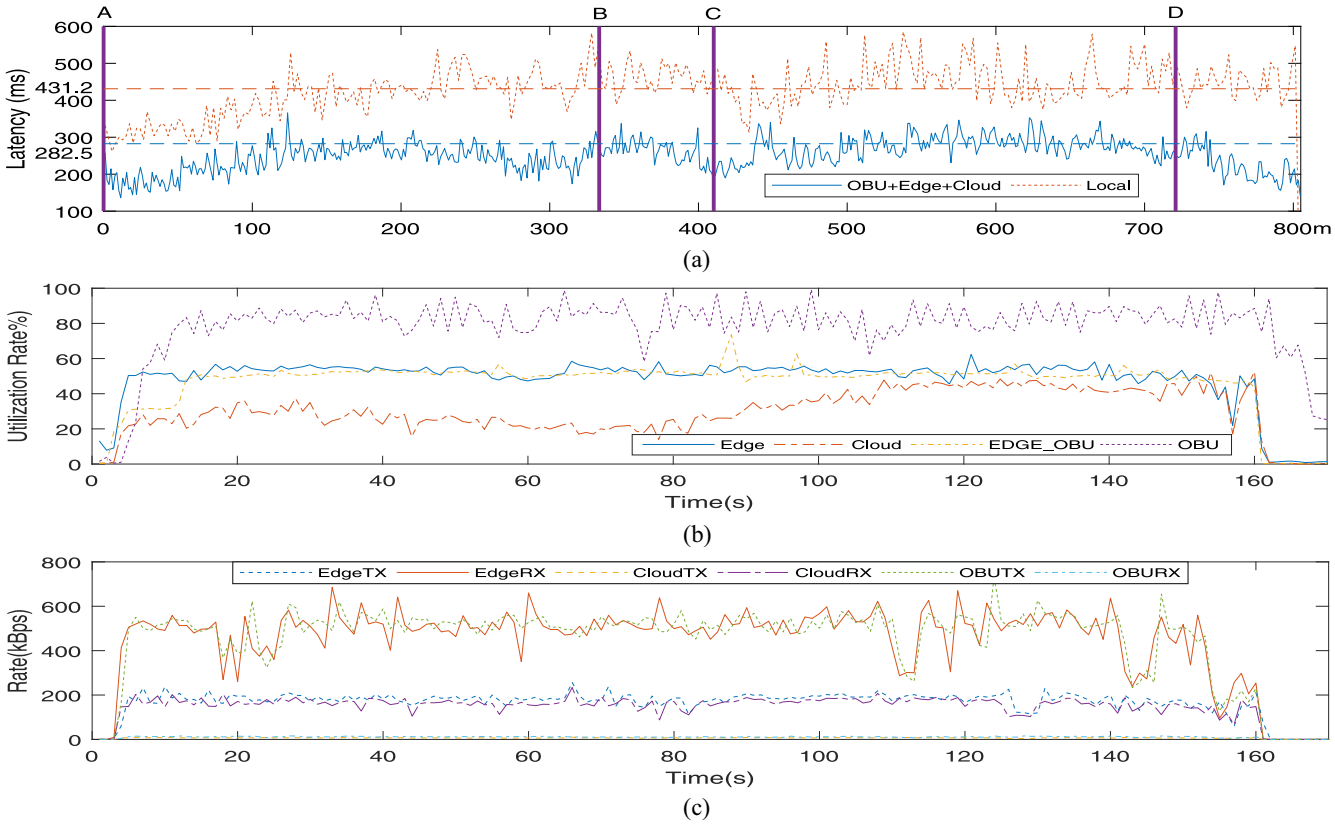


Fig. 11. Resource utilization and performance changes during vehicle driving. (a) Latencies for the 805 m autonomous driving. (b) CPU utilization statistics. (c) Data transmission bandwidth between platforms.

To make task offloading possible, we implement the SLAM algorithm for all platforms, i.e., for OBU, edge, and cloud. Before execution, programs are installed on all platforms. On the one hand, for the data transmission, programs are installed on all platforms, such that during the runtime, they can work cooperatively to provide autonomous driving services and all data that may need to transfer between platforms are limited to input/output data, not the programs themselves. On the other hand, for the execution, this way contributes to faster execution on every platform, since they will run native programs, instead of using VMs or bitcode conversions. What is more, online migrating the program will introduce additional communication overhead, which will significantly increase the complexity of the current model.

C. Performance

To evaluate the effectiveness of our model in the real world, we conduct a road test. The length of the path for the road test is 805 m, including turns, driving straight, and lane changing. As shown in Fig. 10 (a), the travel route contains a closed loop denoted by the red line. The vehicle starts at position A and the direction of travel is indicated by the arrow. The vehicle first passes through a straight road to position B, passes through a circle to position C, then goes straight to position D, and finally returns to position A through an open bridge. The map generated from the real-world path can be observed in Fig. 10 (b), which is tested by an autonomous vehicle at a speed of 20 km/h. Fig. 11(a) presents the latencies for the 805 m drive for both local OBU and offloading executions. The dashed line represents the average latency of the journey and with only local OBU it is 431.2 ms whereas with offloading it is 282.5 ms. From the figure, one can observe that the latencies of offloading reduce by 34% than the case of local OBU, as expected. The second observation is that in both cases the latencies are fluctuant. The main reason is the change of the network state for latencies of offloading executions. The different computation costs to execute the tasks caused by the different number of feature points extracted from LiDAR in each frame are another major reason. For example, compared to the BC segment with more trees, the path

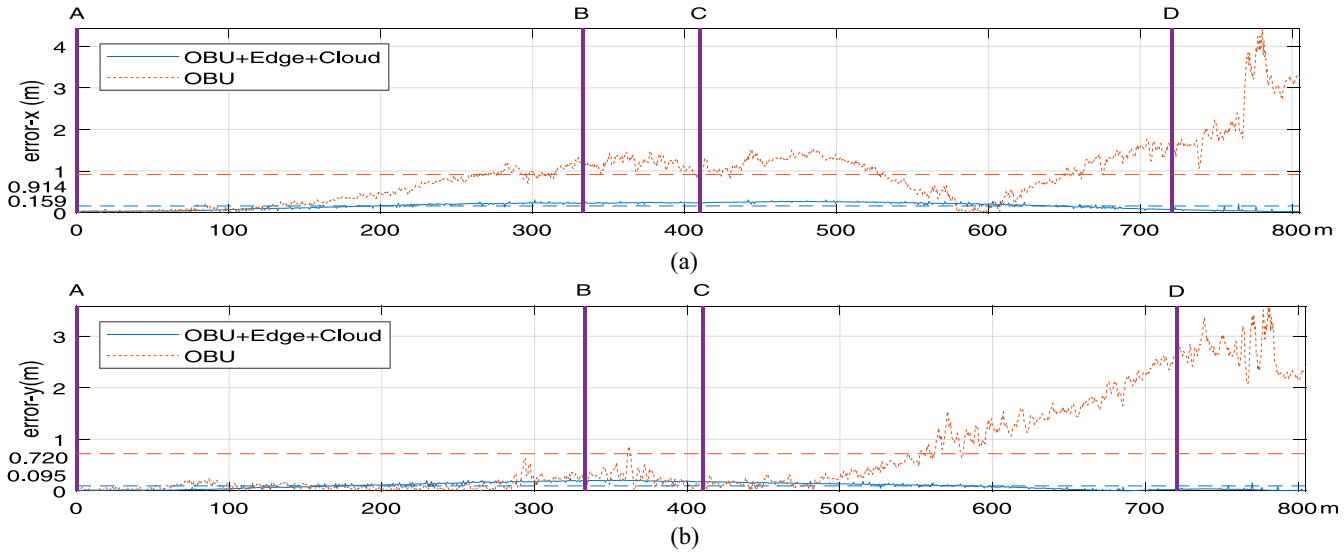


Fig. 12. Localization errors compared with RTK. (a) Lateral error. (b) Longitudinal error.

about 50 m after position C is more empty which means fewer feature points extracted from LiDAR. In addition, benefiting from the good network quality of the vehicle driving, the variance of the offloading executions is 1.6835e+03 while that of local OBU is 4.3868e+03. It means that fluctuations of the latencies for offloading are much less, which shows offloading autonomous driving services via edge computing can get more stable performance. In future work, we will test more cases and scenarios of how quality performance varies with different parts of the trajectory. This will help to further improve our system and bring it closer to real-world deployment.

At the same time, we record the resource utilization of each platform and report the results in Fig. 11(b) and (c). From Fig. 11(b), one can find that edge computing also reduces the use of local computation resources. Combined with Fig. 12, we can also see that in the case of OBU, the CPU has to be fully utilized but still cannot obtain proper accuracy. Fig. 11(c) shows the bandwidth utilization for all three platforms' two resource types (uplink and downlink bandwidth). As the source of data, the autonomous vehicle needs to send raw sensor data to various platforms, so it requires a great uplink bandwidth. But it is not sensitive to downlink bandwidth because it only needs to collect a few bytes of localization results. We also observe that the uplink bandwidth of edge servers is very small. The reason is that the edge server and cloud are connected by cellular networks, and the computing power of the cloud is not particularly obvious compared with the edge server advantage in our experiment.

We report three trajectories in cases of RTK, only OBU, and OBU with both edge and cloud, shown in Fig. 13. OBU is the trajectory computed within the vehicle by using only the OBU. RTK is the geodetic trajectory obtained from differential GPS. We choose RTK odometry data as ground truth reference to evaluate mapping accuracy, as the accuracy of differential GPS can be within decimeters. More details of localization errors can be seen from Fig. 12, where the average localization error is represented by dashed lines. From these figures, we have the following observations. First, the OBU performs worse and

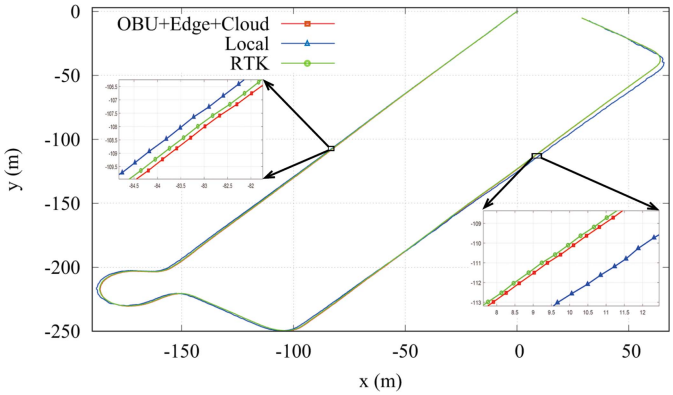


Fig. 13. Trajectory of the vehicle in cases of RTK, only OBU, and OBU with both edge and cloud.

has a bigger deviation from RTK. The reason is that computation resources limit local OBU to a rough localization result from T5 in most cases, which cannot be used to correct accumulative errors in time. Second, the trajectory of the posture computed via edge computing is closely matched with the one of RTK, and its construction result almost has little errors. Compared to OBU, the average accuracy of lateral and longitudinal localization is improved about 5.7 *times* (from 0.914 to 0.159 m) and 7.6 *times* (from 0.72 to 0.095 m), respectively. In addition, the worst case accuracy improves about 15 *times* (from 4.4288 to 0.3098 m) in lateral and 14 *times* in longitudinal (from 3.5799 to 0.2413 m). These results show that compared with using OBU only, the SLAM algorithm based on edge computing can result in better performance.

VIII. Conclusion

In this article, we proposed an approach to offloading autonomous driving services via edge computing. This approach consists of an offline scheduling strategy and an ODA scheduling algorithm, which can adjust the scheduling

sequence during runtime. Both synthetic simulations and real-world deployment demonstrate that offloading autonomous driving services via edge computing can improve QoS of autonomous driving and reduce local computing resources, effectively. From our experiments, we believe edge computing can help to improve autonomous driving. In future work, we will model the network instability and try to expand our approach to multivehicle scenarios.

REFERENCES

[1] A. González *et al.*, "Pedestrian detection at day/night time with visible and FIR cameras: A comparison," *Sensors*, vol. 15, no. 9, pp. 24615–24643, 2015.

[2] Y.-D. Kim, G.-J. Son, C.-H. Song, and H. Kim, "On the deployment and noise filtering of vehicular radar application for detection enhancement in roads and tunnels," *Sensors*, vol. 18, no. 3, p. 837, 2018.

[3] X. Li, B. Yang, X. Xie, D. Li, and L. Xu, "Influence of waveform characteristics on LiDAR ranging accuracy and precision," *Sensors*, vol. 18, no. 4, p. 1156, 2018.

[4] *NVIDIA ISAAC Launches New Era of Autonomous Machines*, Nvidia Newsroom, Santa Clara, CA, USA, Jun. 2018.

[5] Mobileye. (2018). *Autonomous Driving*. [Online]. Available: https://www.mobileye.com/our-technology/evolution-eyeq-chip/

[6] *The DARPA Urban Challenge*, DARPA, Arlington, VA, USA, 2007.

[7] S. Kumar, S. Gollakota, and D. Katabi, "A cloud-assisted design for autonomous driving," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput. (MCC)*, 2012, pp. 41–46.

[8] M. Gerla, "Vehicular cloud computing," in *Proc. 11th Annu. Mediterr. Ad Hoc Netw. Workshop (Med-Hoc-Net)*, 2012, pp. 152–155.

[9] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.

[10] Q. Yuan, H. Zhou, J. Li, Z. Liu, F. Yang, and X. S. Shen, "Toward efficient content delivery for automated driving services: An edge computing solution," *IEEE Netw.*, vol. 32, no. 1, pp. 80–86, Jan./Feb. 2018.

[11] D. Sabella *et al.*, "Toward fully connected vehicles: Edge computing for advanced automotive communications," 5GAA Autom. Assoc., München, Germany, Rep., 2017.

[12] E. Ahmed and M. H. Rehmani, "Mobile edge computing: Opportunities, solutions, and challenges," *Future Gener. Comput. Syst.*, vol. 70, pp. 59–63, May 2017.

[13] MobilEye. (2017). *Autonomous Driving*. [Online]. Available: https://www.mobileye.com/

[14] Apollo. (2017). *Autonomous Driving*. [Online]. Available: http://apollo.auto/

[15] J. M. Borrmann, F. Haxel, A. Viehl, O. Bringmann, and W. Rosenstiel, "Safe and efficient runtime resource management in heterogeneous systems for automated driving," in *Proc. IEEE Int. Conf. Intell. Transp. Syst.*, 2015, pp. 353–360.

[16] 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," 2017. [Online]. Available: arxiv.abs/1711.11114.

[17] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2017.

[18] X. Huang, R. Yu, J. Kang, Y. He, and Y. Zhang, "Exploring mobile edge computing for 5G-enabled software defined vehicular networks," *IEEE Wireless Commun.*, vol. 24, no. 6, pp. 55–63, Dec. 2017.

[19] F. Zhou, Y. Wu, H. Sun, and C. Zheng, "UAV-enabled mobile edge computing: Offloading optimization and trajectory design," in *Proc. ICC*, 2018, pp. 1–6.

[20] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 3, pp. 1628–1656, 3rd Quart., 2017.

[21] W. Zhang, Y. Wen, K. Guan, D. Kilper, H. Luo, and D. O. Wu, "Energy-optimal mobile cloud computing under stochastic wireless channel," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4569–4581, Sep. 2013.

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

[23] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11365–11373, 2018.

[24] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[25] W. Fan, Y. Liu, B. Tang, F. Wu, and Z. Wang, "Computation offloading based on cooperations of mobile edge computing-enabled base stations," *IEEE Access*, vol. 6, pp. 22622–22633, 2018.

[26] Y. Mao, J. Zhang, and K. B. Letaief, "Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems," in *Proc. WCNC*, 2017, pp. 1–6.

[27] H. Shahzad and T. H. Szymanski, "A dynamic programming offloading algorithm for mobile cloud computing," in *Proc. Elect. Comput. Eng.*, 2016, pp. 1–5.

[28] S. Bi and Y. J. A. Zhang, "An ADMM based method for computation rate maximization in wireless powered mobile-edge computing networks," in *Proc. ICC*, 2018, pp. 1–7.

[29] L. Huang, S. Bi, and Y. A. Zhang, "Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks," 2018. [Online]. Available: arxiv.abs/1808.01977.

[30] H. Xing, L. Liu, J. Xu, and A. Nallanathan, "Joint task assignment and resource allocation for d2d-enabled mobile-edge computing," *IEEE Trans. Commun.*, vol. 67, no. 6, pp. 4193–4207, Jun. 2019.

[31] J. Tang, S. Liu, B. Yu, and W. Shi, "Pi-edge: A low-power edge computing system for real-time autonomous driving services," 2018. [Online]. Available: arxiv.abs/1901.04978.

[32] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," 2016. [Online]. Available: arxiv.abs/1610.03295.

[33] Udacity. (2017). *An Open Source Self-Driving Car*. [Online]. Available: https://www.udacity.com/self-driving-car

[34] S.-C. Lin *et al.*, "The architectural implications of autonomous driving: Constraints and acceleration," in *Proc. ASPLOS*, Mar. 2018, pp. 751–766.

[35] L. Verma, M. Fakharzadeh, and S. Choi, "WiFi on steroids: 802.11ac and 802.11ad," *IEEE Wireless Commun.*, vol. 20, no. 6, pp. 30–35, Dec. 2013.

[36] M. Hongzi, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group*, 2017.

[37] R. M. Rao, V. Marojevic, and J. H. Reed, "Analysis of non-pilot interference on link adaptation and latency in cellular networks," 2019. [Online]. Available: arxiv.abs/1901.02574.

[38] S. Chinchali *et al.*, "Network offloading policies for cloud robotics: A learning-based approach," 2019. [Online]. Available: arxiv.abs/1902.05703.

[39] Google. (2017). *Draco 3D*. [Online]. Available: https://google.github.io/draco/#

[40] Stanford. (1993). *The Stanford 3D Scanning Repository*. [Online]. Available: http://graphics.stanford.edu/data/3Dscanrep/

[41] D. Tian, H. Ochimizu, C. Feng, R. Cohen, and A. Vetro, "Geometric distortion metrics for point cloud compression," in *Proc. ICIP*, Sep. 2017, pp. 3460–3464.

[42] CNBC. (2018). *Tesla's Plan to Leave the Auto Industry Behind on In-Car Infotainment*. [Online]. Available: https://www.cnbc.com/2019/11/23/teslas-plan-to-leave-auto-industry-behind-on-in-car-entertainment.html

[43] T. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Upper Saddle River, NJ, USA: Prentice-Hall, 2001.

[44] C. E. Shannon and W. Weaver, "The mathematical theory of communication," *Bell Labs Techn. J.*, vol. 3, no. 9, pp. 31–32, 1950.

[45] M. Abdulla, E. Steinmetz, and H. Wymeersch, "Vehicle-to-vehicle communications with urban intersection path loss models," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Dec. 2016, pp. 1–6.

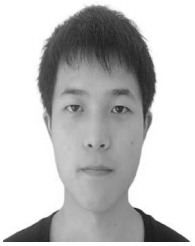
[46] Y. Sun, X. Guo, S. Zhou, Z. Jiang, X. Liu, and Z. Niu, "Learning-based task offloading for vehicular cloud computing systems," 2018. [Online]. Available: arxiv.abs/1804.00785.

[47] J. Levinson, M. Montemerlo, and S. Thrun, "Map-based precision vehicle localization in urban environments," in *Robotics: Science and Systems*. Cambridge, MA, USA: MIT Press, Jun. 2007.

[48] Y. Xu, V. John, S. Mita, H. Tehrani, K. Ishimaru, and S. Nishino, "3D point cloud map based vehicle localization using stereo camera," in *Proc. Intell. Veh. Symp.*, 2017, pp. 487–492.

[49] C. Park, P. Moghadam, S. Kim, A. Elfes, C. Fookes, and S. Sridharan, "Elastic LIDAR fusion: Dense map-centric continuous-time SLAM," 2017. [Online]. Available: arxiv.abs/1711.01691.

[50] J. Zhang and S. Singh, "LOAM: LIDAR odometry and mapping in real-time," in *Robotics: Science and Systems*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2014, p. 9.
[51] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge, U.K.: Cambridge Univ. Press, 2004.



Mingyue Cui received the B.Sc. degree in software engineering from Chongqing Normal University, Chongqing, China, in 2014, and the M.Sc. degree in software engineering from Sun Yat-sen University, Guangzhou, China, in 2017, where he is currently pursuing the Ph.D. degree in computer science under the instruction of Prof. K. Huang.

He is recently focused on SLAM and edge computing in autonomous driving. His research interests are in the areas of autonomous driving and Internet of Things.

Shipeng Zhong received the bachelor's degree in software engineering from Sun Yat-sen University, Guangzhou, China, in 2018, where he is currently pursuing the master's degree with the School of Data and Computer Science under the instruction of Prof. K. Huang.

He is recently focused on offloading autonomous driving services via edge computing. His research interests are in the areas of autonomous driving, cooperative mapping, and edge computing.

Boyang Li received the B.Sc. and M.Sc. degrees in software engineering from Sun Yat-sen University, Guangzhou, China, in 2016 and 2018, respectively, where he is currently pursuing the Ph.D. degree in computer science under the instruction of Prof. K. Huang.

His research interests include the areas of cooperative SLAM and event-based cameras.

Xu Chen (Member, IEEE) received the Ph.D. degree in information engineering from the Chinese University of Hong Kong, Hong Kong, in 2012.

He is a Full Professor with Sun Yat-sen University, Guangzhou, China, and the Vice Director of the National and Local Joint Engineering Laboratory of Digital Home Interactive Applications. He was a Postdoctoral Research Associate with Arizona State University, Tempe, AZ, USA, from 2012 to 2014, and a Humboldt Scholar Fellow with the Institute of Computer Science, University of Göttingen, Göttingen, Germany, from 2014 to 2016.

Prof. Chen was a recipient of the Prestigious Humboldt Research Fellowship awarded by Alexander von Humboldt Foundation of Germany, the 2014 Hong Kong Young Scientist Runner-Up Award, the 2016 Thousand Talents Plan Award for Young Professionals of China, the 2017 IEEE Communication Society Asia–Pacific Outstanding Young Researcher Award, the 2017 IEEE ComSoc Young Professional Best Paper Award, the Honorable Mention Award of 2010 IEEE International Conference on Intelligence and Security Informatics, the Best Paper Runner-Up Award of 2014 IEEE International Conference on Computer Communications (INFOCOM), and the Best Paper Award of 2017 IEEE International Conference on Communications. He is currently an Area Editor of the IEEE OPEN JOURNAL OF THE COMMUNICATIONS SOCIETY and an Associate Editor of the IEEE TRANSACTIONS WIRELESS COMMUNICATIONS, the IEEE INTERNET OF THINGS JOURNAL, and the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS Series on Network Softwarization and Enablers.

Kai Huang (Member, IEEE) received the B.Sc. degree from Fudan University, Shanghai, China, in 1999, the M.Sc. degree from the University of Leiden, Leiden, The Netherlands, in 2005, and the Ph.D. degree from ETH Zurich, Zürich, Switzerland, in 2010.

In 2015, he joined Sun Yat-sen University, Guangzhou, China, as a Professor. He was appointed as the Director of the Institute of Unmanned Systems of School of Data and Computer Science in 2016. He was a Senior Researcher with the Computer Science Department, Technical University of Munich, Munich, Germany, from 2012 to 2015 and a Research Group Leader with fortiss GmbH, Munich, in 2011. His research interests include techniques for the analysis, design, and optimization of embedded/CPS systems, particularly in the automotive, medical, and robotic domains.

Dr. Huang was awarded the Youth Overseas High-Level Talent Introduction Plan 2014 and was granted the Chinese Government Award for Outstanding Self-Financed Students Abroad 2010. He was the recipient of the Best (Student)Paper/Candidates Awards IV 2018, HFR 2018, ROBIO 2017, ESTC 2017, ESTIMedia 2013, SAMOS 2009, and ESTIMedia 2009, and the General Chairs' Recognition Award For Interactive Papers in CDC 2009. He has served as a member of the technical committee on Cybernetics for Cyber–Physical Systems of the IEEE SMC Society, Embedded Systems of China Computer Federation, Vehicle Control, and Intelligence for the Chinese Association Automation, and Robotic Intelligence for Chinese Association of Automation. He also serves as the Deputy Secretory of the technical committee on Intelligent Robotics of China Computer Federation.