Edge-facilitated Augmented Vision in Vehicle-to-Everything Networks

Pengyuan Zhou, *Member, IEEE*, Tristan Braud, *Member, IEEE*, Aleksandr Zavodovski, *Member, IEEE*, Zhi Liu, *Senior member, IEEE*, Xianfu Chen, *Member, IEEE*, Pan Hui, *Fellow, IEEE*, Jussi Kangasharju, *Member, IEEE*

***Abstract*—Vehicular communication applications require an efficient communication architecture for timely information delivery. Centralized, cloud-based infrastructures present latencies too high to satisfy the requirements of emergency information processing and transmission, while Vehicle-to-Vehicle communication is too variable for reliable in-time information transmission. In this paper, we present EAVVE, a novel Vehicle-to-Everything system, consisting of vehicles with and without comprehensive data processing capabilities, facilitated by edge servers co-located with roadside units. Adding computation capabilities at the edge of the network allows reducing the overall latency compared to vehicle-to-cloud and makes up for scenarios in which in-vehicle computational power is not sufficient to satisfy the service demand. To improve the offloading efficiency, we propose a decentralized algorithm for real-time task scheduling and a client/server algorithm for information filtering. We demonstrate the practical applications of EAVVE with a bandwidth-hungry, latency constrained real-life prototype system that connects vehicular vision through Augmented Reality vision. We evaluate this prototype system with real-life road tests. We complement this practical evaluation with extensive simulations based on real-world base station and vehicular traffic data to demonstrate the scalability of EAVVE and its performance in citywide scenarios. EAVVE decreases the latency by 42.6% and 78.7% compared to local and remote cloud solutions while relaxing congestion at the bottleneck by 99% with reasonable infrastructure expenditure.**

***Index Terms*—Edge Computing, V2X, Augmented Reality**

I. Introduction

In recent years, we have seen the apparition of multiple systems assisting or replacing humans in driving vehicles. These systems are becoming increasingly robust but also more complex. In 2018, California and Shanghai led the way towards large-scale adoption of automated cars by authorizing the deployment of autonomous vehicles on public roads for testing purposes [1], [2]. Modern vehicles feature a variety of systems for driver assistance in various scenarios such as lane following, emergency braking, and automated parking. These systems enhance the driving experience and drastically improve road safety. However, in most cases, they are limited to the point of view of a single vehicle. Complex scenarios may benefit from aggregating the points of views of several vehicles. For instance, if the distance to an obstacle is too short to perform emergency braking safely, the vehicle may choose another manoeuvre, such as steering into another lane. The system should not only request status from other nearby vehicles but also advertise the manoeuvre to the most immediate neighbours. Vehicular communication systems play a key role in sharing information between vehicles and roadside units (RSU). Current solutions focus on four types of communication: vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), vehicle-to-cloud (V2C) and vehicle-to-everything (V2X) [3]–[8]. Although these solutions fulfill basic demands, *efficiently sharing complex and large volumes of data among vehicles at scale* remains a challenge.

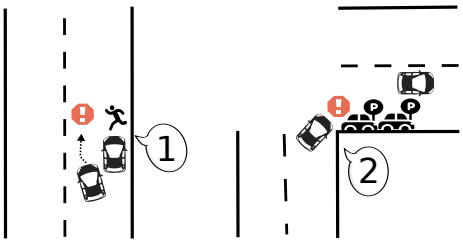


Fig. 1: Vehicular accidents.

In this paper, we address more particularly the problem of *sharing vision among vehicles at scale*. We believe that in order to further improve vehicular safety, vehicles and RSUs should aggregate the information collected by their respective sensors within a more complete vision of the road status. Figure 1 illustrates two types of vehicular accidents partly caused by the lack of visibility.

(i) The driver in the leading vehicle sees the pedestrian running across the street and slows down up to stop. However, the view of the driver in the following vehicle is blocked by the leading vehicle. The driver is not aware of the pedestrian and decides to overtake the leading vehicle, resulting in a fatal accident.

(ii) Some vehicles are double-parked and block a lane. The cars are parked right after the corner, out of the line-of-sight of the cars turning right. The leading vehicle turns slowly and was able to avoid an accident. However, the next car turns too fast to avoid a collision.

Both accidents would be avoided if the vehicles could share their visions in real-time. Sharing vision between two vehicles is a nontrivial issue. Further extending this paradigm to multiple vehicles at a large scale and in real-time significantly increases the complexity of the system. In this scenario, vehicles are flooded with a large number of ambient broadcast messages. To avoid information overload and its consequences (driver distraction, performance drops, network congestion), vehicles must select only the data relevant to their context. To tackle this challenge and improve the performance of V2X, we propose EAVVE (pronounced 'eevee'), the first detailed V2X framework for sharing augmented contextual information in

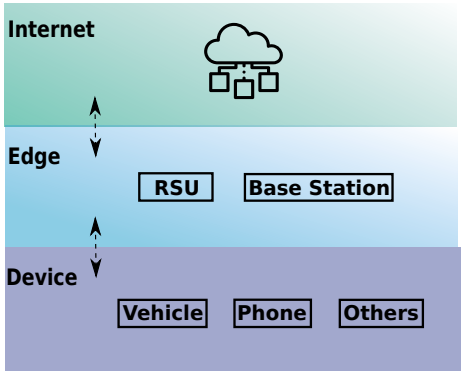


Fig. 2: EAVVE system model. The shaded areas show the logical division between the Device, the Edge (where edge server can be deployed) and the Internet. The inner boxes represent the physical entities.

real-time. V2X presents the advantage of combining the high availability of licensed spectrum technologies (LTE, 5G) with the dynamicity and ubiquity V2V and V2I communication in the unlicensed spectrum [9]. We illustrate the performance of this system by analyzing a specific case of Augmented Reality Head-Up Display (ARHUD), where vision sharing allows to highlight obstacles not in the direct line of sight of the user. In summary, we make the following contributions:

- We design and analyze EAVVE, a new V2X system based on edge computing. EAVVE allows vehicles with and without data processing capabilities to share contextual information at a large scale in real-time (§III).
- We apply EAVVE to a concrete case of connecting vehicle views using ARHUD. This displays the advantages of EAVVE, i.e., improving the scalability and efficiency of contextual information sharing while decreasing the transmission latency (§VII).
- We implement a prototype and evaluate EAVVE through real-life road testing. We expand our evaluation with an extensive set of simulations based on both a real dataset and our test results, and simulation results show that EAVVE offers noticeable performance improvements. It decreases latency by 42.6% and 78.7% compared with local (150km) and remote (2000km) cloud service, scales well in various traffic densities, with reasonable expenditure in infrastructure, 6.3 edge servers per square kilometre in the centre of London (§VIII).

II. Related Work

Emerging technologies not only enable various functions for autonomous vehicles but also bring new challenges.

Edge Computing brings computation close to the user and is a key component of the 5G architecture. Many works have studied the topic [10]–[12]. Edge computing has attracted attention in a vehicular context, such as [13], which explores the integration of 5G, SDN, MEC, and vehicular networks. Uncoordinated strategies for edge service placement have been investigated in [14], and the results have shown that they work well for this problem. Tran et al. investigate the optimal task offloading scheduling method at the edge for faster completion [15]. A solution for predictive task offloading to the edge is suggested in [16]. Rodrigues et al. combine advanced VM migration with transmission power control to minimize service delay experienced by smart vehicles in edge clouds utilization [17]. In [18], the hybrid approach of [17] is extended by sophisticated mathematical modeling and application of the Particle Swarm Optimization algorithm. The potential of software-defined networking for vehicular edge computing is highlighted by [19]. Tang et al. give an outlook for the future role of machine learning and 6G in the context of vehicular networking [20]. Meanwhile, the fundamental issues, i.e., architecture design, communication process, network protocols, and implementation concerns are yet to be explored. In this work, we propose the overall system and detailed functionality design with deployment evaluation based on public datasets.

V2X is gaining more attention from both academia and industry [21], [22]. Most works focus on overall system design and network protocol stack design [6], [23]. Our work integrates edge computing to gain low latency, propose information filtering policy to improve data processing efficiency, build the practical prototype and evaluate its performance with road tests.

Applications. Developing vehicular applications have achieved some results [24]–[27], but without improvement from system and networking point of view, those applications face difficulties to scale in realistic situations.

III. System Design

In this section, we discuss our system design. We first specify the system architecture before presenting the details of the microservices and the data flows between them.

A. System Architecture

Since latency is the key performance factor of vehicular networking system, we simplify the architecture to accelerate the data flow. EAVVE is defined in three layers: *Device, Edge and Internet* as shown in Figure 2. The edge layer contains the edge servers that are co-located with base stations or RSUs and operates within the area defined by the range of its corresponding infrastructure. The device layer contains other wireless devices involved in the communications, such as the vehicles, phones of pedestrians and wireless sensor units attached to bicycles. Vehicles, RSUs, and other devices communicate via protocols such as Dedicated Short Range Communications (DSRC), Cellular V2X (C-V2X) or hybrid architectures [6], [28], [29]. In this work, we consider the connected vehicle vision (CVV) as our use case. CVV requires real-time object detection capabilities. Therefore, we integrate machine learning capacities into the design of the edge servers and smart vehicles, as discussed in the following section.

B. System Data Flow

To better introduce the functionalities and data flows, we classify devices into two categories, namely *client devices*

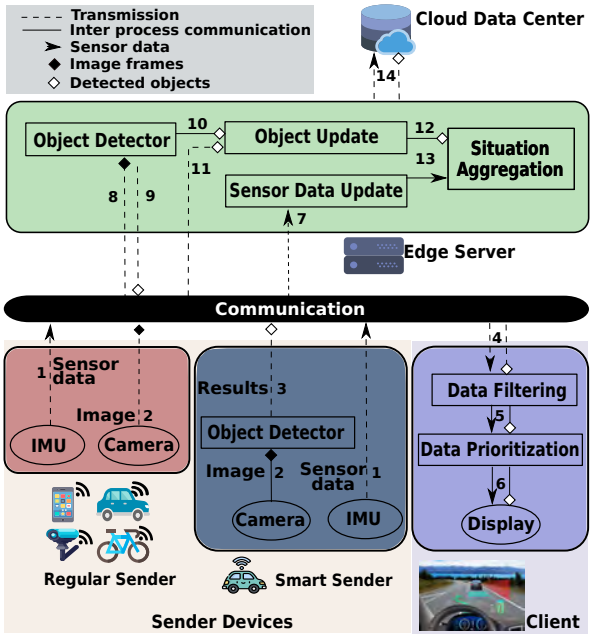


Fig. 3: EAVVE system data flow.

and *sender devices*. A client device refers to any unit that processes and displays received contextual information, e.g., the receiver and ARHUD in a vehicle. A sender device refers to any unit that sends out contextual information such as sensor data, captured camera images and object detection results. In this context, we name a sender with an object detector as *smart sender* as opposed to *regular sender* (Figure 3). An entity may have one or multiple units mentioned above. For instance, a smart vehicle with object detection capacity has a client device and a smart sender, a regular vehicle has a client device and a regular sender, and smartphones, surveillance cameras and bicycles only have regular senders. The edge server provides object detection offloading and data aggregation with an object detector and data filtering module. Client devices, sender devices, and edge devices run microservices continuously to process the data. To minimize operational latency, we need an efficient data flow. As shown by the numbered arrows in Figure 3, the major data flow includes the following steps: (1) Sensor devices collect and broadcast basic information including UID (unique ID), latitude, longitude, motion (speed, heading angle, acceleration), type (pedestrian, bicycle, vehicle), and size of the object. (2) Cameras on smart vehicles take images periodically and load them to the on-board object detector. Cameras on other devices send images to a nearby edge server. (3) The object detector broadcasts the detection results including the type, size, and position of the detected objects (calculated based on relative distance, see §VIII). The sensor data and detection results are broadcasted via DSRC at 10 Hz according to DSRC standard SAE J2735 BSM [30]. (4–5) The client device filters and prioritizes the received data based on a predefined mechanism. (6) ARHUD displays updated information (§VII). (7) The edge server updates the collected sensor data. (8) The edge server loads the received camera images (step 2) to the object detector. (9) The edge server sends the detection result to the sender. (10–11) The edge server updates the detected objects with its detection results combined with information received from nearby smart vehicles. (12–14) The edge server updates the road situation and aggregates data to the cloud database periodically.

As such, each vehicle is able to update the effective road situation and display it in real-time to facilitate driving. Meanwhile, the edge server maintains the nearby road situation and has thus the potential for fine-grained traffic monitoring and control. In the next section, we explain the networking model and detail the algorithms deployed in the vehicle on-board unit (OBU) and on the edge server.

IV. System Model

A. Networking Model

Node Classification. Until self-driving cars become the norm, the typical road scenario will be a mix of regular vehicles among an increasing number of driverless vehicles. Currently, most vehicles have access to the Internet via cellular networks and assist drivers with cloud services such as Google Maps. In the future, we foresee vehicles to be connected not only to the Internet but also to other vehicles and RSUs. As such, we classify the network nodes according to their roles and capacities, as follows. *Smart vehicles* (vehicles with powerful OBU), *regular vehicles* (vehicles without powerful OBU, i.e., standard cars), *edge servers*, *RSUs* (traffic signals, roadside sensors) and *Internet* (base stations, core network and cloud). We assume all vehicles have a wireless interface for V2V and V2I connection, e.g., PC5 interface for C-V2X [23]. For simplicity, we assume all RSUs (that are not co-located with an edge server) are simple gateways without additional processing capabilities and offload their workloads to a nearby edge server. As a result, the network is a hybrid integration, including V2V, V2E, V2I, and V2C. Each node communicates with the network by sending out one-hop broadcast or multicast messages and filters received messages with predefined rules.

Communication Model. In our implementation, the beacons and the computation input/output data transmissions are on the same frequency, i.e., 30 Hz. Each beacon packet has a size of 12 B, each computation input packet of the application has a size of 51.8 KB and each computation output packet is 512 B. Since the beacons have much smaller data volume than the computation input data (image frame) and the output data packet (detection result), beacon congestion control and signaling overhead optimization will have a minimal impact on overall performance. Besides, these topics are already covered by a rich literature [31]–[34]. Instead, we focus the networking model on the entities involved in computation offloading, i.e., vehicles $\mathbf{V} = \{V_1, V_2, ..., V_v\}$ and edge servers $\mathbf{E} = \{E_1, E_2, ..., E_e\}$. $\mathbf{V}_{E_j}$ denotes the set of regular vehicles that are offloading to E_j at a timepoint. The capacity of uplink and downlink of each base station that co-located with an edge server are denoted as B^{ul} and B^{dl} (bps). The instantaneous uplink and downlink data rate of a vehicle are denoted as R^{ul} and R^{dl} (bps). The effective data rate is affected by lots of factors such as signal strength, electromagnetic interference

and noise, as formulated in [35] and [36]. For simplicity, we assume each vehicle gets a fair share of B^{ul} and B^{dl}, i.e., $R^{ul}_{E_j} = B^{ul}/|\mathbf{V}_{E_j}|$ and $R^{dl}_{E_j} = B^{dl}/|\mathbf{V}_{E_j}|$.

B. Task Model

We consider that each vehicle V has a computation task that rises as a sequence of jobs (or invocations), $\tau_V \triangleq (I_V, C_V, O_V, T_V, P_V, D_V)$. Here I_V and O_V denote the data volume of computation input and output data, C_V denotes the execution time, i.e., the total number of CPU cycles multiplied by the clock cycle, required to accomplish the processing of τ_V. T_V denotes the period of τ_V. P_V denotes the priority of τ_V (§VI). D_V denotes the deadline of τ_V, i.e., the period within which O_V must return to the vehicle to be considered valid (see more details in §IV-C). In our implementation, we let all vehicles have the same values of I_V, C_V and O_V.

C. System Characteristics

A road net area can be considered as a large multiprocessor system, within which each edge server functions as a processor, and each vehicle periodically sends out the jobs (instances of its task τ_V). Based on the vehicular application feature, we recap the system characteristics utilizing some terms from multiprocessor scheduling, as follows: 1) *Homogeneous*: The edge servers are identical hence have the same rate of execution of all tasks. From the optimization point of view, how many threads each edge server executes do not affect the complexity of the task scheduling algorithm. For simplicity, we assume each edge server executes all received tasks in one thread. 2) *Arbitrary deadlines*: We set the deadline of each task as the inversely proportional value of its priority, i.e., $D_V = t/P_V$ where t is a predefined period value. The rationale is that a more important task should be prioritized in the sense of being processed and sent back the result faster. 3) *Task-level migration*: The jobs of a task may execute on different edge servers. Each job can only execute on a single edge server. 4) *Fixed job priority*: The jobs of a task may have different priorities. Each job has a single static priority. 5) *Preemptive*: Any task can be preempted by a task with higher priority at any time.

It is worth noting that our system has several different *key points* comparing with a general multiprocessor system: 1) The definition of *priority* is twofold: the importance of the job, e.g. the summarized importance of the identified objects in an image, and, the rank of the job in the processing queue. Assuming using earliest deadline first (EDF) scheduling algorithm, the rank of the job in the queue coincides with its importance (§IV-B). In other words, a more important task has a smaller deadline hence a higher rank in the processing queue. 2) The priority of each task can only be figured out after being processed by an edge server. Hence the server can not tell the deadline of each task before queuing. To tackle this challenge, we propose to use the historical track of each vehicle's task to predict the deadline of its current job (§VI). 3) Unlike a centralized or integrated system, a distributed wireless system like V2X brings considerable latency to collect statistics from clients, especially when performing real-time global scheduling. As such, the system requires solutions other than centralized scheduling.

Next, we formulate the mathematical model and problem. We present the optimization algorithm and prioritization policy addressing the key points in §VI.

D. Offloading Model

The offloading process for each regular vehicle includes input data transmission, processing in an edge server and output data transmission. The time cost for transmitting the computation input data of τ_V to edge server E is:

$$T^{off}_{\tau_V} = \frac{I_V}{R^{ul}_E} \tag{1}$$

The time costs for transmitting the output data to V is:

$$T^{back}_{\tau_V} = \frac{O_V}{R^{dl}_E} \tag{2}$$

The execution time of E on τ_V is:

$$T^{exec}_{\tau_V} = C_V + \sum_{v \in queue} C_v \tag{3}$$

where $\sum_{v \in queue} C_v$ is the sum of the execution time of the tasks ranked before τ_V in the processing queue. The length of queue, $|queue|$, equals the number of received tasks for single thread execution in edge servers (§IV-C). Then we can compute the total overhead of offloading τ_V from a regular vehicle V in terms of latency as:

$$L^E_{\tau_V} = T^{off}_{\tau_V} + T^{exec}_{\tau_V} + T^{back}_{\tau_V} \tag{4}$$

Smart vehicle process data in-car and send out the result. Therefore, the latency only contains the execution and downlink latencies. Since the data transmission of smart vehicles has little effect on the performance of edge offloading, we neglect it in the rest of this paper.

E. Edge Resource Allocation

To improve the system performance, a fundamental challenge is how to utilize edge resources efficiently for better task offloading performance. Following the definition of network nodes in §IV-A, each regular vehicle demands one (multiple-server offloading is left for future investigation) edge server for complementary data processing power. We denote $a_{ij} \in \{0, 1\}$ as the offloading decision of edge server E_i regarding the computation task of regular vehicle V_j, defined as follows.

$$a_{ij} = \begin{cases} 1 & \text{if } V_j \text{ offloads to } E_i \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The allocation can be denoted by a $(0, 1)$-matrix as follows.

$$\mathbf{A_{rv,e}} = (\mathbf{a_1}, \mathbf{a_2}, ..., \mathbf{a_e}) = (a_{ij})_{rv \times e} \tag{6}$$

where rv, e denote the number of regular vehicle and edge servers (as defined in §IV-A), respectively. Based on Equation 1-5, the overall overhead of edge offloaded, $\mathbf{L^E_{\tau_V}}$ can be calculated as

$$\mathbf{L^E_{\tau_V}} = (\frac{C_V}{2} + \lambda) \sum_{j=1}^{e} |\mathbf{a_j}|^2 + \frac{3C_V}{2} rv, \tag{7}$$

where $\lambda = I_V/B^{ul} + O_V/B^{dl}$, $|\mathbf{a_j}|$ is the number of vehicles offloading to E_j. Please refer to Appendix A for the derivation of eq. (7).

V. Problem Formulation

The overall goal of optimization is to maximize the summed priorities of the valid feedback results that are returned in time while minimizing the overall delay. We denote b_{ij} as the validity matrix of feedback results:

$$b_{ij} = \begin{cases} 1 & \text{if } L^{E_j}_{\tau_{V_i}} \leq D_{V_i} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where $\lambda|\mathbf{a_j}|$ is the offloading delay of E_j. Based on eq. (7), the problem can be formulated as

$$\begin{aligned} &G1: Maximize \ \sum_{i=1}^{rv}\sum_{j=1}^{e} a_{ij} b_{ij} P_{V_i} \\ &G2: Minimize \ \mathbf{L^E_{\tau_V}} \\ &s.t. \ C1: \forall i \in [1, rv], \sum_{j=1}^{e} aij = 1 \\ &s.t. \ C2: \sum_{j=1}^{e} |\mathbf{a_j}| = rv \end{aligned} \tag{9}$$

Constraint C1 and C2 guarantee that each vehicle offloads to one and only one edge server at any timepoint. In other words, a regular vehicle always has an offloading edge server even during peak period. However, an edge server with a long processing buffer queue will drop the tasks that are already expired without execution. The minimization problem (G2) equals to the following:

$$Minimize \ \sum_{j=1}^{e} |\mathbf{a_j}|^2 \tag{10}$$

According to Cauchy–Schwarz inequality [37], we know

$$(\sum x_i y_i)^2 \leq (\sum x_i^2)(\sum y_i^2) \tag{11}$$

Set $y_i = 1, \forall i$, we get

$$(\sum x_i)^2 \leq (\sum x_i^2) \tag{12}$$

Together with C2, we get the lower bound of $\mathbf{L^E_{\tau_V}}$:

$$\begin{aligned} &\min \mathbf{L^E_{\tau_V}} = (\frac{C_V}{2} + \lambda) rv^2 + \frac{3C_V}{2} rv \\ &s.t. \ C1: |\mathbf{a_j}| = \frac{rv}{e}, \forall j \in [1, e] \end{aligned} \tag{13}$$

which indicates that G2 is achieved when tasks are equally distributed to edge servers.

We briefly describe a polynomial-time algorithm adapted from global early deadline first (EDF) as a centralized solution of G1 and G2 in Appendix B. Nevertheless, as described in the key point 3 in §IV-C, more improvements are required to encounter the affect of additional latency brought by centralized scheduler. Therefore, we propose a decentralized algorithm to address this challenge in the next section.

Algorithm 1: Regular Vehicles Job Assignment

```
                                          ▷ Regular vehicle
map ← getEdgeServerMap(location)
procedure threadUpdateEdgeServerMap:
    if location changed then
        map ← getEdgeServerMap(location)
    else
        sleep
procedure threadOffloadImageRecJob(job):
    if map has reachable servers then
        broadcast job to edge
    else
        send job to cloud
                                          ▷ Edge server
procedure threadReceiveJob(job):
    put job into execution queue
    broadcast job id and est. completion time
procedure threadReceiveUpdate(id, time):
    if job with id is in queue and
      its estimated completion time > time then
        if job is in execution then
            preempt the job
        else
            remove the job from queue
```

VI. Algorithm

In this section, we present a detailed solution and algorithms to tackle the challenges described in §IV and §V.

An essential feature of the problem, as stated in key point 2 at §IV-C, is that *we do not assume prior knowledge for task priority assignment*, which is a widely adopted assumption in most related works. We include such feature for a good reason, i.e., most data learning tasks such as NLP or image processing can only tell the value (meaning) of the data after processing. However, the value contained in the data is critical for priority (deadline) assignment. To tackle this challenge, we propose to use a mini-batch of historical job priorities as the reference to assign priorities to newly-arrived jobs. For instance, for the setup of batch size 3, the scheduler averages the priorities of the previous three jobs and assigns the priority to the next job using the average value.

We start by addressing the assignment problem of edge server resource. We define a set of nearby edge servers as an *edge server pool*. The servers in a pool have wired connections to each other and can communicate with ignorable latencies. Each pool does not overlap with any other and is assigned a different broadcast IP address. In this work, we rely on the cloud, which executes a service responsible for providing vehicles with the information of their nearby edge server pools. The number of edge servers within each area depends on the server placement strategy and the traffic condition. Please refer to §VIII-D for the details. Technically, based on their location and movement trajectories, vehicles preload from cloud service maps with the locations of edge server pools and select the closest one in the heading direction as the offloading candidate. Given that at least one available edge server pool is within the

wireless link reach, the vehicle broadcasts the job using its wireless interface. There might be multiple edge servers in the pool available to execute the job. To avoid a situation where multiple servers schedule the job for execution, we introduce a decentralized algorithm as shown in Algorithm 1. When the edge server receives a job, it assigns priority to it using the mini-batch average and broadcasts its decision on the wired interface (that is used for communication between edge servers). The job is then placed in a special purpose *wait* queue, which is executed only when the main queue runs out of jobs. In the process of conflict resolution, the server having the shortest main queue wins, and other edge servers drop the task from their wait queues. Finally, the winning edge server places the job in its main preemptive queue. In comparison with the centralized scheduling solutions, the proposed decentralized algorithm suits better V2X offloading scenarios thanks to two of its advantages: 1) It avoids to present a single point of failure. Algorithm 1 distributes the communication between several edge servers while centralized solutions would fail if the scheduler malfunctions. 2) It has a lower signaling overhead. Algorithm 1 requires the edge servers to exchange "estimated completion time", while centralized solutions require the scheduler to collect the waiting queue lengths and job priorities. Next, we introduce the detailed policy that the system uses to prioritize data.

A. Prioritization Policy

Most safety applications require a vehicle to broadcast its own information to all the vehicles within the neighbourhood. However, in most cases, a vehicle only needs information from some vehicles within the immediate neighbourhood. Therefore, it is crucial to design the data sharing policy to avoid unnecessary information processing. This is highly important for assisted driving where the goal is to provide the driver with only valuable information in real-time. For instance, an ARHUD filled with icons representing all nearby vehicles and RSUs can confuse the driver instead of providing assistance. Similarly, propagating nonrelevant information will lead to network congestion and increased latency. We propose an information prioritization policy to filter the data and select the most valuable one to display.

We assume the processing result of each task, O_V, has a list of *objects*, each of which refers to metadata that has some level of importance to the client. More specifically, in our use case, the processing result of each image frame contains a list of detected objects, each of which has a priority determined by its corresponding information, e.g., distance, relevance and catalogue (pedestrian, vehicle, bird). Each *object* is defined as $M_i \triangleq (P_i, I_i)$, where P_i and I_i denote the priority and the corresponding information of object i. Let $\mathbf{M}_V = \{M_1, M_2, ..., M_m\}$ denote the object list in O_V. The priority of τ_V is calculated as the summed priorities of the object, i.e., $P_V = \sum_{i=1}^{m} P_i$.

Furthermore, the priority of an object i is defined as $P_i \triangleq \rho(DI_i, RE_i, IM_i)$, where DI, RE, IM denote the *distance, relevance* and *importance* of the object. The key assumption is the accurate localization of vehicles and detected objects. As the relevant technologies such as RTK GNSS and Depth Estimation develop, we envision that self-localization and relevant coordinates of detected objects will achieve fine accuracy in the foreseeable future.

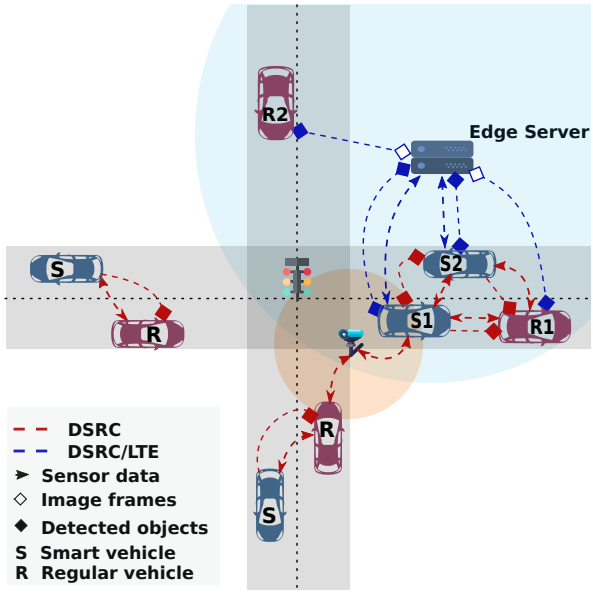


Fig. 4: EAVVE prioritized transmission.

Distance. We filter out data sent from outside the nearby area, defined by the radius of concern. For instance, in Figure 4, the edge server receives the images captured by $R2$ and broadcasts the extracted data. $S1, S2, R1$ filter it out as $R2$ is far away and the likelihood of the messages containing immediately relevant information is lower.

Relevance. We rank the messages filtered by distance according to their *relevance*. A message is considered more relevant if the receiver is currently moving towards the location of the sender. For a moving vehicle, the most relevant message is sent from the closest leading vehicle (or closest RSU which is in front of the vehicle's moving direction); this can be estimated from the moving direction and GPS coordinates of the sender (included in the message) and the current location and direction of the receiver. The messages from other leading vehicles and the following ones have lower relevance. The ones from vehicles in the opposite direction have even lower relevance. For instance, $S1$ receives data from $R1$ and $S2$ and prioritizes the former, since $S2$ is moving in the opposite direction on another lane, thus its observations are less likely to be relevant compared to $R1$.

Importance. The system classifies each detected object into a preference list according to its importance. The preference definition follows common society rule while may slightly vary in different areas. A basic sample rule we utilize is, for instance, $pedestrian > bicyclist > vehicle > obstacle$. Among the huge number of detected objects, an edge server and smart vehicle should select the most important ones and broadcast their existence in priority.

B. Data Filtering Algorithms

EAVVE's architecture relies on parallel threads running simultaneously to handle data in real-time. In this section, we discuss the details of the data filtering algorithms for the *sender device*, the *client device* and the *edge server*.

Sender Algorithm. Sender devices include smart vehicles and regular vehicles that differ in their data processing capabilities. Smart vehicles perform object detection on their OBU and broadcast the results directly. Regular vehicles rely on edge servers for object detection. Due to space limitation, we focus on smart vehicles. The detail of the algorithm is represented in Algorithm 2. A vehicle collects and broadcasts IMU data every 0.1 second by default (lines 4 and 14). Meanwhile, it takes photos and broadcasts the detection results periodically. In this work, the detection procedure runs in a *single sequence* (i.e., not multithreaded), to present the bottom-line performance (line 9). With multithreaded object detection, EAVVE is able to process images at higher frame rates, thus providing better service experience. To synchronize data transmission, the OBU adapts the count down timer of photographing according to the delay of each detection task (line 10). In this way, a vehicle makes its best effort to broadcast IMU data and detection result with the minimum time difference. However, due to networking issues and system performance differences, packets from different vehicles may arrive at a vehicle at different timestamps. The client device uses a buffer to solve this issue, as follows.

Client Algorithm. As shown in Algorithm 3, the client unit uses a buffer to patch together the received packets within a period $tBuff$ (line 4),

$$0 < tBuff \leq T - t(policy) \tag{14}$$

where $t(policy)$ is the policy execution latency with the average time complexity of $\mathcal{O}(1)$, which is about 1.2ms in our road test. $tBuff$ can be set to any value within the interval in eq. (14). The parallel threads push new data incoming within $tBuff$ into the tail of the buffer (lines 7 and 10), while displaying the data in the head of the buffer. Once a same period with sender broadcast, the client resets the buffer (line 3). Larger $tBuff$ potentially increase the amount of data to be displayed, while increasing the number of cross-thread operations correspondingly. In §VIII, $tBuff$ is set to 90ms to display more received data. As mentioned before, vehicles need to display only the most pertinent information by filtering the received data. EAVVE tackles this challenge via three steps:

(i) The system only processes the packet sent from within the concerned radius (line 6).

(ii) Two objects are considered as duplicates if their distance is less than the default tolerance deviation Γ (line 16). The tolerance deviation varies for a different type of object. The rule is that a larger object has a larger tolerance deviation. To minimize latency, OBU randomly chooses one of the duplicates to display.

(iii) Among the duplicates, IMU data always has the highest priority since it is more reliable and accurate than the detection results (line 20).

Algorithm 2: Smart Sender Algorithm

```
1  procedure threadIMU:
2      while (!/*break loop*/) do
3          count down T // T=0.1(s) by default
4          collect IMUdata
5          append IMUdata to imuList
6  procedure threadDetector:
7      while (!/*break loop*/) do
8          take image (IMG)
9          detect Objs // single sequence
10         count down (T - delayDetection)
11         add Objs to detectionList
12 procedure threadSend:
13     if imuList not empty then
14         broadcast IMUdata
15         reset imuList
16     if detectionList not empty then
17         broadcast detected Objs
18         reset detectionList
```

Algorithm 3: Client Algorithm

```
1  procedure threadPreprocess:
2      while (!/*break loop*/) do
3          reset Layer1Tmp, Layer2Tmp
4          while (t < tBuff) do // buffer
5              receive IMUdata and detected Objs
6              if (xSender, ySender) within rC then
7                  append imuData to Layer1Tmp
8              for Obj in Objs do
9                  if (xObj, yObj) within rC then
10                     append Obj to Layer2Tmp
11         reset t
12         count down (T - tBuff)
13 procedure threadDisplay:
14     while (!/*break loop*/) do
15         if Layer2Tmp not empty then
16             if D(Objs) ¡ Γ then
17                 Remove overlapped objs
18             Display Objs of Layer2Tmp
19         if Layer1Tmp not empty then
20             Display senders in Layer1Tmp on top
```

The edge server runs a mixed algorithm combining the *threadDetector* of the sender algorithm and *threadPreprocess* of the client algorithm. It offloads the object detection tasks from regular vehicles via *threadDetector*. Meanwhile, it aggregates IMU data and detection results from nearby vehicles by removing the duplicated data via *threadPreprocess*. It also periodically uploads the aggregated road situation to the cloud database or the traffic centre for traffic monitoring.

VII. Augmented Reality Applications for Vehicular Networks

In this section, we describe the details of the AR application we have implemented to showcase EAVVE. Augmented

Reality Head-up Displays (ARHUD) are a new technology that aims at assisting the driver by showing traffic information on the windshield of the vehicle. Currently, industrial and academic efforts focus on providing better visualization. EAVVE, on the other hand, enhances ARHUD with extensive information gathered from nearby vehicles. In the scenario of *V2E*, an edge server provides object detection services for nearby regular vehicles. An in-vehicle IoT device (e.g., a smartphone setup behind the windshield) works as both the sender device transmitting data to the edge server and the client device visualizing the received data. To detect objects and augment the user's view, the IoT device periodically uploads the sensor data and captured image frames to the edge server. The transmitted data includes the current street view image from the camera, GPS coordinates, motion and timestamp. Upon reception of the data, the edge server first executes object detection on the image with the object detector. With state-of-the-art deep learning frameworks and GPU hardware acceleration, the object detector can detect objects in real-time. In the deployment, we built the prototype using YOLO version 3 and OpenCV in a server with built-in Nvidia GTX 1070 GPU. Each image takes about 20 ms to process. For each detected object, a rectangular boundary (or a representative icon) is also returned by the detector. *In the case* of a smart vehicle with comprehensive data processing capabilities, the application runs on the OBU in a similar way. The application directly analyzes the images captured by the in-vehicle camera, rather than receiving them over the network as in the case with the edge server.

VIII. Evaluation

A. Implementation

In this subsection, we describe the implementation of EAVVE. We follow the system design and use case described in the previous sections to develop a connected vehicle vision system. This system detects pedestrians, cars, buses and traffic lights on camera images and shares the results among vehicles in real-time.

We deploy the object detector (Figure. 3) on a Linux platform with the GPU implementation of YOLO version 3, one of the current state-of-the-art object detection algorithms. We use OpenCV for general image processing such as perspective transformation (from one vehicle's to another one's). We implement the client device and part of the sender device (camera, IMU data collector) on the Android platform, to simulate the hardware and software environment of the vehicular equipment for augmented vision. The GPS sensor reports the GPS coordinates of the vehicle, and the monocular camera captures the front-facing view from the vehicle. We use OpenGL to render the augmented information on top of the camera view. Our current implementation operates with a monocular camera (on Android phone), while real vehicles will most probably embed stereoscopic cameras which will make many transformations easier. Despite this limitation, our prototype is fully functional and hence shows the potential of productisation.

(a) Leading vehicle.

(b) Following vehicle.

(c) Edge server screenshot.

Fig. 5: Road test device setup.

TABLE I: Communication set up

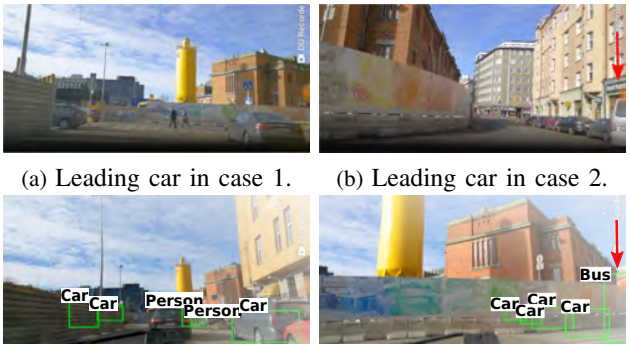
	Protocol	UL Freq	UL Bw	RTT
V2V/V2E	802.11ac	5180MHz	27Mbps	6.06ms
V2LC	4G LTE	2530MHz	12Mbps	50ms
V2RC	4G LTE	2530MHz	12Mbps	160ms

B. Prototype and Road Test

Prototype. We use the prototype described in §VIII-A. To emulate in-vehicle devices, we install the client-side application on two Huawei Mate9 Pro smartphones, each with a 2.4 (1.8) GHz octa-core HiSilicon Kirin 960 CPU and 4GB of memory. Our edge server is deployed on a MSI GS65 Stealth 8SG, each of which has a 6-core I7-8750H CPU, 32GB of memory, and an Nvidia RTX 2080 Max-Q GPU. The hardware capacity of our edge server is similar to a medium-priced commodity edge server in 2018 [38]. To compare the benefits of EAVVE to cloud computing, we create a virtual machine instance on the Google Cloud platform, with 6 vCPUs, 16GB of memory, and an Nvidia Tesla V100 GPU. We placed both phones in a vehicle behind the windshield, working as the in-vehicle camera, IMU and ARHUD, as shown in Figure 5.

Road Test. We drove two vehicles in downtown of a major city, one following the other. We perform the following four road tests with the networking setup shown in Table I.

- V2V: The test demonstrates communication between a smart vehicle and a regular vehicle. We put the backpack in the leading vehicle and connect it with the phone by tethering to imitate a smart vehicle that communicates with the following vehicle wirelessely. In this scenario, the leading smart vehicle processes the captured data and broadcasts the analysis result. The following regular



(a) Leading car in case 1. (b) Leading car in case 2.

(c) Following car in case 1. (d) Following car in case 2.

Fig. 6: Road test of CVV. The detector labels the detected objects with the white boxes and the detection error with red arrows. Note that in (c), the driver can see the pedestrians blocked by the leading vehicle. The driver can see the parked cars around the corner in (d).

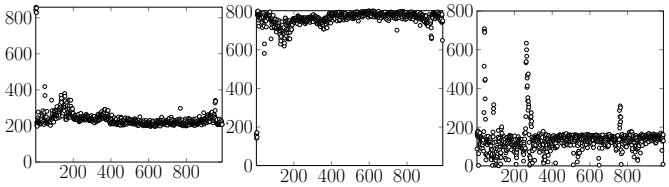
vehicle receives the result and renders it on ARHUD.

- V2E: We then put the backpack on the roadside to imitate an edge server. We set the edge server in Access Point (AP) mode so that the phones in both vehicles communicate with the edge server directly. We set the AP mode with IEEE 802.11ac and throttle the links to get a similar frequency and bandwidth with IEEE 802.11P (5.9GHz, up to 27Mbps). We show the potential improvement brought by newer technologies and infrastructures (e.g., edge servers co-located with base stations and DSRC) through simulations.
- V2LC (Vehicle to Local Cloud): As a comparison, we evaluate the performance of a Google cloud virtual machine in the same local area as the object detector. The phones send the captured images to the cloud and receive the extracted data via standard LTE connections.
- V2RC (Vehicle to Remote Cloud that is about 2000 km away): Finally, we redo the previous test with a remote Google cloud virtual machine.

Each round takes around 20 minutes with around 36000 image transmissions at 30 fps.

C. AR Connected Vehicle Vision (CVV)

Showcase. Figure 6 illustrates two typical use cases of CVV, in which the leading vehicle identifies objects in its vision and transmits them to the following vehicle. Figure 6a shows the common scenario in which pedestrians cross the street unexpectedly. This scenario takes place on a narrow road, where the vision of the driver in the following car is easily blocked while the driver in the front car has a clear view (see Figure 6c). This situation can be potentially dangerous as the following vehicle cannot anticipate events happening in front of the leading vehicle, and has thus less time to brake in case the leading vehicle needs to stop urgently. CVV helps the driver in the following to be aware of the pedestrians blocked by the front car and slow down beforehand. In the second scenario, one vehicle has turned right at an intersection while the other one is also going to turn. The leading vehicle observed several parked cars on the roadside which block one lane. An accident could happen if the following vehicle turned into the blocked lane without driving carefully. With CVV, the driver of the following vehicle can also "see" the parked vehicles out of the line of sight around the corner, as shown by the green rectangles with name tags in Figure 6d. This enables the driver of the second vehicle to avoid turning into the blocked lane. In summary, EAVVE allows enhancing road safety by augmenting the driver's vision with information observed by another vehicle.



(a) V2LC (b) V2RC (c) V2E

Fig. 7: Image transmission latency. X-axis: image ID, Y-axis: latency (ms). We only show a part of the result due to space limitation, the other parts have similar patterns.

Measurements. We run the application in the rounds of road test and measure the step by step latencies. Figure 7 shows the image transmission latency of V2LC, V2RC and V2E. Table II shows the integrated step-by-step latency of each run. The numbers shown are averaged over the whole test round (20 minutes and around 36000 image transmissions and processings). The latency of CVV can be divided into the following components: *Client Data Collection*, *Uplink Latency (image transmission)*, *Object Detection*, *Policy Control*, *Downlink Latency (detection result transmission)* and *Display Rendering*.

As Table II shows, CVV with edge server is much faster than with the cloud. Due to the hardware differences, the algorithms run slightly faster on the cloud server than on the edge server. Nevertheless, when using an edge server, the sum of the uplink and downlink latencies decreases by over 132.9 ms compared with using local cloud service, which contributes to most of the time difference. In the case of a remote cloud service, this difference increases to 663.1 ms. This represents a 42.6% and 78.7% improvement over local and remote cloud service, with an overall latency below 200 ms. This affirms our core assumption that edge computing can significantly improve latency-sensitive workloads by performing data processing closer to the user. V2V has the lowest latency out of all solutions, thanks to its straight data processing pipeline without depending on third-party servers. However, it does require a smart vehicle equipped with local processing capabilities.

As mentioned at the beginning of §VIII, DSRC and 5G have the potential to further improve the performance of EAVVE. Both technologies can significantly decrease the communication latency of V2V and V2E, which will increase the benefit of EAVVE over current solutions even more. We have shown

TABLE II: Step by step latency (ms).

	Overall	Data Collection	Uplink	Processing	Downlink	Policy	Display
V2V	57.7	6.7	N/A	20.3	12.5	1.2	17
V2E	179.8	6.7	122.1	20.3	12.5	1.2	17
V2LC	311.9	6.7	233.0	19.3	34.5	1.4	17
V2RC	842.1	6.7	762.2	19.3	35.5	1.4	17

TABLE III: Number of base stations and macrocells in the selected area of each city.

	London	Nottingham	York
Base stations	4043	725	225
Macrocells	147	36	23

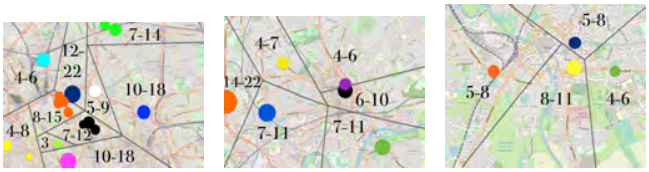


Fig. 8: Edge server placement based on traffic heatmap (London/Nottingham/York). The numbers indicate the min and max number of edge servers required to meet the request demand. The size of the dots relates to the amount of traffic in the area. Maximum number of required edge servers: 125, 67, 33.

Fig. 9: LTE base station distribution. Red dots indicate base stations with coverage $>$ 3000 m.

the obvious benefits of performing computation onboard or at the edge compared to local or remote cloud by showing that the improvement in processing time is not worth the increased communication latency. In the next section, we further expand on this model by discussing the edge server placement and estimate the deployment cost based on a real dataset.

D. Edge Server Placement

To address the edge server placement problem, we consider the base station and traffic distribution patterns in three cities with different population densities, London, Nottingham, and York (all in the UK). For each city, we focus on an area with a size of 4 km * 5 km around the city center. For each area, we use the public LTE base station location data[1] and the traffic volume data[2]. We cluster the traffic volume data according to the GPS coordinates of vehicles and divide the selected area into smaller areas according to the clustering result. The traffic distribution and area partition results are shown in Figure 8. Each colored dot represents the location of the aggregated traffic, with size proportional to the traffic volume in 12 hours during the daytime. In each area, we display the number of deployed edge servers according to the average and peak traffic volume. We then analyze the relationship between the base station distribution and the traffic density, as it influences our co-located edge server placement. Table III shows the number of base stations located in each area. Some base stations have coverage radius larger than 3km; we denote these as "macrocells". We plot these macrocells in red and the others in blue, as shown in Figure 9. The base stations are distributed evenly and reasonably match the density of the traffic. As a result, using base stations as deployment points is not likely going to deviate the edge server placement from the actual traffic patterns. The maximum number of edge servers required to meet the user demand during peak traffic is 125, 67, 33 for each area in London, Nottingham, and York, respectively. This corresponds to respectively 6.3, 3.45 and 1.8 edge server per square kilometer, a reasonable infrastructure investment for high availability, real-time edge computing. As such, we mainly deploy the edge servers within the macrocells located closer to the aggregated traffic.

To evaluate the scalability of our system, we test the system performance in various scenarios. Among the three major parts of EAVVE, V2C and V2V respectively depend on the Internet connection and smart vehicles. V2E, on the other hand, needs to tackle with edge server placement and vehicular traffic density. Therefore, we focus on evaluating the performance of V2E. We reuse the setup described in section VIII-D.

The number of vehicles appearing in the area each hour from 7am to 6pm (12 hours) increases from 7589 to 12948, 8920 to 10923, 3827 to 5596 in the selected areas of London, Nottingham and York, respectively. We simulate the traffic scenarios using `Veins` [39]. Veins is an open source framework for running vehicular network simulations. It is based on `OMNeTpp`, an event-based network simulator, and `SUMO`, a road traffic simulator. Each edge server has a coverage area of 3km, similar to its co-located macrocell. The public dataset we used only contains vehicle counts per hour. To add more granularity, we randomly generate traffic within each area based on the corresponding data collected from real dataset (see §VIII-D). The simulated traffic thus corresponds to the real traffic at a macroscopic level. In the simulation, all vehicles are regular vehicles without image processing

[1]https://unwiredlabs.com
[2]https://data.gov.uk/dataset/gb-road-traffic-counts

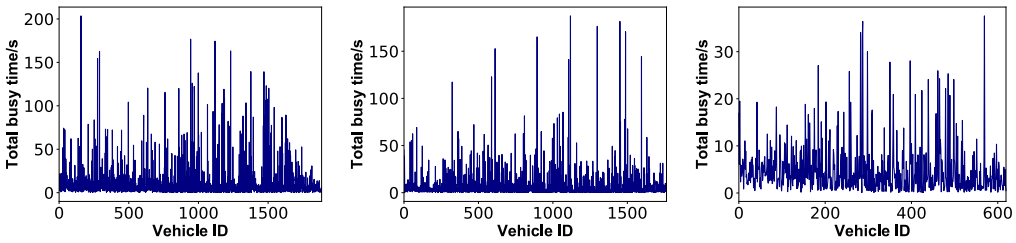


Fig. 10: Total busy time (London/Nottingham/York).

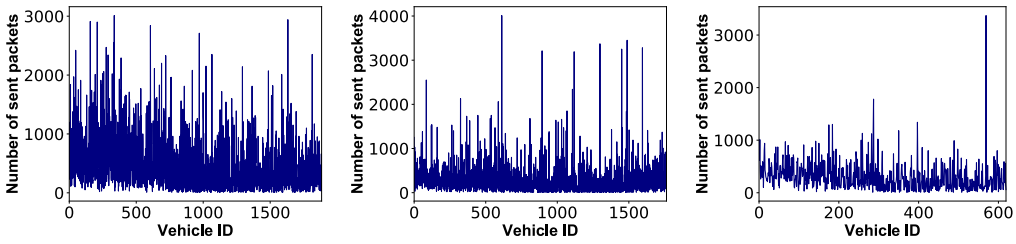


Fig. 11: Number of sent packets (London/Nottingham/York).

TABLE IV: Statistics in Figure 10 and Figure 11.

	London	Nottingham	York
Driving time (99% CI)	464.39s (437.87-490.90)	291.08s (268.65-313.51)	290.87s (264.79-316.94)
Busy time (99% CI)	11.46s (10.27-12.65)	7.55s (6.59-8.51)	4.84s (4.27-5.42)
Sent packets(99% CI)	476.11 (449.40-502.82)	301.88 (279.45-324.32)	305.26 (276.44-334.08)

TABLE V: Minimum and maximum bandwidth requirements at vehicle, edge and cloud level for EAVVE during a day.

	London	Nottingham	York
Vehicle (Min-Max)	528 bps - 500 kbps	528 bps - 500 kbps	528 bps - 500 kbps
Edge Server (Min-Max)	3.19 Mbps - 5.44 Mbps	3.42 Mbps - 5.44 Mbps	3.72 Mbps - 5.44 Mbps
Cloud Server (Min-Max)	402.26 Mbps - 687.61 Mbps	229.48 Mbps - 365.02 Mbps	127.44 Mbps - 186.05 Mbps

capabilities. In the road tests we let the system update at 10 images per second. At this frame rate, the display does not provide a smooth transition of frames as the driver observes the objects flickering on ARHUD and finds it hard to capture all the information displayed. Therefore in the simulation, we let each vehicle send captured images continuously to its nearest edge server at 1 image per second.

E. Scalability

Since all the transmissions in vehicle-to-edge and edge-to-vehicle are broadcast, there may be congestion on the transmission channel. Depending on the vehicular traffic volume, this may generate *busy times* which affect the vehicle's communication to the edge server or vice versa. When a vehicle finds a channel busy, it waits for a specified time period and then tries to retransmit. In our simulations, we did not specifically attempt to discover an optimal back-off strategy and instead relied on the well-known exponential back-off [40]. Figure 10 shows the total busy time experienced by each vehicle. On the x-axis, we show all vehicles in the simulation and the value on the y-axis shows the number of seconds during which the vehicle experienced a busy channel. Figure 11 shows the total number of packets sent by each vehicle. The first 10 minutes of the simulation is considered as a warm-up. We compute measurements after these 10 minutes. The average vehicle speed in all scenarios is under 10km/h, thus reflecting the lower bound of system performance under extremely congested scenarios. Table IV summarizes the statistics with 99% confidence interval and shows that EAVVE scales well to different scenarios. For instance, even though London and Nottingham tend to have higher amounts of busy time than York due to more traffic, the fraction of overall time a vehicle experiences a busy channel remains reasonable. The channel is busy only a few percents of the time (1.66%-2.59%). Moreover, the numbers of packets sent align with the driving time periods. Vehicles do not spend

much time in the back-off state, thus proving that network congestion is negligible. Therefore, EAVVE's V2E strategy scales well to central city areas with different densities of traffic[3].

Table V displays the minimum and maximum bandwidth requirements for our demo application over the course of a whole (simulated) day. During peak hours, a cloud server would require more than 680 Mbps for a central city area (London). When using V2E with edge servers, the load at the co-located base station is around 5 Mbps, which is achievable on LTE networks. V2V requires about 500 kbps per vehicle, achievable on both LTE and DSRC. Overall, compared with V2C, V2E can considerably relax the load at the bottleneck (up to 99%) while providing computing capabilities to regular vehicles.

In summary:

- Our edge server placement strategy leverages the base station distribution and matches the demand with the vehicular traffic patterns well.
- EAVVE improves the AR applications in vehicular networks by decreasing the transmission latency.
- EAVVE is scalable and performs well in different traffic densities. It can also be combined with cloud solutions to optimize costs.

IX. Conclusion

In this paper, we present EAVVE, an architectural framework for vehicle-to-everything applications. Our system exploits the low latency of edge servers to provide real-time emergency detection and notification. Leveraging edge servers at various points in the network and future in-car processing power, EAVVE is able to provide contextual information sharing service for a variety of uses at different scales in time and space based on the filtering and prioritization mechanisms. To validate the concepts behind EAVVE, we build a prototype application that we evaluate through real-world experiments. This application connects vehicular vision with an ARHUD for more comprehensive accident avoidance. In addition, using real traffic data from different cities in the UK, we show that EAVVE is able to scale up to realistic traffic demands and that the required number of edge servers remains manageable, even for large metropolis, e.g., 6.3 edge servers per square kilometre in central London. Our road test results show that, compared to cloud solutions, EAVVE decreases the latency of AR applications in vehicular networks, that is, 42.6% and 78.7% for Connected Vehicle Vision compared with local and remote cloud services. We also investigate the scalability of EAVVE and show that it decreases latency in realistic scenarios for different traffic densities. In summary, EAVVE is an efficient V2X solution which improves the performance by decreasing user latency and reducing network traffic.

[3]To accelerate simulation, we set the traffic update interval as 10s, which leads to the difference of number of sent packets and driving time (some packets sent by the vehicles after they drove outside the area were also counted, which has a negligible effect).

Appendix A
Derivation of eq. (7)

Let $queue_i$ denote the queue of tasks in front of task i, τ_i. At any time, any edge server E_j has a queue of which the length fulfills:

$$0 \leq |queue_i| \leq rv, \forall V_i \in \mathbf{V_{E_j}} \tag{15}$$

where $\mathbf{V_{E_j}}$ denotes the set of vehicles offloading to edge server E_j as defined in §IV-A. We then deduce

$$\begin{aligned}
\mathbf{L}^{\mathbf{E}}_{\tau_\mathbf{V}} &= \sum_{i=1}^{rv}\sum_{j=1}^{e} aij \cdot L^{E_j}_{\tau_{V_i}} \\
&= \sum_{i=1}^{rv}\sum_{j=1}^{e} aij \cdot (T^{off}_{\tau_{V_i}} + T^{back}_{\tau_V} + T^{exec}_{\tau_{V_i}}) \\
&= \sum_{i=1}^{rv}\sum_{j=1}^{e} aij \cdot (\frac{I_V}{R^{ul}_{E_j}} + \frac{O_V}{R^{dl}_{E}} + C_V + \sum_{v \in queue_i} C_V) \\
&= \sum_{i=1}^{rv}\sum_{j=1}^{e} aij \cdot (\frac{I_V|\mathbf{a_j}|}{B^{ul}} + \frac{O_V|\mathbf{a_j}|}{B^{dl}} + C_V + \sum_{v \in queue_i} C_V) \\
&= \lambda \sum_{i=1}^{rv}\sum_{j=1}^{e} aij|\mathbf{a_j}| + C_V \sum_{i=1}^{rv}\sum_{j=1}^{e} aij + C_V \sum_{i=1}^{rv}\sum_{j=1}^{e} aij|queue_i| \\
&= \lambda \sum_{j=1}^{e} |\mathbf{a_j}|^2 + C_V \sum_{j=1}^{e} |\mathbf{a_j}| + C_V \sum_{j=1}^{e} (0 + 1 + 2 + ... + |\mathbf{a_j}|) \\
&= \sum_{j=1}^{e} (\frac{C_V}{2} + \lambda)|\mathbf{a_j}|^2 + \frac{3C_V}{2}|\mathbf{a_j}| \\
&= (\frac{C_V}{2} + \lambda) \sum_{j=1}^{e} |\mathbf{a_j}|^2 + \frac{3C_V}{2} \sum_{j=1}^{e} |\mathbf{a_j}| \\
&= (\frac{C_V}{2} + \lambda) \sum_{j=1}^{e} |\mathbf{a_j}|^2 + \frac{3C_V}{2} rv,
\end{aligned} \tag{16}$$

where $\lambda = \frac{I_V}{B^{ul}} + \frac{O_V}{B^{dl}}$.

Appendix B
Polynomial proof of eq. (9)

To meet the constraint in eq. (13) (G2) while maximizing the priorities of valid results (G1), we adapt the algorithm of global early deadline first (EDF) to tackle the challenges. In each edge server pool (defined in §VI), an edge server functions as the leader and runs the scheduling algorithm. It continuously collects the statistics from other edge servers in the pool, including the waiting queue lengths and the priorities of historical jobs. Meanwhile, it listens to the beacons sent out by the vehicles and matches each vehicle's identity with the priorities of jobs using the UID contained in the beacon and job packets. As such, the leader maintains the updates of nearby vehicles and their historical job priorities. The centralized scheduler sorts all the jobs sent from serving vehicles from high to low according to their priorities. Meanwhile, it sorts the available threads of the edge servers from short to long according to their queue lengths. It schedules the

jobs with the earliest deadline, each to a different thread following the order of the thread sequence, and then the jobs with the next earliest deadline and so on iteratively until all threads are busy or all jobs are scheduled. The algorithm schedules as many jobs with the highest priorities as possible while balancing the offloading jobs among the servers with the maximum effort. The algorithm described above achieves an optimal scheduling solution with the time complexity of $\mathcal{O}(n \log n)$ utilizing a sorting algorithm such as merge sort [41].

REFERENCES

[1] THEVERGE, "California green lights fully driverless cars for testing on public roads," 2018, accessed 2018-06-15. [Online]. Available: https://www.theverge.com/2018/2/26/17054000/self-driving-car-california-dmv-regulations

[2] CHINADAILY, "Shanghai allows autonomous tests," 2018, accessed 2018-06-15. [Online]. Available: http://www.chinadaily.com.cn/business/motoring/2017-11/13/content_34469664.htm

[3] S. Biswas, R. Tatchikou, and F. Dion, "Vehicle-to-vehicle wireless communication protocols for enhancing highway traffic safety," *IEEE communications magazine*, vol. 44, no. 1, pp. 74–82, 2006.

[4] J. Gozálvez, M. Sepulcre, and R. Bauza, "Ieee 802.11 p vehicle to infrastructure communications in urban environments," *IEEE Communications Magazine*, vol. 50, no. 5, pp. 176–183, 2012.

[5] S. Rangarajan, M. Verma, A. Kannan, A. Sharma, and I. Schoen, "V2c: a secure vehicle to cloud framework for virtualized and on-demand service provisioning," in *Proceedings of the International Conference on Advances in Computing, Communications and Informatics*. ACM, 2012, pp. 148–154.

[6] K. Abboud, H. A. Omar, and W. Zhuang, "Interworking of dsrc and cellular network technologies for v2x communications: A survey," *IEEE transactions on vehicular technology*, vol. 65, no. 12, pp. 9457–9470, 2016.

[7] Y. Ren, F. Liu, Z. Liu, C. Wang, and Y. Ji, "Power control in d2d-based vehicular communication networks," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 12, pp. 5547–5562, 2015.

[8] S. Guleng, C. Wu, Z. Liu, and X. Chen, "Edge-based v2x communications with big data intelligence," *IEEE Access*, vol. 8, pp. 8603–8613, 2020.

[9] C. Wu, X. Chen, T. Yoshinaga, Y. Ji, and Y. Zhang, "Integrating licensed and unlicensed spectrum in the internet of vehicles with mobile edge computing," *IEEE Network*, vol. 33, no. 4, pp. 48–53, July 2019.

[10] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 450–465, Feb 2018.

[11] J. Feng, Z. Liu, C. Wu, and Y. Ji, "Mobile edge computing for the internet of vehicles: Offloading framework and job scheduling," *IEEE vehicular technology magazine*, vol. 14, no. 1, pp. 28–36, 2018.

[12] ——, "Ave: Autonomous vehicular edge computing framework with aco-based scheduling," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 12, pp. 10 660–10 675, 2017.

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

[14] O. Ascigil, T. K. Phan, A. G. Tasiopoulos, V. Sourlas, I. Psaras, and G. Pavlou, "On uncoordinated service placement in edge-clouds," in *Cloud Computing Technology and Science (CloudCom), 2017 IEEE International Conference on*. IEEE, 2017, pp. 41–48.

[15] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 1, pp. 856–868, Jan 2019.

[16] K. Zhang, Y. Mao, S. Leng, Y. He, and Y. ZHANG, "Mobile-edge computing for vehicular networks: A promising network paradigm with predictive off-loading," *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36–44, June 2017.

[17] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control," *IEEE Transactions on Computers*, vol. 66, no. 5, pp. 810–819, 2016.

[18] T. G. Rodrigues, K. Suto, H. Nishiyama, N. Kato, and K. Temma, "Cloudlets activation scheme for scalable mobile edge computing with transmission power control and virtual machine migration," *IEEE Transactions on Computers*, vol. 67, no. 9, pp. 1287–1300, 2018.

[19] K. Wang, H. Yin, W. Quan, and G. Min, "Enabling collaborative edge computing for software defined vehicular networks," *IEEE Network*, vol. 32, no. 5, pp. 112–117, 2018.

[20] F. Tang, Y. Kawamoto, N. Kato, and J. Liu, "Future intelligent and secure vehicular network toward 6g: Machine-learning approaches," *Proceedings of the IEEE*, 2019.

[21] C. Wu, Z. Liu, D. Zhang, T. Yoshinaga, and Y. Ji, "Spatial intelligence toward trustworthy vehicular iot," *IEEE Communications Magazine*, vol. 56, no. 10, pp. 22–27, 2018.

[22] X. Chen, C. Wu, T. Chen, H. Zhang, Z. Liu, Y. Zhang, and M. Bennis, "Age of information-aware radio resource management in vehicular networks: A proactive deep reinforcement learning perspective," *IEEE Transactions on Wireless Communications*, 2020.

[23] S. Chen, J. Hu, Y. Shi, Y. Peng, J. Fang, R. Zhao, and L. Zhao, "Vehicle-to-everything (v2x) services supported by lte-based systems and 5g," *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 70–76, 2017.

[24] H. Qiu, F. Ahmad, R. Govindan, M. Gruteser, F. Bai, and G. Kar, "Augmented vehicular reality: Enabling extended vision for future vehicles," in *Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications*. ACM, 2017, pp. 67–72.

[25] H. Kim, X. Wu, J. L. Gabbard, and N. F. Polys, "Exploring head-up augmented reality interfaces for crash warning systems," in *Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*. ACM, 2013, pp. 224–227.

[26] P. Zhou, W. Zhang, T. Braud, P. Hui, and J. Kangasharju, "Arve: Augmented reality applications in vehicle to edge networks," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*. ACM, 2018, pp. 25–30.

[27] ——, "Enhanced augmented reality applications in vehicle-to-edge networks," in *2019 22nd Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN)*. IEEE, 2019, pp. 167–174.

[28] J. B. Kenney, "Dedicated short-range communications (dsrc) standards in the united states," *Proceedings of the IEEE*, vol. 99, no. 7, pp. 1162–1182, 2011.

[29] A. Papathanassiou and A. Khoryaev, "Cellular v2x as the essential enabler of superior global connected transportation services," *IEEE 5G Tech Focus*, vol. 1, no. 2, pp. 1–2, 2017.

[30] SAE, "Dedicated short range communications (dsrc) message set dictionary," 2016. [Online]. Available: https://www.sae.org/standards/content/j2735_201603/

[31] L. Le, R. Baldessari, P. Salvador, A. Festag, and W. Zhang, "Performance evaluation of beacon congestion control algorithms for vanets," in *2011 IEEE Global Telecommunications Conference-GLOBECOM 2011*. IEEE, 2011, pp. 1–6.

[32] F. Librino, M. E. Renda, and P. Santi, "Multihop beaconing forwarding strategies in congested ieee 802.11 p vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 9, pp. 7515–7528, 2015.

[33] H. P. Luong, M. Panda, H. L. Vu, and B. Q. Vo, "Beacon rate optimization for vehicular safety applications in highway scenarios," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 524–536, 2017.

[34] H. Song and H. S. Lee, "A survey on how to solve a decentralized congestion control problem for periodic beacon broadcast in vehicular safety communications," in *2013 15th International Conference on Advanced Communications Technology (ICACT)*. IEEE, 2013, pp. 649–654.

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

[36] C. Hoymann, "Mac layer concepts to support space division multiple access in ofdm based ieee 802.16," *Wireless Personal Communications*, vol. 36, no. 4, pp. 363–385, 2006.

[37] J. M. Steele, *The Cauchy-Schwarz master class: an introduction to the art of mathematical inequalities*. Cambridge University Press, 2004.

[38] DELL, "Poweredge c4130 rack server optimized for gpus and co-processors," 2018. [Online]. Available: https://www.dell.com/en-us/work/shop/povw/poweredge-c4130

[39] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, January 2011.

[40] R. M. Metcalfe and D. R. Boggs, "Ethernet: Distributed packet switching for local computer networks," *Communications of the ACM*, vol. 19, no. 7, pp. 395–404, 1976.

[41] J. Katajainen and J. L. Träff, "A meticulous analysis of mergesort programs," in *Italian Conference on Algorithms and Complexity*. Springer, 1997, pp. 217–228.