Enhanced Augmented Reality Applications in Vehicle-to-Edge Networks

Pengyuan Zhou*, Wenxiao Zhang†, Tristan Braud†, Pan Hui*†, Jussi Kangasharju*
*Department of Computer Science, University of Helsinki
†Department of Computer Science and Engineering, The Hong Kong University of Science and Technology

***Abstract*—Vehicular communication applications, be it for driver-assisting augmented reality systems or fully driverless vehicles, 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. In this paper, we present EARVE, a novel Vehicle-to-Edge infrastructure, with computational units co-located with the base stations and aggregation points. Embedding computation at the edge of the network allows to reduce the overall latency compared to vehicle-to-cloud and significantly trim the complexity of vehicle-to-vehicle communication. We present the design of EARVE and its deployment on edge servers. We implement EARVE through a bandwidth-hungry, latency constrained real-life application. We show that EARVE reduces the latency by up to 20% and the bandwidth at the server by 98% compared to cloud solutions at city scale.**

I. Introduction

Automated driving has recently gotten closer to becoming a reality. In 2018, California and Shanghai authorized the deployment of autonomous vehicles on public roads for testing purposes [1], [2]. In parallel to automated driving, manufacturers are constantly improving the assistance systems embedded inside vehicles. These systems nowadays heavily rely on environment sensing for signaling potential danger to the driver and taking decisions if necessary. For instance, most manufacturers developed emergency braking systems for their top of the line vehicles. By combining information from the embedded radar and camera, the system can detect and prevent imminent collisions.

Although automated systems efficiently improve road safety, they are limited to the point of view of a single vehicle. However, some complex situations require assembling an aggregated point of view over several vehicles to avoid collisions. For instance, if the braking distance is too short to avoid a collision, the vehicle may choose another emergency maneuver such as steering into another lane. The system should request status from other vehicles in the area to assess the safety of the operation. Vehicular communication systems therefore play a key role in sharing information between vehicles and roadside infrastructure units (RSU). Current solutions focus on three types of communication: vehicle-to-vehicle (V2V), vehicle-to-cloud (V2C), and vehicle-to-roadside infrastructure (V2I) [3], [4]. Although these solutions fulfill basic demands, *efficiently sharing complex and large volumes of data among vehicles at scale* remains a challenge.

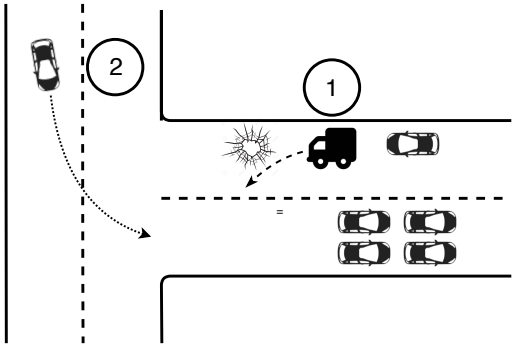

Figure 1: Common connected vehicles scenarios

Figure 1 illustrates two common scenarios, with their relative latency and scale requirements:

1) The leading truck encounters an unexpected pothole. The truck notifies the following cars to avoid a potential accident. *Real-time latency, street scale.*
2) Congested traffic is out of sight for cars planning to take the road on the right. Vehicles in the congestion broadcast to all the network so that cars can compute another optimal itinerary. *Medium latency, city scale.*

Current V2V, V2C, or V2I architectures and solutions cannot handle these scenarios due to the diversity of the requirements. In V2C, the combined latency of transmission, processing, and distribution prevents emergency decisions to be propagated on time. On the other hand, although V2V significantly improves performance at close distance, forwarding information at city-scale is inefficient and costly. Finally, V2I provides better data distribution. However, sharing accurate emergency information entails nontrivial computation and coordination in a limited amount of time. Roadside infrastructures should therefore integrate computing features for fast and reliable emergency information propagation.

Edge computing facilitates latency-sensitive workloads by performing data processing in Edge Servers (ESes) located close to the user. The gain in latency provided by edge computing can be considerable. In Table I, we measured the round trip latency for various servers through an LTE network. The first pingable IP is regarded as *Edge* server. We then

Edge	A	B	C
19.9 ms	24.9 ms	52.4 ms	58.8 ms

Table I: Average network round-trip latency over LTE to different targets

consider the closest cloud servers provided by three leading companies in the market: A (local to the city - 5 km), B (nearby country – 1000 km) and C (A country further away – 2500 km). Unsurprisingly, the latency to the closest cloud server is half the round trip time to the furthest cloud server. The ES presents a 20% improvement compared to the nearest cloud server. These 5 ms may become vital in the case of road safety, especially considering that latency is cumulative in the case of information propagation. For a V2C application residing at either site B or C, the latency improvement is considerable.

In this paper, we propose EARVE, the first *vehicle-to-edge (V2E)* framework to enhance vehicular communications. As an extension of our previous work [5], we propose the specific design of edge server and the interactions between server functionalities and the system (section III). We also implement a prototype and evaluate it with a preliminary experiment (section VI). Our design integrates both cloud and edge computing capabilities for city-wide autonomous and semi-autonomous vehicle communications. First of all, we stress that EARVE network-agnostic in terms of the physical layer on top of which it is run. Currently, two main alternatives exist for vehicular networks, Direct Short Range Communication (DSRC) and 5G [6]. While 5G seems to have its advantages [7], DSRC has already been adopted and deployed for tested solutions [8]. Since EARVE does not depend on any particular features of the underlying network, it can run on current (LTE) and future communication infrastructures seamlessly, while formulating recommendations for service provision and infrastructure deployment. As a concrete example, we apply EARVE to connected vehicle views by demonstrating the use cases, i.e., View Share, in Section IV.

This paper makes three key contributions.

- We present the design of EARVE, which equips edge RSUs with computation and cache capacity. We describe the implementation details and communication flows within the edge servers and between them.
- We describe a concrete application of EARVE to connect vehicle views using Augmented Reality Head-up Display (ARHUD). This use case displays the advantages of EARVE while scaling the problem of vehicle vision from a network perspective.
- We present an evaluation to show that EARVE meets the performance targets we have defined and that it offers noticeable performance improvements with reasonable expenditure in infrastructure.

The rest of paper is structured as follows. In Section II we present the overall system design. Section III provides the detailed description of the implementation of an edge server in EARVE. Section IV presents the two concrete use cases we consider in the paper. We give practical details of our implementation in Section V and present our evaluation in Section VI. We cover related work in Section VII. Finally, Section VIII concludes the paper.

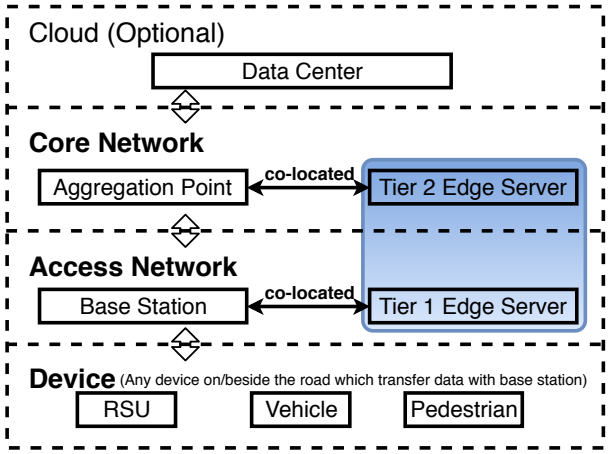

Figure 2: EARVE System Model.

II. System Design

In this section, we describe EARVE's design. First, we describe our proposed architecture for EARVE. We then discuss the major communication processes and propose a deployment scheme. Last, we discuss how EARVE can improve privacy and security in vehicular networks and AR in general.

A. System Architecture

EARVE is defined around three key layers: **device, access network and core network** as shown Figure 2. The **device layer** includes the vehicles, roadside buildings, infrastructures, phones of pedestrians and any other devices involved in the vehicular network. In the rest of this paper, we will call "client" any object in the device layer that transfers data to the ES. The **access network and core network layers** host the ESes at the core of our architecture. We distribute these ESes hierarchically in two tiers. The first tier (T1 ES) consists of ESes co-located with the base stations at access network level[1], while second tier ESes (T2 ES) are co-located with aggregation points in the core network. Finally, we define an optional cloud layer to provide on-demand backup capacity.

The ESes communicate with the vehicles and RSUs via the radio access network, and backs up data with the remote cloud if necessary (e.g., map updates). T1 ESes operate within an area defined by the range of their corresponding macrocell and eventual small cells (see II-C). As they are closer to vehicles, T1 ESes serve latency-sensitive applications such as emergency notifications. T2 ESes collect data from multiple areas (multiple T1 ESes) to provide a larger scale of service and data backup, e.g., to improve traffic flow by sending cruise control messages or controlling traffic light.

[1]Base station in this paper refers to the entity at the edge of the fixed network, e.g., BTS, eNB and gNB.

B. Communication Process

The communication process of EARVE follows 6 basic steps: *neighbor notification, data processing, transmission, dissemination, aggregation, and uploading.* In order to showcase this process, we consider an emergency notification application.

Neighbor notification: Emergency notifications need to be dispatched to the nearest clients with the lowest possible latency. The device detecting the emergency sends a detailed report to the nearest T1 ES. The report compiles the sensor data at the time of the incident with minimal compression to avoid losses and reduce processing times. Sensor data includes image frames, coordinates, velocity and motion direction of the device to help the ES calculate the coordinates of the incident.

Data processing: Once a T1 ES receives a report, it processes the data and caches the extracted information for passing on to later vehicles. As discussed in [9], sharing the views of incidents among vehicles is nontrivial. ESes maintain an up-to-date local map in real time, by collecting data from passing vehicles. This local map is then regularly synchronized with a cloud data center. T1 ESes serve as calibration points where the reported incident is mapped onto absolute coordinates before notification. ESes rely on machine learning based image analysis modules for extracting useful information from the reports. This data is then compared to a predefined rule set to determine the resulting actions. In the case of emergency notifications, ES compares the type and severity of the emergency with the rules and triggers the corresponding actions upon match.

Data transmission: Depending on the severity of the emergency, the T1 ES sends the notification to nearby T1 ESes on a scale defined by the matching rules. For instance, T1 ESes located within the same neighborhood are notified of events that may cause serious congestion.

Data dissemination: The top priority of each ES is to notify nearby vehicles and pedestrians. An ES needs to notify different groups of clients according to the triggered rules. For example, an ES notifies only the nearby vehicles of a "traffic congestion" event, while it notifies the nearest vehicles and pedestrians of a "severe accident" event.

Data aggregation: T1 ESes send data to T2 ESes for applications requiring larger amounts of data (only if aggregation is acceptable to the application). For instance, burst water pipes may cause severe flooding in multiple blocks. To get the whole picture, T2 ES needs to aggregate data sent from all involved T1 ESes within the damaged area.

Data upload: T1 and T2 ESes synchronize with the cloud to keep city-scale data up-to-date. T1 ESes also forward the emergency data to cloud for backup. This data may be used later for deploying new city-scale road security policies.

C. Deployment

Due to the space limitation, we will skip the details of deployment. Please refer to our previous work for more details [5].

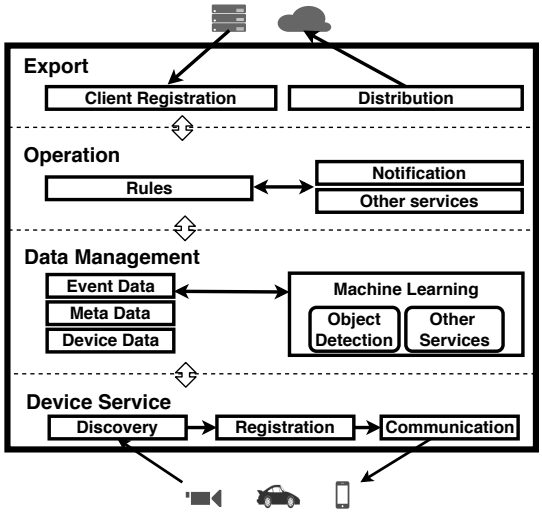

Figure 3: Edge Server Design.

D. Privacy and Security

V2V communication exposes each vehicle's private information to others. The mobility pattern of a driver is easily discovered which raises security and privacy concerns. Anonymization can hide the true identity of the sender, however, it raises issues in trust in the information shared in V2V networks. As a result, it is not easy to protect user privacy while providing trustful information distribution. Our system provides a preliminary solution for those problems with the help of ESes. As the information collection and distribution point, an ES collects data directly from users and distributes it after removing the private information of the original sender. As such, users share valuable information while hiding their identity from the larger public. Besides, our edge service follows a subscription mechanism, therefore only the users who trust and willing to use the edge service will share their information.

III. Edge Service

In this section, we describe the architecture of the services deployed at the edge. We first characterize the major data flows between server microservices. Then, we discuss the multithreading data process of ES and client device.

A. Server Architecture

To achieve low enough latency, ESes need an efficient data process flow with only the key microservices. On the other hand, the architecture needs to remain flexible and allow adding more microservices in the future. We design our ESes as shown in Figure 3. This architecture spreads among four major planes defined as follows:

The **Device Service** layer is the interface through which the ES communicates with different devices. Any client in need of edge service communicates with ES through this layer. At this layer, the major microservices include *Device Discovery*, *Registration* and *Communication*.

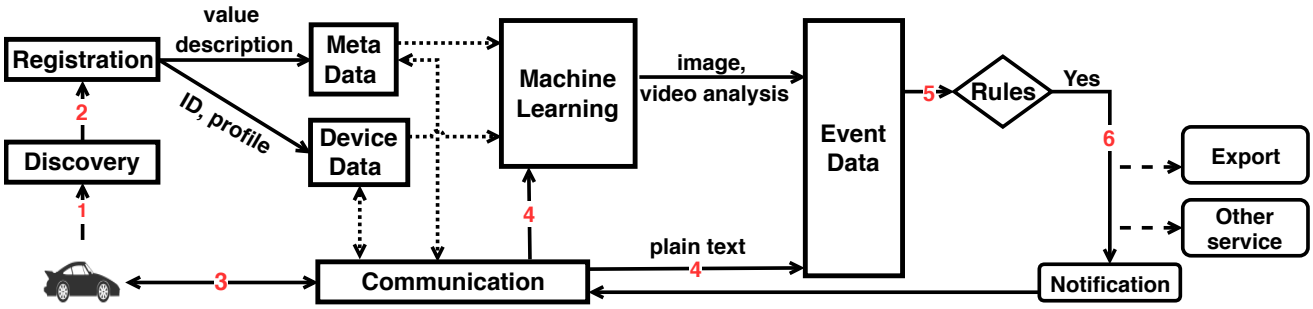

Figure 4: Data flow of Edge Server.

- *Device Discovery* allows the automatic discovery of devices entering the coverage area of the ES. We propose a subscription model so that the ES only communicates with the clients willing to use the edge service. The edge service provider reserves a specific IP address for service data transmission. Any device demanding edge services sends its basic information (e.g., ID, profile and coordinates) to the specific IP address periodically. The information is included in a message similar to the "HELLO" packet in OSPF for dynamic discovery of neighbors. The T1 ES within the base station identifies the information by filtering the destination IP address and registers the device. Users can adapt the frequency of messages and turn it on or off freely. We choose to use this model to give the user more flexibility in using the edge service.
- *Device Registration* generates a **unique ID for the discovered device and asks for its profile and value description**. The device profile is a descriptive file including all basic information of the device. This profile is composed of key-value pairs that describe the parameters potentially sent by the vehicle. For instance, a vehicle registers with the following description: {*V: velocity, mile/h; D: motion direction*}. Later the same vehicle going southbound at 30 miles/h will send {*V: 30; D: south*}.
- *Device Communication* transfers data with the device. The formats of data include plain text and image frames. The transport protocol is flexible and depends on specific use cases. In this work, we use UDP for low latency image transmission. However, for other applications, it may be needed to develop additional mechanisms to ensure reliable and ordered transmission while keeping the low latency requirements.

The **Data Management** layer analyzes and manages the data received via the Device Service layer. The major microservices include *Device Data*, *Meta Data*, *Event Data* and *Machine Learning*.

- *Device Data* stores the basic device data such as UID and profile for the ES to uniquely identify the device.
- *Meta Data* stores the values description and other metadata.

Timestamp	Coordinates	Type	Data	Device ID

Table II: Event entry

- *Event Data* stores the up-to-date events sent by devices. The database stores events as unique sorted entries (see Table II). The entries are organized in multiple levels and sorted from left field to right field (at the exception of "Device ID"). We base this ordering rationale on the scenario where multiple vehicles, pedestrians, and RSUs witness the same event or accident at the same time. To uniquely identify the event and ignore the duplicated reports, the "Timestamp" and "Coordinates" of the event have the highest priority for sorting. The "Type" field defines different types of events from normal map update to a severe accident report. Each event has a predefined TTL based on its type and gets removed from the database after expiration. The "Data" field contains all the key-value pairs from the device report. The "Device ID" is the unique ID of the sender and not included in the order of sorting, to avoid duplicated reports of the same event from different devices.
- *Machine Learning* analyzes the image frames sent by devices, extracts the key information and forwards it to the Event Data microservice. In this work, we integrate the machine learning algorithm of object detection into ES, which runs on GPU.

The **Operation** layer generates rules and applies them to the event data. The ES triggers a rule's actions if an event data matches the rule's condition(s). The major microservices include *Rules*, *Notification* and *others*.

- The *Rules* microservice generates new rules on-demand. Each rule has one or multiple conditions and actions. The ES matches each event data entry with the rule set and triggers its actions in case of a match. Upon detection, the ES triggers the transmission of notification and the export of data to neighbor clients. The
- *Notification* microservice sends out alerts/notifications to specific clients (groups) defined in the rule(s). For instance, if the event data contains an anomaly, the "Notification" service is triggered and the ES sends out notifications to all devices defined by the rule, e.g., all

vehicles discovered within 5 minutes.

The **Export** layer is responsible for exporting data to other ESes (T1 and T2) or the cloud if necessary. The major microservices include *Client Registration* and *Distribution*.

- *Client Registration* provides the interface for northbound clients to register to the system. In our system, northbound clients include nearby T1 ESes, T2 ESes, cloud data center and emergency center etc.
- *Distribution* sends to subscribers the event data based on the rules or historical statistics for backup.

Figure 4 depicts the major data process flow in the ES, which includes the following steps:

1) An ES discovers a device entering its coverage area.
2) The server registers the device and stores its basic information. The necessary information includes UID of device (generated by the ES), device profile (brief description of the device) and value description (key-value pairs to explain the meaning of value possibly sent by the device). The UID and profile of the device is stored in the Device Data microservice while value description is stored in Meta Data.
3) The ES communicates with the devices via the Communication microservice. It identifies each device by matching the device information with the corresponding Device Data. The ES allows a device to send plain text and image frames to report on different kinds of events.
4) Upon receiving plain text, the ES extracts the event data directly from the text and stores it. Upon receiving image frame, the ES analyzes it through machine learning algorithms and extracts the event data. In this paper, we only focus on one application of image processing: object detection.
5) After extracting the event data, the ES matches it with the rules and eventually triggers the corresponding actions.
6) The actions include notifications to vehicles, pedestrians or emergency centers, propagation to nearby ESes and cloud backup etc.

IV. Augmented Reality Applications for Vehicular Network

In this section, we describe the details of an AR application realized by V2E to showcase EARVE. We select an in-vehicle IoT device (e.g., HUD, smart rearview mirror) as our client device, and assume there is a T1 ES nearby the vehicle client and needs to provide object detection services.

View Share: To detect objects and achieve augmentation of the views, the client periodically uploads the sensor data and captured image frames to the ES, including the current street view image from the camera, GPS coordinates of the vehicle from the GPS receiver, the orientation of the vehicle from the IMU, and the timestamp. Together with the IoT device's IP address, the $\{$*camera image, GPS, orientation, timestamp, IP*$\}$ tuple is the major input of our system. Upon receiving the sensor data, the ES first executes object detection on the camera image with the object detector. With state-of-the-art deep learning frameworks and GPU hardware acceleration, the object detector is able to detect objects in real time. For each detected object, a rectangular boundary is also given by the detector.

With the object detection results, the ES calculates the GPS coordinates of the detected objects. The first step is calculating the relative positions of the objects from the vehicle. For each object, we choose the middle point of its bottom boundary as its position in the camera image. We need to transfer this position in pixels into position in meters, which is the object's position in the real world. To achieve this, we change the perspective of the camera image so that the new pixels correspond to the bird's-eye view from the top of the street. With the object's 2D position in the bird's-eye view image, we can calculate the object's relative position (both distance and direction) from the vehicle in meters after some calibration. With the vehicle's GPS information, the ES calculates the GPS for each detected object. Then all information is extracted as an event entry embedded into Event Data database. ES processes the data, applies the corresponding rules, and sends back $\{$*timestamp, object name, object GPS*$\}$ tuples to the clients specified by the rule's actions. When the client receives the message from the ES, it displays the detected objects in a manner of AR. With the object's GPS, the vehicle's GPS and orientation, the client is able to calculate the relative position (both distance and direction) of the object from itself. This relative position is transformed into 2D on the screen after perspective and unit transformation. With this design, drivers are able to see the objects that are hidden by front vehicles in real-time in an AR manner.

V. Implementation

In this section, we describe the implementation details of our system. We follow the use case described in the previous section and develop a simple object recognition system for vehicles. This system detects pedestrians and cars on pictures sent from the client device's embedded camera and returns the results in real-time. Our client is implemented on the Android platform, simulating the hardware and software environment of the vehicular equipment for augmentation. The GPS sensor reports the GPS coordinates of the vehicle, and the monocular camera captures the front-facing view from the vehicle. OpenGL is utilized for rendering the augmented information on top of the camera view. The communication between client and server is based on sockets with all the information packed in our own formats. The plain text and image frames sent by the client, as well as the message sent by the server, are transmitted over UDP socket for low-latency transmission.

Our server is deployed on a Linux platform. For object detection, we utilize the GPU implementation of YOLO version 3, which is the state-of-the-art object detector. We use OpenCV for general image processing like perspective transformation (from one vehicle's to another one's). For *ES implementation*, we build our prototype on top of EdgeX Foundry project, a vendor-neutral open source framework for IoT edge computing [10]. We use EdgeX as the skeleton

framework with proper adaption and add more microservices to build the ES for EARVE. For instance, to have the best knowledge of up-to-date events, we change the database maintenance mechanism of *Event Data* to multi-level uniquely sorting. We add the device discovery microservice and tune the communication module by adding UDP socket method. Moreover, we integrate machine learning modules into the ES architecture to improve its data analysis ability.

VI. Evaluation

We built a proof-of-concept system and now present our evaluation. To emulate an in-vehicle IoT device, we installed our client on a Xiaomi Mi5 smartphone, with a 2.15GHz quad-core Snapdragon 820 CPU and 4GB of memory. Our ES is deployed on a local Linux PC, with a six-core i7-5820K CPU, 64GB of memory, and an Nvidia GTX 1080Ti GPU. The hardware specification of our ES is similar to a medium-priced edge server in 2018 [11]. To compare the benefits of EARVE to cloud computing, we create a virtual machine instance on the Google Cloud platform, with 6 vCPUs, 16GB of memory, and an Nvidia K80 GPU. The phone is connected to the Internet through a WiFi access point on the ES. As such, our ES deployment works also as a "base station" from the perspective of the phone client, which follows our deployment proposal, namely co-locating ES with the base station.

The RTT from the phone to the ES is 6.84ms, and the RTT from our ES to the Google cloud virtual machine is 28.76ms. More than 90% vehicles driving at 100 km/h have only 7.6 milliseconds RTT in LTE network [12]. Besides, vehicles normally drive much slower in urban areas, therefore our evaluation setup represents a realistic LTE vehicular network approximation. The measurements shown in Table I are also in line with our setup's ES-to-cloud RTT. Although being an indoor evaluation, our setup is close to vehicle networks' reality and validated for testing performance in real scenarios.

A. ES placement

To address the ES placement problem, we consider the base station and traffic distribution patterns in central London. The selected area has a size of 3.91km * 5.75km. For this area, we use public LTE base station location data[2] and traffic volume data[3]. We cluster the traffic volume data according to its GPS coordinates and divide the selected area into 7 small areas according to the clustering result. The traffic distribution and area partition results are shown in Figure 5. Each colored dot represents the location of the aggregated traffic, with size proportional to the traffic volume in 12 hours during daytime. In each area, we display the number of deployed ESes, and average and the peak traffic volume. We evaluate both cases to have a better understanding of ES placement's influence on system performance (see subsection VI-C).

We then analyze the relationship between the base station distribution and the traffic density, as it influences our co-located ES placement. There are 3455 LTE base stations

[2]https://unwiredlabs.com

[3]https://data.gov.uk/dataset/gb-road-traffic-counts

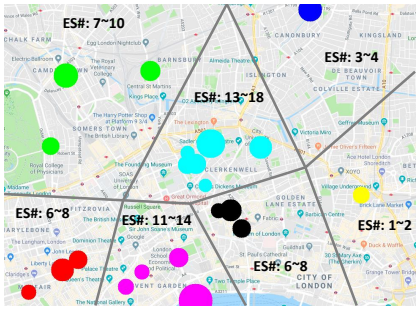

Figure 5: ES placement based on traffic heatmap

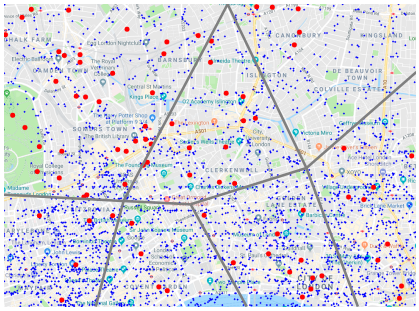

Figure 6: LTE base station (in red, the ones with coverage > 3000m) distribution in the selected area of London.

located within this area, among which 81 base stations cover more than 3000m, comparable to a macrocell. We plot these "macrocells" in red and the others in blue, as shown in Figure 6. The base stations are distributed evenly and reasonably match the amount of traffic in dense areas. As a result, using base stations as deployment points is not going to deviate the ES placement from the actual traffic patterns. The maximum number of ESes needed to meet the user demand during peak traffic is 64. Thus, we deploy ESes within the macrocells which are closer to the location of the aggregated traffic.

B. AR Applications

We then carried out experiments into the data processes in our representative AR application: View Share. We ran the application using ES and cloud for 1s (View Sharing) over 100 runs and measured the step by step latencies. Each run includes the entire integrated workflow of the application (see section IV). The result is shown in Figure 7.

The total latency for View Share can be divided in the following segments: *Client Data Collection*, *Uplink Latency*, *Object Detection*, *Policy Control*, *Downlink Latency* and *Client Rendering*. As Figure 7 shows, View Share with ES is 32.9ms faster than with the cloud. Due to the hardware difference, algorithms run slightly faster on ES than on cloud. Nevertheless, the sum of uplink and downlink latencies decreases by 21.7ms, which contributes to most of the improvement. This represents a 20% improvement over cloud computing, with an overall latency under 100ms. This proves our core assumption that edge computing can significantly improve latency-sensitive workloads by performing data processing closer to the user. It

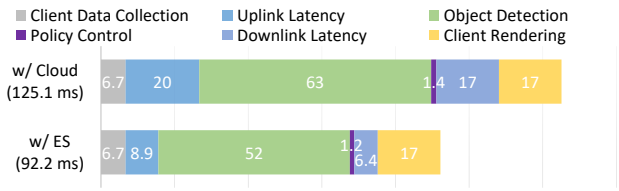

Figure 7: View Share Latency Decomposition.

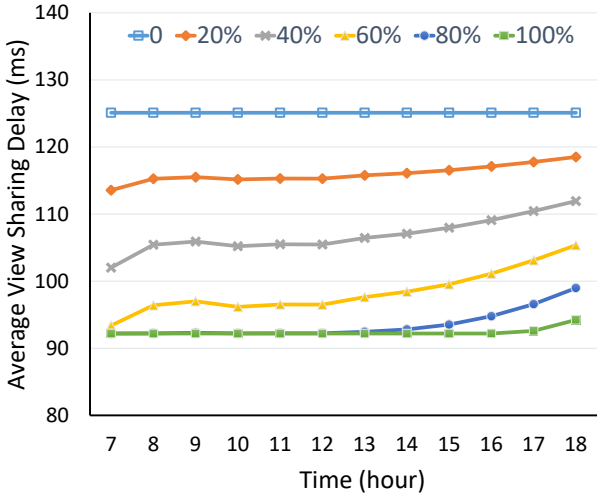

Figure 8: Average latency in different time periods for various load distribution between ES and cloud (100% is Edge only)

also addresses part of our second challenge (section I): edge computing improves AR application in vehicle networks, on the granularity of single workflows.

C. Scalability

To evaluate the scalability of our system, we test the bandwidth requirements and average latency for different vehicle traffic densities. We reuse the setup described in subsection VI-A. We select traffic data from 7am to 6pm (12 hours), during which the number of passing vehicles increases from 8262 to 14494 and has a peak at 6pm. Each ES has a coverage area of 3km, similar to its co-located macrocell. The average vehicle speed is 36km/h (10m/s). Each vehicle spends on average 5min within the coverage area of a given ES. The public dataset we used only contains vehicle count per hour. Here we build a vehicle distribution model based on Normal distribution. In our experiment, we use 5 minutes (300 s) as the cycle of this normal distribution, with a mean of 150 s and variance of 150 s, and the same pattern repeats 12 times for an hour. For every 5 minutes, the number of vehicles within the coverage of ES changes with time, as the first-appear time and the speed of these vehicles vary. Each vehicle sends captured images continuously to the nearest ES or cloud at 30 fps. Out of the 30 frames per second, the first frame is 960 x 540 pixels (51.8 KB) which is used for object detection.

In Table 3, we display the minimum and maximum bandwidth requirements of our demo application over the

	Device	Edge Server	Cloud Server
Min	0.0581 MB/s	2.89 MB/s	184.9 MB/s
Max	0.0581 MB/s	5.07 MB/s	324.5 MB/s

Table III: Minum and Maximum bandwidth requirements at device, edge and cloud level for EARVE during a day.

day.During peak hours, a cloud server running such operations would require more than 2.5 Gb/s for a single software. On the contrary, when using edge server, the load at the co-located base station is around 40 Mb/s with 0.465 Mb/s per device, which is achievable on LTE networks, and considerably relaxes the overall load at the bottleneck (up to 98%).

Next, we test the delay of the AR application, under different ES deployments and traffic densities along the 12 hours. As shown in Figure 8, we tested 6 ES deployments with different numbers of ESes. The different deployments are defined by the ratio of "summation of edge capacity" to "overall demand". The definition of "edge capacity" depends on specific service provisions. For instance, the period (T) of object detection is $1s$. Then a single ES has capacity calculated as follows,

$$C_{object} = T/delay = 1000(ms)/52(ms) = 19$$

which means an ES can process 19 tasks of object detection (on GPU) *within the corresponding period*, parallelized. Based on this definition, we defined following ES deployments. "0" is the pure cloud solution without any ES deployment, and we extend the cloud server resource to make it capable of fulfilling all the requests (provides the optimal performance for a cloud solution, to be compared with ES solutions). "100%" is the pure edge solution when the overall capacity of deployed ESes can fulfill the peak demand *on average*, e.g., there are 14494 passing vehicles during the hour around 6pm in the area, the average number of object detection tasks received by each ES during this hour is 1200 per second, which is calculated as follows:

$$N_{r/ES} = \Delta N_v * T * f = 14494/3600 * 300 * 1$$

where ΔN_v is the incremental number of discovered vehicles per second, T is the time a vehicle crosses the coverage of an ES, f is the request frequency. In this particular hour, it requires 64 ESes $(1200/19)$ to fulfill the average demand (as shown in Figure 5). With this setup, pure edge solution can fulfill *most* requests during the peak hour, except for those peak time points when significantly more than the average number of vehicles sending requests simultaneously. The other deployments are mixtures of edge and cloud solutions, where ESes can fulfill specific percentages of the requests and forward the rest to the cloud, e.g., "80%" represents when deployed ESes can fulfill 80% of the requests and the rest 20% are sent to the cloud. The result in Figure 8 shows that deployments with more ESes have lower delays. Comparing with the pure cloud solution, the pure edge solution decreases delay by 32ms for View Sharing in most periods, while the others also decrease delay at different levels. "80%" deployment gets similar delays with the pure edge solution.

In summary:

1) Our ES placement proposal follows the practical traffic and base station distribution.
2) EARVE improve the AR applications in vehicle networks by decreasing the transmission latency.
3) EARVE is scalable and performs well in different traffic densities. It can also be combined with cloud solutions to optimize the costs.

VII. Related Work

Emerging technologies enable various functions for autonomous vehicles but also bring new challenges.

Network protocols: Direct Short Range Communication (DSRC), Device to Device (D2D) and 5G, improve data transmission [6]–[8]. However, the large volumes of data will challenge current computation resource deployments and risk making them bottlenecks [13]. In this paper, we focus on V2E communication and select LTE as the network protocol. The rationale is straightforward, it accords with our scheme that ESes are co-located with base stations. Moreover, the major network workloads of our system and AR application, are image transmission and notification broadcast (or multicast depending on the rules). LTE outperforms DSRC in both workloads because of its longer coverage range and throughput performance, as shown in work [14].

Edge Computing: Edge computing to bring computation close to the user has attracted attention, such as [15] which explores integration of 5G, SDN, MEC and vehicular network. Uncoordinated strategies for edge service placement have been investigated in [16] and the results have shown that they work well for this problem. Meanwhile, the fundamental issues, i.e., architecture design, communication process, network protocols and implementation concerns are yet to be explored.

Applications: Efforts on developing vehicular applications have achieved some results [9], [17], but without improvement from system and networking point of view, those applications face difficulties to scale in realistic situations.

VIII. Conclusion

In this paper, we present EARVE, an architectural framework for vehicle-to-edge applications. Our system exploits the low latency of Edge servers to provide real-time emergency detection and notification. Thanks to its layered architecture, EARVE provides servers at street, neighborhood, and city that allow for a variety of usages at different scales in time and space. EARVE also presents the advantage to be mostly agnostic to the network and the hardware of vehicles and offloads most of the computations to ESes. To validate the concepts behind EARVE, we build a prototype application that we evaluate through both simulations and real-life conditions. Using real traffic data from London, we show that EARVE improves vehicular network significantly with reasonable requirements in terms of number of installed edge servers. Our evaluation results show that, compared to cloud solutions, EARVE decreases the latency of AR applications in vehicle newtorks, e.g., 26.3% for View Sharing. We also investigate the scalability of EARVE and show that it decreases latency in realistic scenarios for different traffic densities. We test mixed edge and cloud solutions and find out that more ES deployments bring larger improvements. In a word, EARVE is an efficient V2E solution which improves the performance by decreasing user latency and reducing network traffic.

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] M. Faezipour, M. Nourani, A. Saeed, and S. Addepalli, "Progress and challenges in intelligent vehicle area networks," *Communications of the ACM*, vol. 55, no. 2, pp. 90–100, 2012.

[4] M. Gerla, E.-K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 241–246.

[5] 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.

[6] A. Nordrum, "Autonomous driving experts weigh 5g cellular network against dedicated short range communications," *IEEE Spectrum, Cars That Think*, 2016.

[7] MOVIMENTO, "Connected car battle lines are drawn between 5g and dsrc," 2018. [Online]. Available: https://movimentogroup.com/media-coverage/connected-car-battle-lines-drawn-5g-dsrc/

[8] Alessio Filippi, Kees Moerman, Gerardo Daalderop, Paul D. Alexander, Franz Schober, and Werner Pfliegl, "Ready to roll: Why 802.11p beats lte and 5g for v2x," 2016.

[9] 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.

[10] T. L. Foundation, "Edgex foundry," 2017. [Online]. Available: https://www.edgexfoundry.org/

[11] 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

[12] Q. Xiao, K. Xu, D. Wang, L. Li, and Y. Zhong, "Tcp performance over mobile networks in high-speed mobility scenarios," in *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*. IEEE, 2014, pp. 281–286.

[13] T. Braud, F. H. Bijarbooneh, D. Chatzopoulos, and P. Hui, "Future networking challenges: The case of mobile augmented reality," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 1796–1807.

[14] Z. Xu, X. Li, X. Zhao, M. H. Zhang, and Z. Wang, "Dsrc versus 4g-lte for connected vehicle applications: a study on field experiments of vehicular communication performance," *Journal of Advanced Transportation*, vol. 2017, 2017.

[15] 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.

[16] 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.

[17] 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.