Future Generation Computer Systems 94 (2019) 237–249

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

CASQ: Adaptive and cloud-assisted query processing in vehicular sensor networks☆

Yongxuan Lai [a,b,*], Lu Zhang [a,b], Fan Yang [c], Lv Zheng [a,b], Tian Wang [d,*], Kuan-Ching Li [e]

[a] Shenzhen Research Institute, Xiamen University, Shenzhen 518000, China
[b] Software School, Xiamen University, Xiamen 361005, China
[c] Department of Automation, Xiamen University, Xiamen 361005, China
[d] College of Computer Science and Technology, Huaqiao University, Xiamen 361021, China
[e] Department of Computer Science and Information Engineering, Providence University, Taichung 43301, Taiwan

ARTICLE INFO

Article history:
Received 31 January 2018
Received in revised form 20 October 2018
Accepted 26 November 2018
Available online 1 December 2018

Keywords:
Cloud-assisted
Query result forwarding
Data storage
Query processing
VANETs

ABSTRACT

Vehicles in urban cities are equipped with increasing more sensing units. Large amount of data are continuously generated and they bring great potentials to the intelligent and green city traffic management. However, data gathering and query processing remain key and challenging issues due to the huge amount of sensing data, changeable road conditions, rapid network topology and density changes caused by the movement of vehicles. There is great necessity for the cloud and the vehicular sensor networks to integrate and enhance each other on the cooperative urban sensing applications. In this paper we propose an adaptive and cloud-assisted query processing scheme for VANETs, that adopts the concept of edge nodes and integrates the cloud and vehicular networks to facilitate data storage and indexing, so queries could be processed and forwarded along different communication channels according to the cost and time bounds of the queries. Moreover, the cloud calculates result forwarding strategy by solving a Linear Programming problem, where the query results select the best path either through the 4G channel or through the DSRC (Dedicated Short Range Communication). This research is one of the first steps towards the integration of the cloud and the vehicular networks, as well as edge nodes and the 4G channel, to improve the effectiveness and efficiency of the query processing in VANETs. Extensive experiments demonstrate that up to 94% of the queries could be successfully processed in the proposed scheme, much higher than existing query schemes, while at the same time with a relatively low querying cost.

1. Introduction

Vehicular nodes are equipped with more and more sensing units, and large amount of sensing data such as GPS locations, speed, video clips, and emission values of greenhouse gases are generated [1,2]. These data are shared or uploaded as input for applications that aim at more intelligent transportation, more emergent responses, and more reduction on pollution and fuel consumption. This brings about the concept of *cooperative vehicular sensing*, which is at the heart of intelligent transportation and green traffic management system [3,4]. Key components of the system are a combination of pervasive vehicular sensing system and a central control and analyzing system [5,6], and recently vehicular ad-hoc sensing networks (VANETs) have emerged [7–9]. Vehicles traveling on the road exchange information with nearby vehicles through the V2V (vehicle to vehicle) and V2I (vehicle to infrastructure) communications, and data can be disseminated to reach far distance by using moving cars as intermediates, following multi-hop routing protocols.

Nevertheless, vehicular data gathering or query processing remains a key and challenging issue in VANETs. On one hand, vehicular nodes are limited to road topology while moving, where the network usually suffers rapid topology and density changes under various road conditions and high moving speeds, so the communications are usually fragmented and intermittent-connected. On the other hand, vehicular sensed data are in large volume and characterized as continuous generation. The sensed data should be filtered and pre-processed before being shared or uploaded, data filtering and query processing technologies that tailored to the VANET environment are highly required [4].

Generally speaking, there are two key strategies for data gathering in VANETs: the *push* and the *pull*, which are similar to those

☆ This research is supported by the Natural Science Foundation of China (61672441, 61872154), the Shenzhen Basic Research Program, China (JCYJ20170818141325209), the National Key Technology Support Program, China (2015BAH16FF01), and the State Scholarship Fund of China Scholarship Council (201706315020).
* Corresponding authors.
E-mail addresses: laiyx@xmu.edu.cn (Y. Lai), wangtian@hqu.edu.cn (T. Wang).

https://doi.org/10.1016/j.future.2018.11.034
0167-739X/

considered in the field of distributed and mobile databases [10–12]. In a push-based model, each vehicle senses the data and proactively uploads them to a central server through V2V or V2I communications [13,14]. As a node receives data from its encountered nodes, it has to decide whether the data are relevant or not. The node might incur unneeded overheads such as duplicate messages or irrelevant data. In a pull-based model, queries are submitted from ordinary nodes or the cloud [15,16]. Vehicles should be able to interpret, route, and process those queries, and route back the query results to query requesters.

Pull-based model provides more flexibility in terms of types of queries [17]. As queries could in principle be diffused far away to retrieve remote data in the pull-based model, there are three steps in the query processing: (1) query requester diffuses the query to different data sources, either directly or by using multi-hop relaying techniques; (2) each node that receives the query computes a partial query result based on its local data; and (3) nodes route the query result to the query requester. However, most of existing pull-based query schemes assume no fixed data server available in VANETs, since they only consider the resource of the in-network vehicular nodes [16–18]. Inevitably, those approaches incur relatively large query delays. Moreover, routing the query results back to the query originator is a challenging problem in VANETs because the query requester is moving in the meanwhile. The performance degrades when routing results to the query requester merely based on simply deducing the geographic locations in the field.

To overcome these drawbacks of existing query processing schemes in VANETs, we propose an efficient scheme namely CASQ (*Cloud-Assisted data Storage and Query Processing*) in this paper, to take advantage of the fog/edge nodes, and integrate the cloud and vehicular networks to facilitate data storage and indexing, so queries could be processed and forwarded along to different communication channels, including the 4G and DSRC (Dedicated Short Range Communication [19]), according to the cost and time bounds of the queries. Queries are firstly uploaded to the cloud and are directed to edge nodes (also called RSU, road side units) to extract query results. The cloud then computes a strategy of result forwarding by solving a QRF problem (*query result forwarding problem*) that defines how to deliver the query results through their best paths, either through the 4G channel or through the DSRC communications. Query results are diffused to a set of RSUs that the query requester would visit along its travel path, and the requester could fetch results from these nodes before the query is outdated.

The main challenges lie in two aspects: (1) the data should be stored and indexed in way that queries could be directed to the RSUs to access the query results efficiently and quickly, and (2) query results should timely be forwarded back to the query requester, who is moving and expects to fetch the query results on its way to the destination. The ordinary nodes, edge nodes (RSUs) and the cloud should cooperate with each other to process the query, and the forwarding mechanism of queries and results should be well designed. The main contributions of this paper are as follows:

1. We propose a query processing scheme called CASQ that integrates the cloud/edge computing and vehicular networks. The proposed scheme takes advantage of resources at the cloud and edge nodes to cooperatively store and index data, so that queries could be processed efficiently and timely. Data readings are stored at local nodes or RSUs, and indexed by the cloud; queries are processed at the cloud and directed to RSUs to extract the query results.
2. We map the query result forwarding problem into a linear programming problem that could be solved efficiently, where the query results choose their best paths either through the 4G or the DSRC. Query results are then able to be diffused to RSUs along the traveling path of the query requester, so the query requester could fetch the results just before the query is outdated. In addition, we adopt a priority-based strategy for the management of data segments and update the indexes accordingly at the cloud. The proposed scheme manages the limited storage of RSUs and keeps a up to date list of index entries simultaneously.
3. We conduct experiments based on real trajectory data and simulations to demonstrate the effectiveness of the proposed algorithm in vehicular sensing applications. Up to 94% of the queries could be successfully processed in the proposed scheme, much higher than existing query schemes, while relatively low query cost at the same time.

To the best of our knowledge, this paper is the first report on the integration of the cloud, the edge nodes in vehicular networks, and the 4G communications channels to improve the effectiveness and speeding up of the query processing in VANETs. The rest of the paper is structured as follows. Section 2 describes the related work, Section 3 introduces some preliminaries and defines the network model, and Section 4 presents the detailed description of the CASQ algorithm, including data storage, indexing, and the query processing. Section 5 describes the environmental setup and analyzes the simulation results, and finally, Section 6 concludes the paper.

2. Related work

Vehicles could be viewed as powerful mobile sensors. Recent researches have addressed the problem of data and information gathering in vehicular networks. Zhao et al. [3] proposed several vehicle-assisted data delivery (VADD) protocols to forward the packets to the best road with the lowest data-delivery delay, among which the hybrid probe (H-VADD) protocol has a much better performance. Lee et al. [13] proposed the MobEyes system for proactive urban monitoring. The system exploits the vehicle mobility to opportunistically diffuse concise summaries of the sensed data, so it harvests these summaries and builds a low-cost distributed index of the stored data to support various applications. Palazzi et al. [14] proposed a delay-bounded vehicular data gathering approach that exploits the time interval to harvest data from the region of interest satisfying specified time constraints, and properly alternates the data muling and multi-hop forwarding strategies. Nevertheless, the solution has to be integrated with a geocast protocol for the query propagation, which incurs large time delay. Lai et al. [4] proposed a fog-based two-level threshold strategy to gather data of events in VANETs. In the monitoring phase, nodes sense the environment in low cost sensing mode and generate sensed data. Whenever the probability of event exceeds some threshold, nodes transfer to the event-checking phase. Some nodes are selected to transfer to the deep sensing mode to generate more accurate data of the environment.

Several other works have addressed query processing in vehicular networks. PeopleNet [15] is an infrastructure proposal for information exchange in a mobile environment. It relies on the existence of a fixed network infrastructure to send a query to an area that may contain relevant information. Lee et al. [16] proposed the FleaNet scheme, a mobility assisted query dissemination where the node that submitted a query periodically advertises it only to its one-hop neighbors and looking forward if they can provide some answers from information stored on their local storage. Similar to FleaNet, Roadcast [18] is a content sharing scheme for VANETs, and it queries other vehicles that it encounters on the way. The keyword-based queries are submitted by the users and the scheme tries to return the most popular content relevant to the query. Differed to the proposed CASQ scheme, FleaNet and Roadcast only

query from the one-hop neighbors, the ratio of successful queries is relatively low, and they do not need to consider the problem of routing back the query results.

Xu et al. [20] considered the problem of searching documents in a vehicular network. They adapted the concept of Distributed Hash Table (DHT) [21] to a mobile environment and proposed a Hybrid Retrieval (HR) approach that selects between a flooding or a DHT scheme for indexing and searching based on the expected costs. Similar to [20], Delot et al. [17] proposed the GeoVanet scheme, which uses a DHT-based model that identifies a fixed geographical location where a mailbox is dedicated to the query to allow the user to retrieve his/her results in a bounded time. Paczek et al. [22] introduced a method of selective data collection for traffic control applications. The underlying idea is to detect the necessity of data transfers on the basis of uncertainty determination of the traffic control decisions, and sensor data are transmitted from vehicles to the control node only at selected time moments. The above-mentioned schemes have the drawback, since they only consider the resource of in-network vehicular nodes, which are dynamic in nature and usually incur relatively large query delays. Queries are first diffused to nodes that have the query answer, then the data readings that contain the query results are routed to a fixed geographical location. Finally, the query requester issues another request to fetch the data from that location, so it takes longer time delay for a requester to get the query results.

Recently, there is also a research trend to integrate the cloud and vehicular networks. Eltoweissy et al. [23] for the first time coined the term of Autonomous Vehicular Clouds (AVC), where a group of large number of autonomous vehicles whose corporate computing, sensing, communication, and physical resources can be coordinated and dynamically allocated to authorized users. The concept of VANET Cloud, however, is highly related to "Fog/Edge computing" [8,24,25], which extends traditional cloud computing paradigm to the edge of networks. Bonomi et al. [26] defined the characteristics of fog computing and its role in the framework of Internet of Things. They emphasized the fact that the fog/edge brings new elements to the realm of Internet of Things through reduction of service latency and improvement of QoS (Quality of Service). Recently, Kai et al. [27] delivered a survey on some opportunities and challenges related to the context of fog computing in VANETs.

Our early results [9] have demonstrated that the cloud and 4G channel could be integrated with VANETs, that takes full advantage of resource at the cloud and the edge nodes/RSUs for query processing at VANETs. Data are cooperatively stored and indexed, and queries are processed and forwarded along different paths according to the cost and time constraints of queries. Despite the integration and mutual interaction among the cloud and the edge nodes (RSUs), queries in VANETs could be processed effectively and efficiently.

3. Preliminaries

Fig. 1 presents a three-layered vehicular network that consists of network layer, fog layer, and the cloud layer. We assume that each vehicle v_i monitors the surrounding environment through periodical sensing. These data readings are then sent to fog nodes (RSUs) through DSRC communications [28], which are one hop V2I or multi-hop V2V transmissions. Fog nodes own computing, storage and communication capabilities, as they are located at the edge of the network and cooperate with the cloud, adopting a "fog-cloud" collaborative computing and storage strategy to provide a unified, efficient, and low-latency services for various applications. The three-layered vehicular cloud system makes position-relevant and real-time applications possible.

Queries are submitted by vehicular nodes to retrieve query results. A query is denoted by $query(s, f, t, a)$, where s is the source

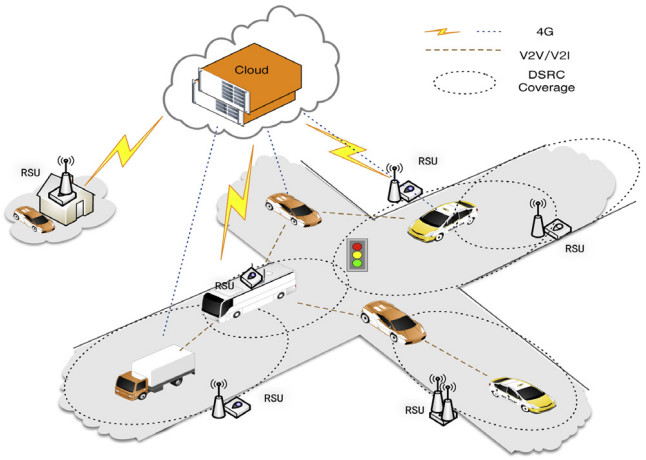
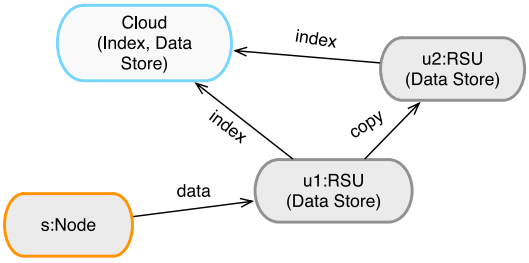
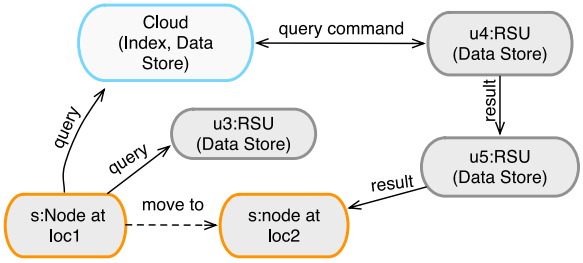


Fig. 1. Illustration of a three-layered Vehicular Ad-hoc Network.



(a) Steps in data storage phase



(b) Steps in query processing phase

Fig. 2. Steps of data storage and query processing.

node that generates the query, f is a filter of the query, t is a time bound that query results should be returned, and a is the information attached by the source node to facilitate the query processing. When a vehicular node submits a query, it keeps moving along its path to its destination. At this point, we assume the target and traveling path are known, e.g. through the GPS navigation system, and hence could be extracted as the input for the scheme. The cloud and RSUs work cooperatively to process the query and cache the results on intermediate RSUs, from which the query requester would fetch the query results along its way to destination before the query is expired.

4. CASQ Framework

4.1. Overview

CASQ takes advantage of resources at the cloud and edge nodes to cooperatively store and index data, so that queries could be processed efficiently and timely. Fig. 2 shows the overall steps of data

storage and query processing. At the storage phase, ordinary nodes sense the data and upload part of the data to the RSU through DSRC communications [28]. When an RSU, e.g. $u1$ receives the data, it extracts the sketch of data and sends an index update to the cloud. Within the query processing, data might also be forwarded and copied to other RSUs, e.g. $u2$, for storage, yet its index also updated accordingly at the cloud. At the query processing phase, a query is submitted by a node, e.g. s, at location $loc1$, and then forwarded to the neighboring nodes and RSUs. If the neighboring nodes or RSUs, e.g. $u3$, return no query results, the query is then forwarded to the cloud through the 4G channel. The cloud processes the query by searching its indexing database, and selectively sends a query command to a set of RSUs, e.g. $u4$, where the desired data is stored. The query is processed at $u4$ and query results are extracted. The query results are then forwarded to a set of intermediate RSUs, e.g. $u5$, where s moves to another location $loc2$ and is able to fetch the query results.

The idea of the proposed scheme is straightforward, yet the main challenges lie in two aspects: (1) the data should be stored and indexed so that queries could be directed to the RSUs and query results be accessed quickly, and (2) query results should timely be routed back to the query requester, who is moving and expects to fetch the query results on its way to the destination. The ordinary nodes, RSUs and the cloud should cooperate with each other to process the query, where the forwarding mechanism of queries and results should be well designed.

Detailed descriptions of the scheme are presented in the following subsections.

4.2. Data storage and indexing

Large amount of data are generated in the network from various data sources in VANETs. Data are either sensed by the equipped sensing devices in vehicles, or forwarded by electronic devices owned by drivers or passengers, e.g. smart phones, through the intra-vehicle communication channels.

As a piece of data is generated, it is uploaded to nearby RSUs through the V2I communications, and the RSU that receives the data would store the data and maintain an index entry of the data in the cloud through RSU-to-cloud communications. Each piece of data is denoted by $data(id, sk, segs)$, where id is the identification, sk is a sketch describing the data, $segs$ is a list of data segments, in which might consist of larger size of data. Each segment is denoted by $seg(sid, ssk, scon)$, which is in similar form as $data$. Here sid is id of the segment, ssk is the sketch of the segment, $scon$ is the detailed data of the segment. Sketches could be viewed as a type of metadata that are generated and evolved as data are stored and processed. The properties of metadata include data id, size, tag, locations, length, etc. Example of sketch of data that described the attributes and structure of data and data segments follows next:

```
// ssk(d1): an example of sketch of data d1,
// which is a clip of sensed video.
{
    id: 4578,
    sk:  {
        source: "node124",
        time: "16 Dec 2016 09:24:27 GMT"
        size: "512 M",
        type: "mp4",
        tags: "accident, rain",
        locations: "u24"
    }
    segs:[{sid:1,
     ssk:{
        data_id: "4578",
          size: "64 M",
         tags: "accident, rain",
         length: "5 minutes",
        locations: "u24" },
      scon: "/var/4578/001.data"
  } ]
}
```

Note that the *locations* attribute indicates where the data are stored and could be retrieved. Sketches could be stored in self-described formats, e.g. JSON or XML, which are fully supported by the mainstream search engines deployed at the cloud. As the cloud is abundant of storage and computing resources, it is feasible to deploy search engines to store and index the sensing data efficiently. For example, the Elasticsearch engine[1] is a distributed, RESTful search and analytics engine capable that provides easy APIs to store and query the sketches. When RSUs receive segments of new data, they would send sketches of these segments to the cloud, so the cloud could track all the data and their copies within the network.

4.3. Query processing

Different methods are adopted to process a query, depending on different surrounding environments of vehicular nodes. If a node currently has no neighbors nor in contact with an RSU, the query is sent to the cloud directly through the 4G channel. On the other hand, if a node is within the coverage of an RSU or neighboring nodes, the query is first sent to the RSU or neighbors for processing through the V2V or V2I communications. If the query could be fully answered, the result is immediately returned and the processing is finished. Yet there are cases when the query is not fully answered:

(1) The RSU has only part of the query results, the query should be forwarded to the cloud for further processing. Both the *result searching* and *result forwarding* steps are needed to search the query results and deliver them to the query requester.
(2) The RSU has the full data of query result, yet the node that submits the request has moved out of its coverage or there is not enough time receiving the result through the V2I or V2V communications. A *result forwarding* step is needed to transfer and deliver the query results appropriately.

4.3.1. Search and forward query results

When a query is received by an RSU or the cloud, the receiver would search its data storage to retrieve the query results. If RSUs receive the query, they would search their local data storage to extract the query results. In the case that the cloud receives the query, the search engine deployed would perform the query searching on the storage of sketches. The search engine returns data sketches that indicate the locations of the desired data storage, and the cloud then sends query command messages to the dedicated RSUs to extract the query results.

One main constraint of the query processing is the time constraint, which is denoted by t. A query is successfully processed only when the query result is returned to the requester within t, otherwise the query is failed". Therefore, query results are acquired at an RSU, e.g. u_s, as the results should be forwarded back to the query requester (also called source node) on time. Here we first introduce some notations:

- Symbol s denotes the query requester (source node);

[1] https://www.elastic.co/products/elasticsearch.

- Symbol $\mathbb{M} = \{m_1, m_2, \ldots, m_k\}$ denotes a set of k RSUs where the query results are retrieved;
- Symbol $\mathbb{D} = \{d_1, d_2, \ldots, d_l\}$ denotes a sequence of l RSUs that source node s would pass through to fetch the query results;
- Symbol $x_{i,j}$ denotes the amount of data forwarded from m_i to d_j;
- Symbol $x_{*,j} (= \sum_{i=1}^{k} x_{i,j})$ denotes the total amount of data received at d_j.

We assume there are three basic communication channels in VANETs: DSRC (V2V, V2I), Wired (I2I) and 4G. The time needed differs in different channels and forwarding strategies. In CASQ, query results could be forwarded along two paths, which are described as follows:

(1) *Path*1: Query results are uploaded to the cloud from RSUs in $\mathbb{M}$, and the cloud forwards them to requester s directly through 4G. Being the cloud denoted by d_0, the amount of data forwarded from m_i to the cloud by $x_{i,0}$, the time needed for the transmission is:

$$\varphi(x_{*,0}) = \eta_{4g} + \gamma_{x_{*,0}} + max(\frac{x_{*,0}}{BW_{u,4g}}, \frac{x_{*,0}}{BW_{d,4g}}) \tag{1}$$

where η_{4g} denotes the cost of time to establish a connection, $\gamma_{x_{*,0}}$ denotes the time needed for data processing at the cloud and RSUs, BW denotes the bandwidth of the channel, and the subscript u and d denotes the up and down link of the channel. The *max* function implies the uploading of query results to the cloud is in parallel with the downloading to the source node s.

(2) *Path*2: Query results are forwarded to RSUs in $\mathbb{D}$ through I2I or V2V communications, and data are fetched when s moves through these RSUs. The time needed could be estimated by the cost of traveling time for s to visit RSUs in $\mathbb{D}$ one by one:

$$\rho = \frac{dis(u_s, d_1)}{v_{u_s,d_1}} + \sum_{j=1}^{l-1} \frac{dis(d_j, d_{j+1}) * \psi(j)}{v_{d_j,d_{j+1}}} \tag{2}$$

$$\psi(j) = \begin{cases} 1, & if \quad \sum_{z=j+1}^{l} x_{*,z} > 0 \\ 0, & if \quad \sum_{z=j+1}^{l} x_{*,z} = 0 \end{cases} \tag{3}$$

where $dis(a, b)$ is the distance between a and b, $\psi(j)$ is a function denoting whether d_j is the last RSU participating the forwarding, and $v_{a,b}$ is the average speed of node s moving from a to b. As mentioned in Section 3, we assume the speed of vehicles could be estimated through periodic data exchange between vehicles and RSUs.

Data are transferred through 4G channel in *Path*1. It is faster yet suffers higher economic cost. In contrast, data forwarding through *Path*2 is cheaper, since it takes advantage of the DSRC channels, despite incurs larger delay due to the "store-carry-forward" strategy and multi-hop routing protocols.

4.3.2. Solving the query result forwarding problem

At extreme cases, all query results are forwarded along one path, either *Path*1 or *Path*2. At more general cases, the data could be forwarded through both paths to strike a balance between the cost and time delay.

As the query is bounded by a time interval to return query results, we define the query result forwarding (QRF) as a linear programming problem:

$$\text{Minimize: } (x_{*,0} * (c_{u,4g} + c_{d,4g}) \tag{4}$$

$$+ \sum_{i=1}^{k} \sum_{j=l}^{l} x_{i,j} * c_{i,j} \tag{5}$$

$$+ \sum_{j=1}^{l} \sum_{i=1}^{k} x_{i,j} * c_{j,s}) \tag{6}$$

Subject to:

$$x_{i,j} \geq 0, \quad i = 0, 1, \ldots, k, \quad j = 0, 1, \ldots, l \tag{7}$$

$$(x_{*,0} + \sum_{i=1}^{k} \sum_{j=1}^{l} x_{i,j}) = |data| \tag{8}$$

$$\frac{x_{*,j}}{BW_{i2i}} \leq time(s, d_j), \quad j = 1, 2.., l \tag{9}$$

$$\frac{x_{*,j}}{BW_{i2v}} \leq \frac{2R}{v_{d_j}}, \quad j = 1, 2.., l \tag{10}$$

$$\varphi(x_{*,0}) \leq t - \Delta \tag{11}$$

$$\rho \leq t - \Delta \tag{12}$$

where $x_{i,j}$ is the amount of data forwarded from m_i to d_j, $x_{*,0} (= \sum_{i=1}^{k} x_{i,0})$ is the total amount of data received at d_0, i.e. the cloud. Symbol $c_{u,4g}, c_{d,4g}$ is the unit cost of the up and down channel of 4*G*, which are predefined unit cost parameters. Symbol $c_{i,j}$ is the unit cost of transmissions from m_i to d_j, symbol $c_{j,s}$ is the unit cost of transmissions from d_j to source node s. So, term (4) is the cost forwarding results through *Path*1, term (5) is the cost forwarding results from RSUs in $\mathbb{M}$ to intermediate RSUs in $\mathbb{D}$ through *Path*2, and term (6) is the cost of forwarding the results from RSUs in $\mathbb{D}$ to the query requester s. At this point, k and l are the sizes of $\mathbb{M}$ and $\mathbb{D}$ respectively, given that k is calculated at the cloud and l is estimated by the path that s would visit when moving to its destination. The objective is to minimize the overall cost of the query result forwarding. Considering the 4*G* channel expensive and might be overloaded by the large amount of vehicular nodes, $c_{u,4g}$ is set much larger than $c_{i,j}$, which leads to forwarding some data through *Path*2 in the model.

Constraint (8) ensures all the data should be forwarded to the query source either through the cloud or through DSRC communications, where $|data|$ is the amount of query results. At constraint (9) and (10), R is the radius of the coverage area of RSU, BW_{i2i} is the bandwidth of the RSU-to-RSU (I2I) communication channel, BW_{i2v} is the bandwidth of the RSU-to-Vehicle (I2V) channel, $time(s, d_j)$ the cost of time from the location of node s to d_j, v_{d_j} is the average speed within the coverage area of d_j. Constraint (9) implies the data be prepared and received at d_j before s arrives the coverage area of d_j to fetch the data; constraint (10) implies there should be enough time for the node to fetch the data when it passes through the coverage area of d_j. Symbol $\varphi(x_{*,0})$ and ρ are defined at Eqs. (1) and (2), t is the time bound for the query, Δ is the time that has elapsed before forwarding the query results. Constraint (11) and (12) ensure the forwarding time should be within the time bound of the query. In this way, query results are forwarded in parallel through different communication channels.

As described in Algorithm 1, the output of the solver is variables $\{x_{i,j} | i = 1.., k, \ j = 0, .., l\}$. These parameters are routed to the RSUs at $\mathbb{M}$ as commands. A command, e.g. C, contains the destination (C.dest) and channel of the forwarding (C. channel), and the amount of data to be forwarded $x_{i,j}$. When an RSU, e.g. m_i, receives a command, it acts accordingly to send $x_{i,j}$ of the query result to d_j. Note that the query result forwarding problem could also be solved within the network if RSUs are with computing capability and have the solver installed. By borrowing the concept of "edge computing" [8,24], RSUs are viewed as an edge node that could find a solution to the query result forwarding problem. Yet at this research we assume the cloud is assigned to solve the query forwarding problem due to its abundant computing resource.

4.3.3. Setting of input parameters

BW_{i2v}, BW_{i2i} are input parameters of the QRF model that indicate the bandwidth of the communication channels. The bandwidth could be set according to DSRC communications. The time cost $time(s, d_j)$ and the speed v_{d_j} are other inputs for the QRF model. It is assumed the current location of query requester and its traveling path are known, e.g., through the GPS-based navigation assistance system, RSUs would learn and monitor the speeds and time costs of road segments.

A road segment, denoted by $S(I_1, I_2)$, is determined by two intersections I_1, I_2. Given two sensing readings that are generated by a vehicle when traveling along a road segment $S(I_1, I_2)$ at time slot ts, i.e. $d_1 = (s, t_1, I_1, dr)$, $d_2 = (s, t_2, I_2, dr)$, $t_1, t_2 \in ts$, the cost of S measured by s is:

$$mv(s, S, ts) = t_2 - t_1 \tag{13}$$

The calculation cost is done locally at vehicular nodes, and at intersection I_2 the measured value $mv(s, S, ts)$ is uploaded to the RSU.

Given a set of measured mv values, the time cost and average speed along road segment S at time slot ts could be estimated as follows:

$$cost(S, ts) = \frac{1}{|\Omega(S, ts)|} \sum_{mv \in \Omega(S,ts)} mv, \quad speed(S, ts) = \frac{len(S)}{cost(S, ts)} \tag{14}$$

where $\Omega(S, ts)$ is the set of received mv values that estimate the cost of S within time slot ts, $|X|$ is the cardinality of set X, and $len(S)$ returns the length of road segment S.

The time cost and speed of road segments are also uploaded to the cloud so the cloud has the data to calculate the input parameters for the QRF problem. The path from the current location of vehicle s to RSU d_j could be represented by connecting intersections of road segments along the path: $loc(s) \rightarrow x_2 \rightarrow \ldots \rightarrow x_{k-1} \rightarrow loc(d_j)$, where loc returns the location of a node.

We set $x_1 = loc(s)$, $x_k = loc(d_j)$, then the cost of time from current location of s to d_j could be estimated by:

$$time(s, d_j) = p_k - p_1 \tag{15}$$

where p_i is the time when x_i is visited by s. p_i is defined in a recursive way as follows:

$$p_1 = t, \quad p_i = p_{i-1} + cost((x_{i-1}, x_i), p_{i-1}), \quad i \in [2, k] \tag{16}$$

where t denotes the current time, (x_{i-1}, x_i) is a road segment on the path.

4.4. Update of storage and index entries

Data are generated from vehicular nodes and routed to RSUs, then stored at RSUs every period of time. Therefore, at some time point, the RSUs should free some space for new data, and the index entries at the cloud should be updated next.

As a data segment could not answer any query for a given period of time, it is advised to be transformed or removed from the network to free some storage space for the newly generated or received data. In CASQ, we adopt a priority-based strategy for the management of data segments and update the indexes accordingly at the cloud.

Data segments are stored at RSUs and organized into a list. The list is sorted according to the *important weight*, which is defined as a tradeoff between the 'freshness' and 'hotness' of the content. Formally speaking, the importance weight of seg is defined as:

$$seg.imp = \frac{T - (now - seg.ls\_time)}{T} * \alpha + \frac{seg.nr}{max(1, max_r)} * (1 - \alpha) \tag{17}$$

Algorithm 1: Messages handling at ordinary nodes

```
1  if generate data at s then
2      send(data, s.RSU, DSRC);
3  if generate query at s then
4      if s.NB = φ and s.RSU = φ then
5          send(query, cloud, 4G);
6      else send(query, s.RSU ⋃ s.NB, DSRC);
7      ;
8  if receive query then
9      result=search(query, localStorage);
10     if full_answer(result, query)==true then
11         result_forward(result, query.source);
12     else send(query, cloud); ;
13 if in contact with d_j ∈ 𝔻 then
14     result = fetch(d_j, query);
15     append_answer(result, query);
```

where $\alpha \in [0, 1]$ is the balance factor, T is a user defined time duration, now is the current time, $seg.ls\_time$ is the latest time when seg is requested, $seg.nr$ is the number of times that seg is queried, max_r is the maximal nr for all data segments in the storage list. Here the divisor T and $max(1, max_r)$ are used to normalize the weight of freshness and hotness, so they are of the same magnitude in range [0, 1]. All these parameters are recorded and updated locally at edge nodes. When a RSU receives a data read from vehicular nodes, it has to make new space to store that data. If there is enough free space, the segment is inserted into the ordered list according to its importance; otherwise, the segments with lower importance would be transformed or removed to free some storage space. More details are referenced and found at [29].

As a data segment is removed at a RSU, the *id*'s of the RSU, the data reading, and the segment are wrapped into a *remove* message and routed to the cloud. The search engine at the cloud would remove the segment accordingly upon receiving the message. Additionally, once all index entries of data segments that belong to a piece of data are removed, the index entry of the data is also removed. In this way, the data stored at RSUs and the index entries at the cloud are synchronized.

4.5. Algorithm description

Algorithms 1–3 present the messages handling pseudocode of CASQ scheme in an ordinary node, the RSUs, and the cloud respectively.

In Algorithm 1, when a piece of data is generated, it is uploaded to the RSU through the DSRC communications (line 1–2). Though, when a query is generated, if the node currently has no neighbors or is not in contact with an RSU, the query is sent to the cloud directly through the 4G channel (line 3–5), and if a node is within the coverage of an RSU or in contact with neighboring nodes, the query is first sent to the RSU or neighbors for query processing through V2V or V2I communications (line 6). If the query could be fully answered, the result is immediately returned and the query processing is finished (10–11). Though, when the query is not fully answered, the query is forwarded to the cloud for further processing (line 12). Finally, a node is in contact with an RSU in set $\mathbb{D}$, it fetches the query results from these edge nodes (line 14–15).

In Algorithm 2, when RSU receives data from ordinary nodes, it checks whether need to free some space (line 2). If there is not enough space, it would remove some data segments according to their importance weight and send an *update* message to the cloud to synchronize the index entries (line 4–5). Next, it stores the data

Algorithm 2: Messages handling at edge nodes.

```
1  if receive data from node then
2      d = check_remove(data);
3      if d! = null then
4          remove(d, localStorage);
5          send(remove(d), cloud);
6      store(data, localStorage);
7      send(data.sk, cloud);
8  if receive query from node then
9      result=search(query, localStorage);
10     if full_answer(result, query)==true then
11         result_forward(result, query.source);
12     else send(query, cloud); ;
13 if RSU m_i receives command from cloud then
14     for (x_{i,j}, d_j) in command do
15         segs = get_data(x_{i,j}, localStorage);
16         send(segs, d_j);
```

Algorithm 3: Messages handling at the cloud.

```
1  if receive index from RSU then
2      update(index, indexStorage);
3  if receive remove(d) from RSU then
4      remove(d, indexStorage);
5  if receive query from node then
6      problem = search(query, indexStorage);
7      commendSet CS= LPSolver.solve(problem);
8      for command C in CS do
9          send(C, C.dest, C.channel);
```

(line 6), and upload an index entry of the newly inserted data to the cloud (line 7). As RSU receives a query, it searches the query result at its local storage (line 9). If the query could be fully answered, the result is immediately returned and the query processing is finished (line 8–11). On the other hand, when the query is not fully answered, e.g., the RSU has only part of the result, the query should be forwarded to the cloud for further processing (line 12). When a RSU, e.g. m_i, receives a command, it acts accordingly to send $x_{i,j}$ of the query result to d_j (line 13–16).

In Algorithm 3, as the cloud receives an index entry of data, the entry is stored and the index is updated (line 2). Though, as the cloud receives a remove message, the entry that corresponding to the data segment d is removed from the index storage (line 4). When the cloud receives a query, it searches the query on the index storage and defines the *problem* in line 6, which indicates where to get the query results. The *problem* is then defined as a linear programming problem and solved by the solver (line 7), and the solved parameters are routed to the RSUs that have the query results as commands (line 8–9). A command, e.g. C, contains the destination ($C.dest$) and channel of the forwarding ($C.\ channel$), and the amount of data to be forwarded, i.e. $x_{i,j}$.

4.6. Complexity analysis

From Section 4.5 we could see that the message handling algorithms at the ordinary nodes, RSUs, or the cloud are linear in nature. And the amount of messages linearly relates to the amount of sensing data or generated queries.

The complexity of the CASQ scheme lies in computing the paths of query result forwarding. As discussed previously, the query result forward problem is transformed into a linear programming problem, which is well studied and could be solved efficiently in the worst case [30]. For a linear programming instance of n variables and m constraints, each iteration takes polynomial time $O(mn)$. The total number of iterations on problems arising in practice is usually fast, e.g. the simplex method [31]. Conventional wisdom suggests that number of iterations in practice is about $3m$, so the complexity for a linear programming instance is $O(3m^2n)$.

For the QRF problem presented in this work there are $(l+1)*k$ variables and $(kl+2l+3)$ constraints, the total complexity is:

$$O(3(kl+2l+3)^2*(l+1)k) \cong O(3k^3l^3) \tag{18}$$

where k and l are the sizes of $\mathbb{M}$ and $\mathbb{D}$ respectively. As there are p queries running on the cloud, the complexity is $O(3pk^3l^3)$.

Table 1
An instance of GPS record at Xiamen Taxi Dataset.

id	lon	lat	time
8250864460	118.024232	24.475228	2014-07-02 15:33:03
Speed	Direction	Occupied	
61 (km/h)	270 (°)	"vacant"/"occupied"	

5. Experimental results

To verify the performance of the proposed scheme, experiments are conducted on the Opportunistic Network Environment (ONE) simulator[2] [32] with real-world road network and trajectory datasets.[3] We import maps of Xiamen City from the OpenStreetMap and follows the PathMapBasedMovement to simulate the movement of nodes. Data reading and queries are injected into the simulation field as events, where ideal transmission channels are assumed. In this section, we present the data pre-processing, the environmental setup and the detailed experimental analysis.

5.1. Data pre-processing

The Xiamen Taxi Dataset is used for the simulation, that consists of one-month trajectory data of about 5,000 taxicabs in Xiamen city, China during July 2014, where there are about 220 million GPS position records and 8 million live trips. The trajectory reporting frequency is 1–2 times per minute, yet for this simulation we extracted 506 taxis — about 1/10 of the trajectories for performance evaluation. Table 1 lists the format of GPS data as example, with *id* is the identification of the trajectory or the transaction, *lon*, *lat* denotes the longitude and latitude of the position, *time* denotes the timestamp when the position is recorded.

Maps available and provided by OpenStreetMap are used to build a road network that contains totally 52 479 road segments and 49 773 intersections. After building a road network, it is of vital importance to mapping GPS trajectories into corresponding roads, which is also called map matching, which purpose is to integrate the positioning data with the spatial road network data, to identifying the actual way on which the vehicle is traveling and further to determine the vehicle location on that path. For each GPS record, the map matching is processed in three steps: (1) Identifying possible road segments, (2) Identifying candidate road segments, and (3) Weighting candidate road segments.

[2] http://akeranen.github.io/the-one/.
[3] https://1drv.ms/f/s!AvvP9GtlhRdqawzHyhbu-ZbfFdQ.

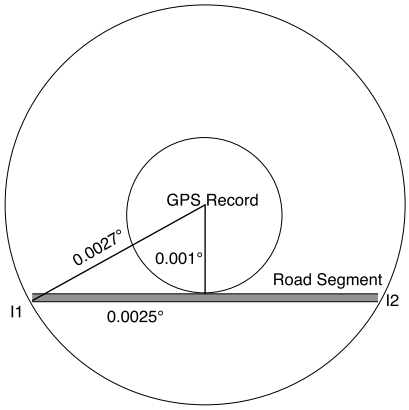


Fig. 3. The maximum distance appears when the road segment is tangent to the circle and the tangent point is the midpoint.

5.1.1. Identifying possible road segments

For each GPS record, it is inefficient yet complex to match all possible road segments to search suitable road segments to the record. Rather, we only need to identify a few road segments that cover all possible segments for the GPS record whilst filter others. GPS location errors can be as large as 100 m in a city with dense tall buildings and viaducts. As a matter of fact, 100 m can be roughly regarded as 0.001 latitude or longitude. That is, imagine a circle of radius 0.001 latitude or longitude centered at the GPS record, the GPS record can only reside on the road segments that intersect or tangent to the circle. From our investigation, 99.27% road segments in the road network are less than 0.005 latitude or longitude long, as described in Fig. 3.

Therefore, we test each road segment on the following criteria: whether there exists a road segment that meets the condition that the distance between the endpoint and the GPS record is less than 0.0027 latitude or longitude long. The GPS record would not reside on a road segment that fails to meet this criteria.

5.1.2. Identifying candidates to road segments

After obtaining all possible road segments of a vehicle's GPS record, we need to identify candidates to road segments satisfying a number of basic conditions.

First, we need to check whether the distance between the road segment and the GPS record is less than the distance threshold, which is set to 0.001 latitude or longitude as depicted before. To calculate the distance between the road segment and the GPS record, we need to calculate the closest point on the road segment, and can be achieved by projecting the location of GPS record onto the road segment. If the projection point lies on the road segment, the distance between the road segment and the GPS record turns to be the distance between the projection point and the GPS record. Otherwise, if the projection point lies outside the segment, we should calculate the distances from the GPS record to the road segment's two terminal points, and choose the shorter one.

Next, we also need to confirm the difference between the headings of the road segments and the driving direction is less than 60°, a heading difference threshold used in Pfoser's work [33], in which the direction of road segment can be calculated using its two terminal points (first and second points). There are two directions on one road segment, and the smallest value as the difference between road segment and vehicle heading is selected. If the difference is less than the threshold, the road segment will be identified as a candidate road segment; otherwise, the road segment is not considered as a candidate road segment even if the GPS record-Road segment distance is short.

Table 2
Default parameters of the simulations.

Parameter	Default value	Description
n_RSU	60	Number of RSUs
n_v	506	Number of vehicular nodes
$\gamma_{x_{*,0}}$	$(1, -5]$	Time of data processing at cloud or RSUs
η_{4g}	0.5 s	Waiting time to establish 4G connection
n_time	16 h	Total simulation duration
d_sense	60 s	Time interval of data sensing
q_rate	0.1/m	Query rate of nodes
bound	$N(600, 40)$ s	Time bound of query
BW_{4g}	20/5 Mbps	Download/upload bandwidth of 4G channel
BW_v	500/250 Kbps	Download/upload bandwidth of V2V and V2I
$c_{u,4g}$, $c_{i,j}$	10^{-2}, 10^{-4} $/MB	Cost of the 4G and V2V channel
store	2 G	Storage space in RSUs
α	0.5	$\in [0, 1]$, the balance factor in Eq. (17)

5.1.3. Weighting candidate segments

Once identified a set of candidate road segments for each GPS record, each road segment is given a weight based on the following two factors: (1) proximity between the locations of GPS record and the road segment, (2) similarity between the vehicle heading and the direction of road segment. Given one GPS record g and its k candidate road segments $S = s_1, s_1, \ldots, s_k$, the score of one candidate road segment s_i can be computed as:

$$score(g, s_i) = \theta_1 \frac{dis\,(g, s_i)}{0.001} + \theta_2 \frac{\alpha\,(g, s_i)}{60^\circ} \tag{19}$$

where θ_1 and θ_2 are the weights of location proximity and direction similarity respectively. $dis\,()$ is the function of the distance between one GPS record and one road segment, and $\alpha\,()$ is the angle difference function between them. After the score is calculated for each candidate road segment from the GPS record, the road segment that has the minimal score is considered as the matched road of the GPS record on the map.

After the map matching, a speed network of roads is generated according to the trajectories described in Section 4.3.3, where the speed of the vehicles ranges from 5 to 80 km/h and differs according to road segments and time periods. Fig. 4 depicts the snapshots of speed network of Xiamen City, China, limited to a rectangle area of $[118.0660E, 118.0990E] \times [24.4300N, 24.5300N]$.

5.2. Experimental environment setup

There are 120 RSUs evenly deployed along the roads. The communication range of I2I or I2V used by the vehicles to exchange data is set to 60 m, yet the 4G channel does not have the limit of communication range. As defined in Eq. (1), the time needed for the data processing at the cloud or RSUs, i.e. $\gamma_{x_{*,0}}$, is set around 1–5 s, and the waiting time to establish the connection η_{4g} is set 0.5 s.

The total simulation time is set to 16 h within a day, from 7:00 to 23:00, and every node senses a data reading every 60 s. Data samples consist of pictures, which are selected in random from the Google Open Images dataset [34], ∼9 million URLs to images that have been annotated with image-level labels and bounding boxes spanning thousands of classes. To simulate different sizes of sensing data, 1 to 5 images from the dataset are bundled together as one sample of the sensing. A query is defined as picture retrieval that have been generated through the sensing process given a label or tag. The time bound for the query follows a normal distribution: t∼N(600 s, 40), and every node generates a query at an average rate of 0.1 query/minute. The Elasticsearch platform[4] is used as the storage and search engine at the cloud, and GLPK for Java[5] is used as a solver for the QRF problem.

4 https://www.elastic.co/products/elasticsearch/.
5 http://glpk-java.sourceforge.net/.

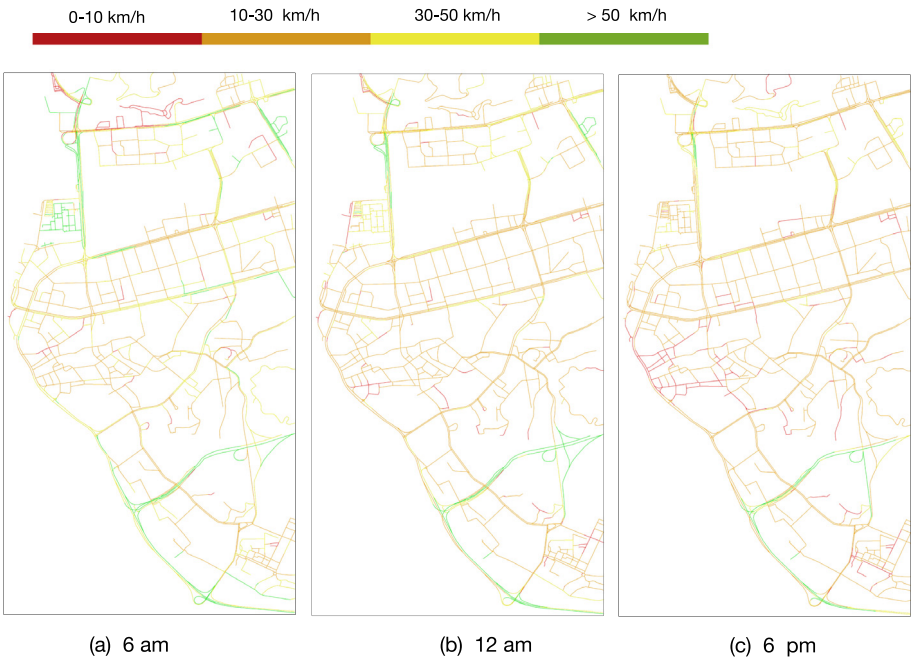


Fig. 4. Snapshots of speed network of Xiamen City, China, limited to an area of $[118.0660E, 118.0990E] \times [24.4300N, 24.5300N]$.

We use the ratio of successful queries and the total cost of query processing as the main metrics for the performance analysis. A query is successfully processed only when its results are returned to the requester before the bounded time. The bandwidth of the 4*G* channel is set 20 Mbps/5 Mbps for the down/up links, the bandwidth of the V2V or V2I channel is 500 Kbps/250 Kbps for the down/up links, and the bandwidth of the I2I channel is 124 Mbps. The cost of the channel $c_{u,4g}$ is set 10^{-2} \$/MB and $c_{i,j}$ is set 10^{-4} \$/MB. The default parameters of the simulation are summarized in Table 2.

It is worth to note that ideal links are assumed below the application layer. That is, as two nodes meet and establish a connection, the query request, query result and metadata can be wrapped into one message respectively. Issues from other layers are not considered, for instance, MAC and channel competition. Indeed, vehicular network is a broad research area, so Cross-layer optimization is needed to have better efficiency, yet it is out of the scope of this paper.

5.3. Compared schemes

Besides CASQ, four other schemes are also implemented for the comparison purposes:

- (1) Centralized: all sensed data are uploaded to a centralized cloud server through 4G, and queries are processed on the cloud;
- (2) Flooding: each vehicle receiving the query from one of its neighbors relays it, and the query results are flooded back to the query requester;
- (3) FleaNet [16]: query requester periodically advertises the query only to its one-hop neighbors, seeking if they can provide any answer from data stored on their local storage;
- (4) GeoVanet [17]: it uses a DHT-based model that identifies a fixed geographical location (RSU) where a mailbox is dedicated to the query, permitting the user to retrieve his/her results within a bounded time.

Table 3
Comparison of performance of the algorithms.

	Query ratio (%)	Query delay (s)	Cost ($)
Centralized	100.00	2.85	2223.49
Flooding	34.25	549.23	55.18
FleaNet	9.01	591.34	12.13
GeoVanet	53.79	604.11	29.15
CASQ	**94.26**	**594.57**	**286.51**

5.4. Experimental analysis

5.4.1. Overall performance

Based on the trajectory dataset, about $4.85 * 10^5$ samples are sensed and $4.8 * 10^4$ queries are generated. Table 3 shows the performance of schemes aforementioned. The Centralized scheme uploads all the data through the 4G channel, and stores them at the cloud. So it has all the queries successfully processed, with a query radio of 100% and a relatively small query delay (2.85 s). Yet it also incurs the largest workload for the telephony network, whose unit cost is expensive (10^{-2} \$/MB). The total cost is about 2223.49 dollars for the data upload and query result downloaded. In contrast, the Flooding, FleaNet and GeoVanet only depend on the in-network DSRC communications for the query processing. The total cost is less than 60 dollars, given the small unit cost of V2V or I2V communications (10^{-4} \$/MB). However, the query ratio is much lower. It is less than 55% for all three schemes. The query radio of FleaNet is about 9.01% while the Flooding is about 34.25%. This is due to, the query in FleaNet is only forwarded to one-hop neighbors, while in Flooding scheme queries and results are flooded, where more nodes could receive the query and hence have more probability to answer the query and return the results. In GeoVanet, queries are forwarded to fixed RSUs for storage, so the query results could be fetched by the query requester. Nevertheless, this scheme has relatively higher query radio, about 53.79% of the queries are successfully processed, and the cost is as low as about 29.15 dollars. Besides the Centralized scheme, the query delay is close to the time bound of the queries, of about 600 s. Note that the expired queries are not accounted for the delay calculation.

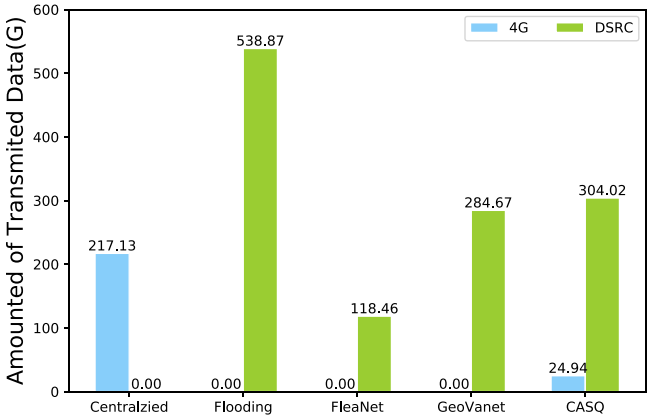


Fig. 5. Amounted of transmitted data through the 4*G* and DSRC channels.

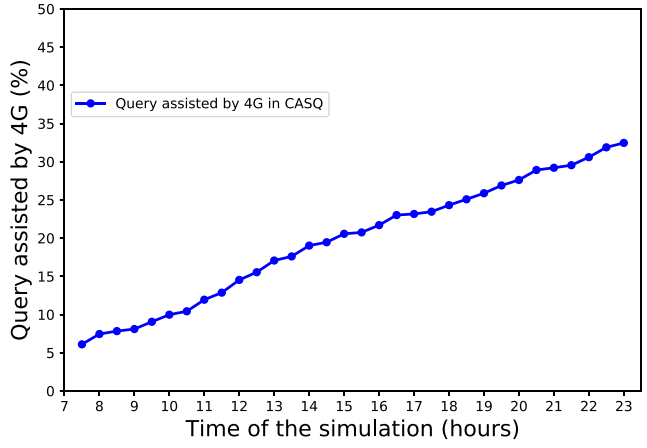


Fig. 6. Accumulated percentage of queries assisted by 4G.

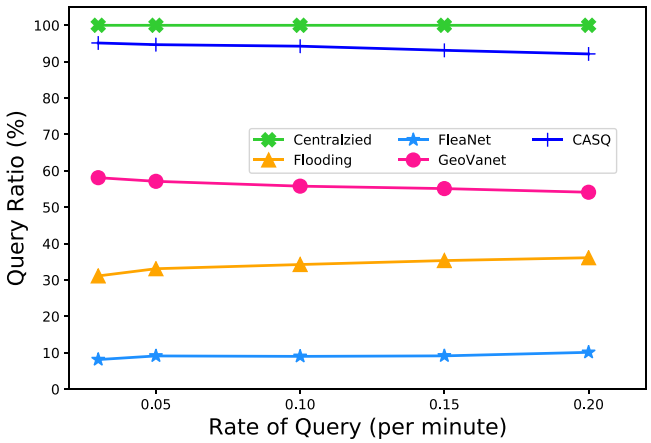


Fig. 7. Impact of query rate to the query ratio.

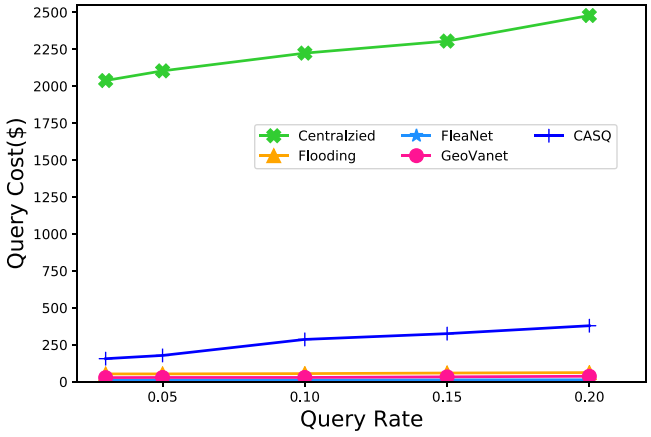


Fig. 8. Impact of query rate to the cost of query.

Fig. 5 shows the amount of transmitted data through the 4*G* and DSRC channels. In the Centralized scheme about 217 G sensed data are routed to the cloud through the 4G network, while in Flooding, FleaNet, GeoVanet scheme, the amount of data transmitted through the DSRC channel are 538.87, 118.46, and 284.67 G respectively. These schemes depend solely on the DSRC transmissions, so there are no 4G transmissions. CASQ adopts a hybrid approach for the query processing, where the amount of transmitted data through 4G and DSRC are 24.94 and 304.02 respectively. The 4G channel is adopted to forward results of the query requester, making large number of queries be processed on time. As shown in Fig. 6, about 31.2% of the queries in CASQ are assisted by the 4G channel, and at least part of the query results are forwarded through the *Path*2 to the query requester. Thus, the proposed CASQ scheme has a query radio as high as 94.26%, also due to the cloud playing a vital role on the indexing of data, and supports to forward the queries to RSUs that have the query answers, speeding up the query processing and avoids query expiration. Without the assistance of the cloud and the 4G channel, these queries would otherwise break the bounded time and be dropped. On the other hand, this increases the cost of the query processing. It costs about 286.51 dollars, about 5 times more than that of the Flooding scheme, yet still much cheaper than that of the Centralized scheme. The economic cost of CASQ is about 12.8% of the Centralized scheme.

In our experiments, the movement of vehicular nodes follows the historical trajectories in real-world datasets, rather than synthesized datasets. Thus, we do not change the environment setting such as network size or speeds to study such impacts.

5.4.2. *Flexibility of queries*

Pull-based model provides more flexibility in terms of queries. Queries could be forwarded to various areas through multi-hop transmissions, and the types of queries, the filters, the query rates, and the bounded time of query could be varied according to users's requirements. At this subsection, we study the impact of query parameters.

Figs. 7 and 8 show the impact of query rate to the successful query ratio and query cost, where larger query rate generates larger number of queries. The main part of the Centralized scheme is to upload the sensed data to the cloud, where query results are extracted and routed back to the requester. Thus, as the query rate grows, the cost of the query increases from 2038 dollars to 2471 dollars. This is due to larger amount of query results should be downloaded to the requesters. In the Flooding, FleeNet and GeoVanet schemes, the sensed data are stored within the network. The query ratios increase a little bit with the query rate, as well the costs increase about 20–30 percent accordingly, given that more queries are to be forwarded to the encountered nodes, noting that the impact of query rate is relatively small on these schemes. For the CASQ scheme, the ratio of successful queries are barely not affected with the increase on the number of queries, yet the cost increases from 156.3 dollars to 379.1 dollars as the query rate grows from 0.05 to 0.20. This is due to larger amount of query results should be forwarded to the requester. Additionally, due to limited bandwidth of the I2V channel, more data have to be routed through the 4G channel to the requester, which leads to an increase for the overall cost.

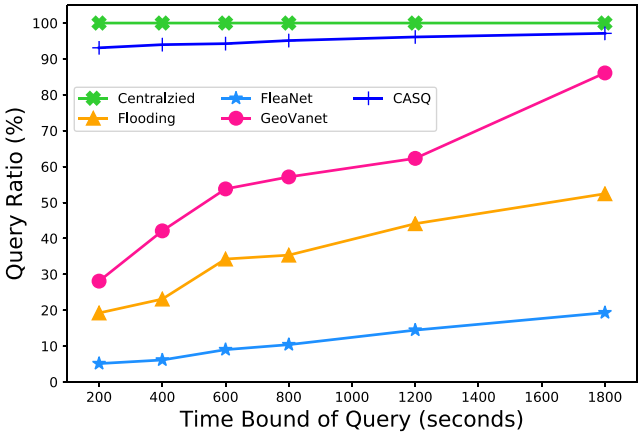


Fig. 9. Impact of bounded time to the query ratio.

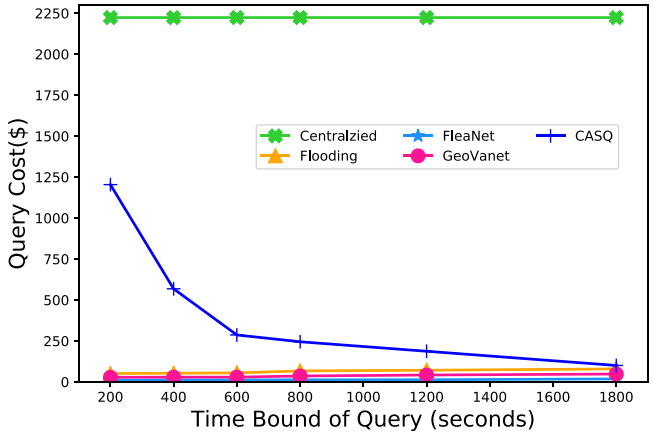


Fig. 10. Impact of bounded time to the cost of query.

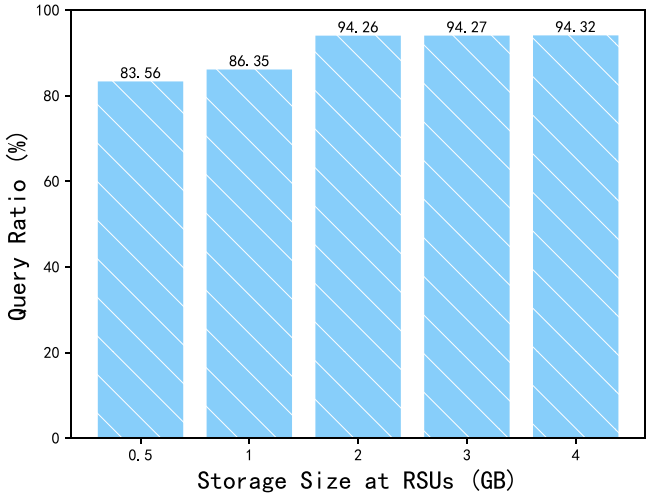


Fig. 11. Impact of storage size in RSUs.

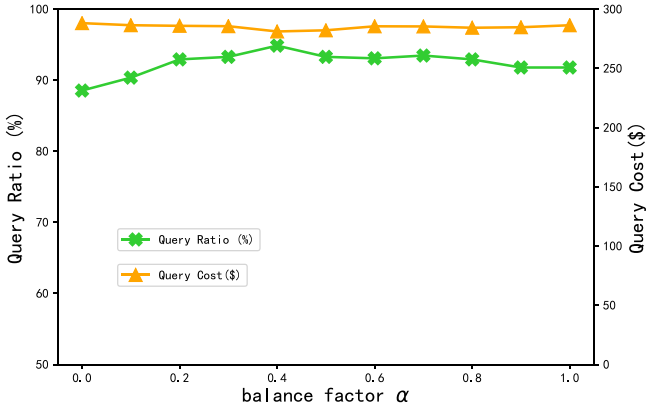


Fig. 12. Impact of balance factor α.

Figs. 9 and 10 show the impact of the bounded time of query to the successful query ratio and query cost. Such ratios increase with the time bound as more time is given for the query processing. When the time bound is small, e.g. 200 s, the ratios are less than 30% for all the in-network query processing algorithms. This is due to the query and results are routed through a 'carry-store-forwarded' style, and delays are common for pieces of data to be routed to the destination. Thus, lost of queries are expired and dropped because there are not enough time. When the bounded time grows, the ratio increases. This is especially true for the GeoVanet scheme that has the problem of routing back the query results through multi-hop communications. Data are stored at a hashed location, the queries are forwarded to that location to extract the results, which are finally fetched by the requester through another request. The ratio of the GeoVanet scheme grows from 28% to 86% when the bounded time increases from 200 s to 1800 s.

For the CASQ scheme, the ratio grows tiny with the bounded time, yet the cost decreases from 1204.6 dollars to 102.5 dollars as the bounded time increases from 200 s to 1800 s. The scheme would select the *Path*1 strategy through 4*G* channel to route back the query result when the bounded time is small, which leads to an increase of the overall cost. In contrast, if there is enough time for data to be forwarded to the requester, the cheaper channel of DSRC is chosen to transmit the data. Generally speaking, the query schemes benefit from the increased bounded time, but CASQ could be adaptively tuned to achieve a high ratio of successful queries when increasing the budget of query processing.

5.4.3. *Impact of storage size at RSUs*

As discussed in Section 4.4, RSUs cache the sensed data that might be queried by the users. Therefore, the storage size of RSUs has also impact on the proposed scheme.

Fig. 11 depicts the impact of the storage size at RSUs. From the figure, we could note that the query ratio grows with the storage size. It is clear that the larger storage space, the more data segments could be cached and successfully queried. The ratio of success increases from 83.56% to 94.26% when the storage size increases from 0.5 to 2 GB. Nevertheless, such ratio does not increases when the storage size turns larger, i.e., to 3 or 4 GB. This is because part of the failed queries are due to the movement of vehicles, which would expire the queries if results are not able to be forwarded back to the query requester. For this part of failed queries, merely increasing the cached storage size at RSUs would not help.

Fig. 12 depicts the impact of α defined at Eq. (17), which is a balance parameter. From this figure, we could see that the query ratio increases with α, as it grows to the maximal at about 94.86% when α is 0.4 then decreases to 91.79% when α is updated to 1.0. This reflects the tradeoff between the 'freshness' and 'hotness' of the sensed data. When α is small, 'hotness' dominates the weight and data that are queried more often would be reserved, while the newly sensed data would be deleted when there is not enough storage space. When α is large, the opposite case occurs. In this dataset and simulation scenario, CASQ achieves the best performance when α is around 0.4. The balance factor has relatively small

impact on the query cost, which stays stable at around 286∼288 dollars when α varies.

6. Conclusions and future works

In this paper, we have proposed an adaptive and cloud-assisted data storage and query processing scheme in VANETs. Data readings are stored at local nodes or RSUs, and indexed by the cloud. Queries are then processed at the cloud and directed to RSUs to extract the query results. The cloud calculates a result forwarding plan by solving the QRF problem, where the query results choose their best paths either through the 4G or the DSRC communication channels. Finally, query results are diffused to RSUs along the traveling path of the query requester, so it can fetch the results just before the query is outdated. Experimental results demonstrate the effectiveness of the proposed algorithm in vehicular sensor networks. Up to 94% of the queries could be successfully processed in the proposed scheme, much higher than existing query schemes, while at the same time incurring a relatively low querying cost.

As future work, we will investigate more types of queries within the framework of CASQ. For example, the time or location based filter would be considered in the queries, and more efficient and compressed indexes at the cloud would be explored. In addition, we are going to study the impact of traffic patterns to optimize the query processing procedures in VANETs.

References

[1] U. Lee, E. Magistretti, M. Gerla, P. Bellavista, A. Corradi, Dissemination and harvesting of urban data using vehicular sensing platforms, IEEE Trans. Veh. Technol. 58 (2) (2009) 882–901, http://dx.doi.org/10.1109/TVT.2008.928899.

[2] Y. Lai, F. Yang, L. Zhang, Z. Lin, Distributed public vehicle system based on fog nodes and vehicular sensing, IEEE Access 6 (2018) 22011–22024.

[3] J. Zhao, G. Cao, Vadd: Vehicle-assisted data delivery in vehicular ad hoc networks, IEEE Trans. Veh. Technol. 57 (3) (2008) 1910–1922.

[4] Y. Lai, F. Yang, J. Su, Q. Zhou, T. Wang, L. Zhang, Y. Xu, Fog-based two-phase event monitoring and data gathering in vehicular sensor networks, Sensors 18 (1) (2017) 82.

[5] T. Wang, Y. Li, G. Wang, J. Cao, M.Z.A. Bhuiyan, W. Jia, Sustainable and efficient data collection from wsns to cloud, IEEE Trans. Sustain. Comput. (2017).

[6] T. Wang, J. Zeng, Y. Cai, H. Tian, Y. Chen, B. Wang, et al., Data collection from wsns to the cloud based on mobile fog elements, Future Gener. Comput. Syst. (2017).

[7] S. Al-Sultan, M.M. Al-Doori, A.H. Al-Bayatti, H. Zedan, A comprehensive survey on vehicular Ad Hoc network, J. Netw. Comput. Appl. 37 (2014) 380–392.

[8] A. Dua, N. Kumar, S. Bawa, A Systematic Review on Routing Protocols for Vehicular Ad hoc Networks, Vol. 1, Elsevier, 2014, pp. 33–52.

[9] Y. Lai, L. Zheng, T. Wang, F. Yang, Q. Zhou, Cloud-assisted data storage and query processing at vehicular ad-hoc sensor networks, in: The Third International Symposium on Sensor-Cloud Systems, Springer, 2017, pp. 692–702.

[10] D.J. Abadi, S. Madden, W. Lindner, Reed: Robust, efficient filtering and event detection in sensor networks, in: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, pp. 769–780.

[11] A. Silberstein, Push and pull in sensor network query processing, in: Southeast Workshop on Data and Information Management (SWDIM06), Raleigh, North Carolina, 2006.

[12] Y. Xu, S. Helal, M. Scmalz, Optimizing push/pull envelopes for energy-efficient cloud-sensor systems, in: Proceedings of the 14th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems, ACM, 2011, pp. 17–26.

[13] U. Lee, B. Zhou, M. Gerla, E. Magistretti, P. Bellavista, A. Corradi, Mobeyes: smart mobs for urban monitoring with a vehicular sensor network, IEEE Wirel. Commun. 13 (5) (2006) 52–57.

[14] C.E. Palazzi, F. Pezzoni, P.M. Ruiz, Delay-bounded data gathering in urban vehicular sensor networks, Pervasive Mob. Comput. 8 (2) (2012) 180–193.

[15] M. Motani, V. Srinivasan, P.S. Nuggehalli, Peoplenet: engineering a wireless virtual social network, in: Proceedings of the 11th Annual International Conference on Mobile Computing and Networking, ACM, 2005, pp. 243–257.

[16] U. Lee, J. Lee, J.-S. Park, M. Gerla, Fleanet: A virtual market place on vehicular networks, IEEE Trans. Veh. Technol. 59 (1) (2010) 344–355.

[17] T. Delot, N. Mitton, S. Ilarri, T. Hien, Geovanet: A routing protocol for query processing in vehicular networks, Mob. Inf. Syst. 7 (4) (2011) 329–359.

[18] Y. Zhang, J. Zhao, G. Cao, Roadcast: a popularity aware content sharing scheme in vanets, ACM SIGMOBILE Mobile Comput. Commun. Rev. 13 (4) (2010) 1–14.

[19] Q. Xu, T. Mak, J. Ko, R. Sengupta, Vehicle-to-vehicle safety messaging in dsrc, in: Proceedings of the 1st ACM International Workshop on Vehicular ad hoc Networks, ACM, 2004, pp. 19–28.

[20] Q. Xu, H.T. Shen, Z. Chen, B. Cui, X. Zhou, Y. Dai, Hybrid retrieval mechanisms in vehicle-based p2p networks, in: International Conference on Computational Science, Springer, 2009, pp. 303–314.

[21] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, S. Shenker, Ght: a geographic hash table for data-centric storage, in: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications, ACM, 2002, pp. 78–87.

[22] B. Paczek, Selective data collection in vehicular networks for traffic control applications, Transp. Res. C Emerg. Technol. 23 (2012) 14–28, data Management in Vehicular Networks.

[23] M. Eltoweissy, S. Olariu, M. Younis, Towards autonomous vehicular clouds, in: International Conference on Ad Hoc Networks, Springer, 2010, pp. 1–16.

[24] Y.C. Hu, M. Patel, D. Sabella, N. Sprecher, V. Young, Mobile edge computinga key technology towards 5g, ETSI White Paper 11 (11) (2015) 1–16.

[25] J. Zeng, T. Wang, Y. Lai, J. Liang, H. Chen, Data delivery from wsns to cloud based on a fog structure, in: Fourth IEEE International Conference on Advanced Cloud and Big Data (3), IEEE, 2016, pp. 959–973.

[26] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ACM, 2012, pp. 13–16.

[27] K. Kai, W. Cong, L. Tao, Fog computing for vehicular ad-hoc networks: paradigms, scenarios, and issues, J. China Univ. Posts Telecommun. 23 (2) (2016) 56–96.

[28] J.B. Kenney, Dedicated short-range communications (dsrc) standards in the united states, Proc. IEEE 99 (7) (2011) 1162–1182.

[29] Y. Lai, Z. Chen, W. Wu, T. Ma, Multiple-resolution content sharing in mobile opportunistic networks, Wireless Commun. Mobile Comput. 15 (16) (2015) 1991–2003.

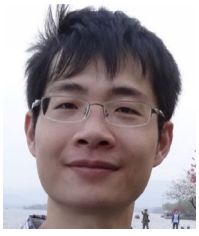
[30] K.G. Murty, Linear Programming, John Wiley & Sons, 1983.

[31] S. Reveliotis, An Introduction to Linear Programming and the Simplex Algorithm, School of Industrial and Systems Engineering, Georgia Institute of Technology, 1997.

[32] A. Keränen, J. Ott, T. Kärkkäinen, The one simulator for dtn protocol evaluation, in: SIMUTools '09: Proceedings of the 2nd International Conference on Simulation Tools and Techniques, ICST, New York, NY, USA, 2009.

[33] D. Pfoser, C.S. Jensen, Capturing the uncertainty of moving-object representations, Vol. 1651, 2002, pp. 111–131.

[34] I. Krasin, T. Duerig, e.a. Alldrin, Openimages: A public dataset for largescale multi-label and multi-class image classification., Dataset available from https://github.com/openimages, 2017.



Yongxuan Lai received the Ph.D. degree in computer science from Renmin University of China in 2009. He is currently an associate professor in Software School, Xiamen University, China. His research interests include network data management, vehicular ad-hoc networks, big data management and analysis.

Lu Zhang is a postgraduate student in the Department of Software Engineering at Xiamen University, China. Her research interest includes vehicular networks and data mining using trajectories.

Fan Yang received the Ph.D. degree in control theory and control engineering from Xiamen University, Xiamen, China, in 2009. He is currently an associate professor in the Institute of Pattern Recognition & Intelligent Systems, Department of Automation at Xiamen University. His current research interests include mobile computing, pattern recognition, data mining and bioinformatics. He has published more than 50 journal articles and conference papers.

Tian Wang received the B.Sc. and M.Sc. degrees in computer science from Central South University in 2004 and 2007, respectively, and the Ph.D. degree from the City University of Hong Kong in 2011. He is currently a Professor with National Huaqiao University, China. His research interests include wireless sensor networks, fog computing, and mobile computing.

Kuan-Ching Li received the Ph.D. and M.S. in Electrical Engineering and Licenciatura in Mathematics from University of Sao Paulo, Brazil in 2001, 1996 and 1994, respectively. Professor Li is currently a Professor in the Department of Computer Science and Information Engineering at Providence University, Taiwan. His research interests include cluster and grid computing, vehicular networks, parallel software design, and life sciences computing.