Mobile Netw Appl (2010) 15:83–96
DOI 10.1007/s11036-009-0170-9

Service Scheduling of Vehicle-Roadside Data Access

Yang Zhang · Jing Zhao · Guohong Cao

Published online: 16 May 2009

Abstract As vehicular networks become popular, more and more people want to access data from their vehicles. When many vehicles want to access data through a roadside unit, service scheduling becomes an important issue. In this paper, we identify some challenges in vehicle-roadside data access. As vehicles move pretty fast, the requests should be served quickly. Also, vehicles may upload data to the roadside unit, and hence the download and upload requests compete for the same bandwidth. To address these challenges, we propose several service scheduling schemes. We first propose a basic scheduling scheme called $\mathcal{D} * \mathcal{S}$ to consider both service deadline and data size. We then enhance it by using a single broadcast to serve multiple requests. Finally, we identify the effects of upload requests on data quality, and propose a Two-Step scheduling scheme to provide a balance between serving download and update requests. Simulation results show that the Two-Step scheduling scheme outperforms other scheduling schemes.

Keywords roadside unit (RSU) · upload/download · data quality · scheduling

Y. Zhang (✉) · J. Zhao · G. Cao
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA
e-mail: yangzhan@cse.psu.edu

J. Zhao
e-mail: jizhao@cse.psu.edu

G. Cao
e-mail: gcao@cse.psu.edu

1 Introduction

Recently, vehicle-roadside data access has received considerable attention [1–5]. With RoadSide Unit (RSU) such as 802.11 access point, vehicles can access data stored in the RSU or even access the Internet through these RSUs. From 2003, the US Department of Transportation has invested millions of dollars [6] to integrate vehicles and RSUs and to make the current transportation system more intelligent. Chrysler-Daimler has introduced the "InfoFuel" system to provide Mercedes drivers the ability to access wireless data through roadside hotspots (http://www.leearmstrong.com/DSRC/DSRCHomeset.htm). Also, FCC dedicates the 5.9 GHz frequency specifically allocated to vehicle-vehicle and vehicle-roadside communications and enacts the Dedicated Short Range Communications (DSRC) standard in 2004. Later, IEEE develops several standards to ensure that vehicles and roadside infrastructures can communicate with each other. All these efforts in academy, government, and industry make vehicle-roadside data access mature enough to be used in our daily life.

In vehicle-roadside data access, the RSU can act as a router for vehicles to access the Internet. Although this can bring many benefits to the drivers, the deployment cost and maintenance cost are very high. As another option, RSU can also be just used as a buffer point (or data island) between vehicles. In this paper, we focus on the latter paradigm due to its low cost and easy deployment. In this paradigm, all data on the RSUs are uploaded or downloaded by vehicles. For example, some data, especially those with spacial/temporal constraints, only need to be stored and used locally. The following applications also belong to this case where the

data are buffered at the RSUs, and will not be sent to the Internet.

1. Value-added Advertisement and Content Sharing: Store owners may want to advertise their sale or activity information in nearby area. Without Internet connection, they can ask the running vehicles to carry and upload the advertisement information to nearby RSUs [7] and to share with each other [8]. At the same time, other vehicles driving around can download these advertisements and visit the stores.
2. Real-Time Traffic: Vehicles can report real-time traffic observations to RSUs [9]. The traffic data are stored at RSUs, providing real-time query and notification services to other vehicles. The data can be used to provide traffic conditions and alerts such as road congestion and accidents.
3. Digital Map Downloading: Due to the storage limitations of memory card and frequent road construction, it is impossible for vehicles to install all the most up-to-date digital maps before traveling. Hence, vehicles driving to a new area may hope to update map data locally for travel guidance.

Different from traditional data access system in which users can always wait for the service from the data server, vehicles are moving and they only stay in the RSU area for a short period of time. As a result, there is always a time constraint associated with each request. Meanwhile, to make the best use of the RSU and to share the information with as many vehicles as possible, RSUs are often set at the roadway intersections or areas with high traffic. In these areas, download (query) requests retrieve data from the RSU, and upload (update) requests upload data to the RSU. Both download and upload requests compete for the same limited bandwidth. As the number of users increases, what request to be served at what time will be critical to the system performance. Hence, it is important to design an efficient service scheduling algorithm for vehicle-roadside data access. In this paper, we design efficient solutions for scheduling vehicle-roadside data access. Our contributions are as follows.

1. We first propose a basic low complexity scheduling scheme called $\mathcal{D} * \mathcal{S}$ which considers both *data size* and *request deadline*.
2. We improve the performance of the basic scheduling algorithm by using *broadcasting* techniques to serve more requests.
3. We study the tradeoffs between *service ratio* and *data quality*, and propose a *Two-Step* scheme to address the tradeoffs between uploads and downloads;
4. We conduct extensive simulations to study the performance of the proposed scheduling schemes.

The rest of this paper is organized as follows: the related work is summarized in Section 2. Section 3 presents the background and necessary preliminaries. Section 4 describes the limitation of three naive schemes and presents a new scheduling scheme called $\mathcal{D} * \mathcal{S}$. In section 5, we study how to optimize scheduling by broadcasting. An improved scheme called $\mathcal{D} * \mathcal{S}/\mathcal{N}$ is proposed. We discuss the impact of data staleness and propose a two-step scheduling scheme in Section 6. Section 7 evaluates the performance of the proposed schemes. Finally, we conclude the paper in Section 8.

2 Related work

Vehicle-roadside data access is an important component for data access in vehicular networks. The authors in [1] illustrated a basic picture of how running vehicles contact with roadside hot spots through a "drive-thru" data access. They gave us the first understanding of the impact of the vehicle's speed, transmission rate, 802.11 bit-rate, and packet size on throughput and delay of vehicle-roadside communication. Bychkovsky et al. [2] did another experiment to confirm the advantage of vehicle-roadside data access. Their findings showed that there are plenty of Wi-Fi networks in residential areas in and around cities. The unplanned, community-driven "openWi-Fi" wireless network composed of *in situ* access points can provide feasible and abundant resources for vehicle-roadside data access. Zhao et al. [10] studied to use roadside infrastructures to store and rebroadcast their buffered data to "drive-thru" vehicles at the intersection. Further, several mobile computing systems such as CarTel [3] and MobiEyes [9] were designed and implemented by MIT and UCLA, respectively to collect, process and deliver data from sensors located on vehicles to roadside infrastructures for analysis. The similar projects include the DieselNet at Umass [11], FleetNet in Germany [12], InternetCAR in Japan [13]. Although these works discuss how to benefit from vehicle-roadside data access, they do not study the prerequisite problem: *if multiple vehicles want to access the roadside infrastructure at the same time, how to schedule their service priorities.* Scheduling is an important issue for data access in vehicle networks. In this work, we will study the application-layer scheduling for vehicle-roadside data access and give the most optimized solutions.

There are a large amount of work related to CPU and job scheduling in the literature. Wong studied several scheduling algorithms such as first-come-first-serve (FCFS), longest wait time (LWT), most requests first (MRF) in the broadcasting environments [14]. Later, many broadcast scheduling algorithms have been proposed to reduce the waiting time and energy consumption [15–18]. Acharya and Muthukrishnan [19] addressed the broadcast scheduling problem in heterogeneous environments, where data items have different sizes. The solution is based on a new metric called *Stretch*, defined as the ratio of the response time of a request to its service time. Based on stretch, they proposed a scheduling algorithm, called longest total stretch first (LTSF) to optimize the stretch and achieve a balance between the worst case and the average case. However, a straightforward implementation of LTSF is not practical for a large system, as at each broadcast time, the server has to recalculate the total stretch for every data item with pending requests in order to decide which data to broadcast next, and hence the scheduling algorithm becomes a bottleneck due to the high computation overhead. The work in [20] introduced the concept of Quality Contracts (QCs) which combines the two incomparable performance metrics: response time or Quality of Service (QoS), and staleness or Quality of Data (QoD). QCs allows individual users to express their preferences for the expected QoS and QoD of their queries by assigning "profit" values. They proposed an adaptive algorithm to maximize the total profit from submitted QCs. This work is also based on point-to-point communication and does not take advantage of broadcasting. All these works mainly focus on responsiveness such as average/worst-case waiting time or fairness without considering the time constraints of the user requests. However, in vehicular networks, time constraint of the request has to be considered.

Jiang and Vaidya [21], Rajan et al. [22] and Xu et al. [23] studied the scheduling problem in real-time broadcasting environment and took time constraint into account. Jiang and Vaidya [21] and Rajan et al. [22] investigated only periodic push-based broadcast, which is fundamentally different from on-demand broadcast in system architecture. The authors of [23] investigated online scheduling algorithms for time critical on-demand data broadcast. However, they ignored the data update issue. They assumed that data are read only or can only be updated by the server. Hence, they only tried to improve the service ratio for download broadcasting. In contrast, our vehicle-roadside data access model is different as both update and download compete for the same bandwidth. Also, missing the update degrades the data quality.

3 Background and preliminaries

3.1 System model

As shown in Fig. 1, a large number of vehicles retrieve (or upload) their data from (or to) the RSU when they are in the communication range. The RSU (server) maintains a service cycle, which is non-preemptive; i.e., one service can not be interrupted until it finishes. When one vehicle enters the RSU area, it listens to the wireless channel. All vehicles can send requests to the RSU if they want to access the data. Each request is characterized by a 4-tuple: $<$*v-id*, *d-id*, *op*, *deadline*$>$, where *v-id* is the identifier of the vehicle, *d-id* is the identifier of the requested data item, *op* is the operation that the vehicle wants to do (upload or download), and *deadline* is the critical time constraint of the request, beyond which the service becomes useless. All requests are queued at the RSU server upon arrival. Based on the scheduling algorithm, the server serves one request and removes it from the request queue.

Different from traditional scheduling services, data access in vehicular networks has two unique features: 1) The arrival request is only active for a short period of time due to vehicle moving and coverage limitations of RSUs. When vehicles move out of the RSU area, the requests not served have to be dropped; and 2) Data items can be downloaded and uploaded from the RSU server. The download and update requests compete for the service bandwidth.

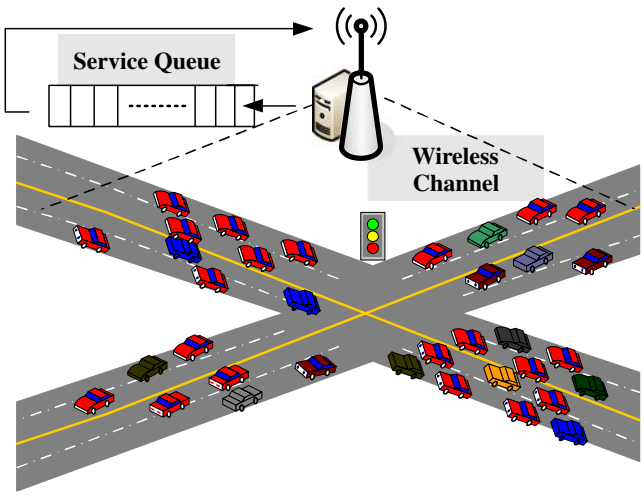


Fig. 1 The architecture of vehicle-roadside service scheduling

We assume that each vehicle knows the service deadline of its request. This is reasonable because when a vehicle with GPS device enters the coverage area of a RSU, it can estimate its leaving time based on the knowledge of its driving velocity and its geographic position.[1]

3.2 Performance metrics

In most previous work, the metrics for scheduling algorithms are responsiveness (e.g., average/worst-case waiting time [15, 17, 18]) or fairness (e.g., stretch [19, 24]). In most of these works, requests do not have time constraints and the data on the server are not updated or updated only by the server. However, in the vehicle-roadside data access scenario, requests not served within a time limit will be dropped as the vehicles move out of the RSU area. Since update requests compete bandwidth with other download requests, some data may become stale after an update is missed, degrading the service quality. Therefore, compared with responsiveness and fairness, providing fresh data to more vehicles is more important and we use the following metrics for scheduling vehicle-roadside data access.

1. *Service Ratio*: Service ratio is defined as the ratio of the number of requests served before the service deadline to the total number of arriving requests. A good scheduling scheme should serve as many requests as possible.
2. *Data Quality*: Data will become stale if a vehicle has the new version of the data but fails to upload it before the vehicle moves out of the RSU range. The staleness of the data will degrade the data quality for the download service. In this paper, we use the percentage of fresh data access to represent the data quality of the system. Therefore, a good scheduling scheme should update data in time and try to avoid data staleness.

It is difficult to achieve both high service ratio and good data quality. Giving more bandwidth to download requests can have a higher download service ratio, but a higher update drop ratio and hence low data quality. If update requests get more bandwidth, the service ratio decreases. There is always a tradeoff between high service ratio and good data quality. In the following sections, we first focus on improving the service ratio, and then design scheduling algorithms considering both service ratio and data quality.

[1] After a vehicle establishes the connectivity with one RSU, it can get the geographic information and radio range of the RSU through beacon messages. With its own driving velocity and position information, the vehicle can estimate its living time, which is its service deadline.

4 The basic scheduling schemes

The primary goal of a scheduling scheme is to serve as many requests as possible. We identify two parameters that can be used for scheduling vehicle-roadside data access:

- *DataSize*: If the vehicles can communicate with the RSU at the same data transmission rate, the data size can decide how long the service will last.
- *Deadline*: If a request can not be served before its deadline, it has to be dropped. Thus, the request with an earlier deadline is more urgent than the request with a later deadline.

4.1 Three naive schemes

There are three naive Schemes:

- First Come First Serve (FCFS): the request with the earliest arrival time will be served first.
- First Deadline First (FDF): the request with the most urgency will be served first.
- Smallest DataSize First (SDF): the data with the smallest size will be served first.

Figure 2 compares the service ratios under these three naive scheduling schemes. The experiment is conducted using the same simulation environment described in Section 7. The inter-arrival time of the

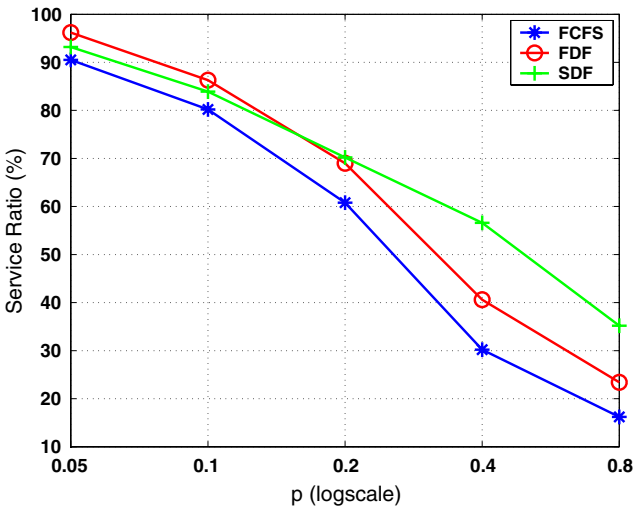


Fig. 2 Service ratio for FCFS, FDF and SDF scheme

requests is determined by the percentage of vehicles that will issue service requests, which is varied along the x axis.

As shown in the figure, when the request arrival rate is low, FDF outperforms FCFS and SDF. This is because when the workload is low, the deadline factor has more impact on the performance. After the urgent requests are served, other pending requests can still have the opportunity to get services. However, when the request arrival rate increases, the service ratio of FDF drops quickly while SDF performs relatively better. Since the system can always find short requests for service, SDF can still keep a higher service ratio. FCFS does not take any deadline or data size factors into account when making scheduling decision, it has the worst performance.

Clearly, FDF and SDF can only achieve good performance for certain workloads only. This motivates us to integrate the deadline and data size to improve the performance of scheduling.

4.2 The $\mathcal{D} * \mathcal{S}$ scheduling

FCFS does not consider data size and request deadline. FDF gives the highest priority to the most urgent requests while neglecting the service time spent on those data items. SDF takes the data size into account but ignores the request urgency. As a result, none of them can provide a good scheduling. Inspired by [18], we propose a new scheduling scheme, called $\mathcal{D} * \mathcal{S}$ to consider both data size and deadline when scheduling vehicle-roadside data access. Intuitively,

- Given two requests with the same deadline, the one asking for a small size data should be served first.
- Given two requests asking for data with same size, the one with earlier deadline should be served first.

Motivated by the above observations, each request is given a service value based on its deadline and data size, called DS_value, as its service priority weight.

$$DS\_\text{value} = (\text{Deadline} - \text{CurrentClock}) * \text{DataSize}$$

Here, product is used to connect the deadline and data size factors because these two factors have different measurement scales and/or units. With product, different metrologies will not bring any negative effect on the comparison of two DS_values.

At each scheduling time, the $\mathcal{D} * \mathcal{S}$ scheme always serves the requests with the minimum DS_value.

4.3 The implementation of the $\mathcal{D} * \mathcal{S}$ scheme

A straightforward implementation of the $\mathcal{D} * \mathcal{S}$ scheme is to compute the DS_values of all requests, and then select the smallest one at the decision tick. This implementation has a computation complexity of $\mathcal{O}(m)$, where m is the number of pending requests. Here, we propose a different data structure to reduce the computation complexity to $\mathcal{O}(\log m)$. In the following, we first give the detail of the implementation and then formally prove its correctness and efficiency.

4.3.1 The data structure and the pruning algorithm

The data structure uses two sorted lists to store the requests. One list called D-List (Deadline-list) is used to record the deadline (D value) of each request. The other list called S-List (dataSize-list) is used to record the size (S value) of the data item that is asked by the request. D-List is ordered by the increasing deadline of each request and S-List is sorted in ascending order of the data size. As Fig. 3 shows, the searching procedure starts by examining the entry at the top of D-List. With an index we can easily find the corresponding size entry in S-List and calculate its DS_value. The MinDS is set to the DS_value of the first request in D-List. At the same time, MinS can be calculated using D' which is the deadline value of the next entry in the D-List, and the current clock time. Since the D-List is ordered increasingly, it is known that for any unexamined entry to have a DS_value less than MinDS, it must have an S value satisfying the inequality

$$S < \frac{\text{MinDS}}{D' - \text{CurrentClock}}$$

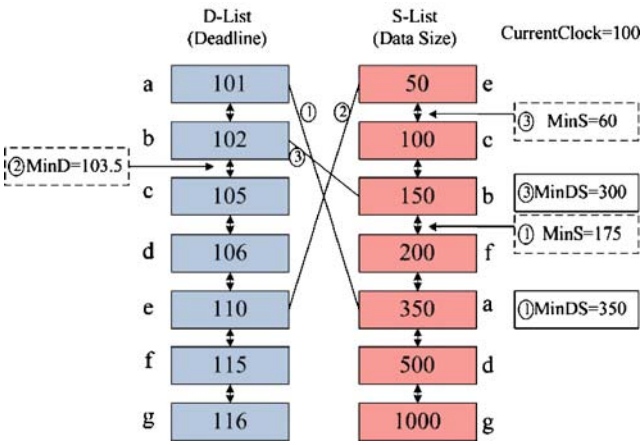


Fig. 3 Search space pruning structure

Then, we examine the entry at the top of S-List and calculate its DS_value with its S value and its corresponding D value in D-List. The MinDS value need to be updated with the current DS_value if the current DS_value is less than MinDS. Similarly, since the S-List is sorted in ascending order, an unexamined request has a DS_value less than MinDS only if its deadline value satisfies

$$D < \frac{MinDS}{S'} + CurrentClock$$

where S' is the data size value of the next entry in the S-List.

The search process keeps alternating between the D-List and S-List, updating the MinDS value when an entry with a DS_value less than MinDS is encountered and pruning the search space. The process stops when the checked entry goes across MinD or MinS, or when the search reaches halfway of both lists. At this point, MinDS is known to be the minimum DS_value for all requests and that recorded request can be served. Clearly, with this pruning technique, the search space can diminish quickly and the computation complexity can be reduced.

Figure 3 shows a simple example using these two lists. Suppose that the current clock is 100. First, the top entry (request *a*) in D-List is examined and the MinDS is set as 350 (= (101–100)*350). With this MinDS, we can calculate MinS=175 (=350/ (102–100)). Next, we check the entry of request *e* (the top of the S-List). The DS_value of request *e* is 500 (= (110–100)*50), which is larger than the current MinDS, so MinDS does not need to be updated. We can also get the value of MinD as 103.5 (=350/100+100). Then the second entry of D-List is checked. Its DS_value is 300 (= (102–100)*150). Since it is less than the current MinDS, MinDS should be updated to 300, and MinS is set to 60 (=300/ (105–100)). Next, we go to the second entry (request *c*) of S-List. Because its size is larger than the current MinS value, the search process can stop here since the unexamined requests (with size $\geq$ 100 and deadline $\geq$ 103.5) do not have a DS_value less than MinDS=300. In this example, we only need to check three index entries to find the most suitable request for service. The overhead to maintain this data structure is very low. Once an entry is added to the list, it is not moved until the corresponding request is served or dropped.

4.3.2 The analysis of the space pruning structure

First, we prove the correctness of the proposed space pruning algorithm. After that, we show that the computation complexity of the algorithm is $\mathcal{O}(\log m)$.

Theorem 1 *When the pruning process terminates, the MinDS is obtained.*

Proof (Proof by Contradiction.) Assume to the contrary that Theorem 1 is not correct. Then when the pruning process terminates, there must exist at least one request, whose DS_value is smaller than current MinDS and it has not been checked yet. Without loss of generality, we may assume that this request is request *i*. According to the algorithm, neither the deadline value nor the size value of request *i* has been checked. However, both the D-List and the S-List are ordered increasingly. This means both the deadline value and the data size value of request *i* are larger than those of the request who currently achieves the MinDS, respectively. Obviously, for all numbers which is larger than zero, the product of two larger numbers is always larger than that of two smaller numbers. This contradicts the assumption that request *i* has a smaller DS_value than MinDS while not been examined when the pruning process terminates. Hence, the supposition is false and Theorem 1 is correct. □

Theorem 2 *The computation complexity of the proposed pruning algorithm is* $\mathcal{O}(\log m)$*, where m is the number of pending requests.*

Proof Since both the deadline list and the data size list are ordered increasingly. Then as the first request in deadline list is examined, its data size value can be located at any position of the size list. Similar to the property of a binary-search-tree, after the first pruning, one half of entries in the data size list can be pruned on average. This process can be executed alternatively between two lists and recursively among all remaining entries in the lists until the MinDS value is obtained. Therefore, similar to the performance of search in a binary-search-tree, the computation complexity of the proposed pruning algorithm is $\mathcal{O}(\log m)$. □

5 Download optimization: broadcasting

5.1 The basic idea

The $\mathcal{D} * \mathcal{S}$ scheduling scheme considers both request deadline and data size, and it serves one request at one time. Observe that some vehicles may ask for the same data and the wireless communication has the broadcast capability. If we can delay some requested data and broadcast it before the deadlines, several requests may be served through a single broadcast. For example, several vehicles at an intersection want to check the

same traffic information. One broadcast can serve all these requests. With this optimization, the scheduling performance can be improved.

To improve the broadcast efficiency, the data with more pending requests should be served first. We add one more parameter to the $\mathcal{D}*\mathcal{S}$ scheme, i.e., the number of pending requests for the same data (N). We call the new scheme $\mathcal{D}*\mathcal{S}/\mathcal{N}$. In this new scheme, the service value, $DSN_$value, is calculated as

$$DSN\_\text{value} = (\text{Deadline} - \text{CurrentClock}) * \text{DataSize}/\text{Number}$$

5.2 Theoretical analysis of the $\mathcal{D}*\mathcal{S}/\mathcal{N}$ service ratio

In this subsection, we give a theoretical analysis of the service ratio in the $\mathcal{D}*\mathcal{S}/\mathcal{N}$ algorithm. It has been shown that "equal spacing" broadcast offers the optimal scheduling performance [21]. That is, the service ratio of a data item is maximized when the broadcast instances of the item are equally spaced. Therefore, we assume our scheduling satisfies the equal spacing property with a broadcast interval vector $< s_1, s_2, ..., s_n >$, where s_i is the spacing interval between consecutive transmissions of item i. As Fig. 4 shows, a request arriving at time t can be served successfully if and only if its deadline is shorter than $s_i - t$. Moreover, if the arrival rate of requests becomes large, e.g., approaching infinity, the number of requests for item i arriving in any very short duration δt can be approximated by $\lambda_i * \delta t$, where λ_i is the arriving rate for item i. Therefore, if the deadlines of the requests asking for data item i satisfy the exponential distribution with mean μ, the service ratio of requests for data item i can be calculated by

$$\frac{\int_0^{s_i} \lambda_i e^{-\frac{1}{\mu}(s_i - t)} dt}{\int_0^{s_i} \lambda_i dt} = \frac{\mu\lambda_i \left(1 - e^{-\frac{s_i}{\mu}}\right)}{\lambda_i s_i} = \frac{\mu \left(1 - e^{-\frac{s_i}{\mu}}\right)}{s_i}$$

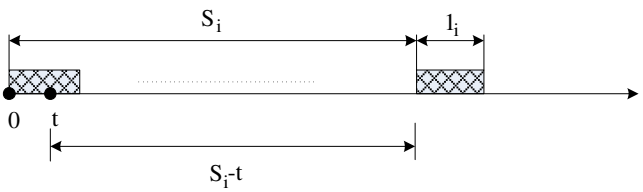


Fig. 4 Equal spacing broadcast model

Note that the above expression is λ_i independent. Therefore, the total service ratio (SR) can be calculated as

$$SR = \sum_{i=1}^{N} \frac{\mu p_i \left(1 - e^{-\frac{s_i}{\mu}}\right)}{s_i}$$

where p_i is the access probability of data item i

We use l_i to denote the time required to serve data item i, then an inter-broadcast interval of s_i implied a fraction $\frac{l_i}{s_i}$ of the bandwidth is used to broadcast item i. Therefore,

$$\sum_{i}^{N} \frac{l_i}{s_i} = 1$$

With this limitation, we can use Lagrange multiplier method to calculate the maximum value of SR, if the service scheduling satisfies

$$\frac{p_i}{l_i}\left(\frac{s_i}{\mu} e^{-\frac{s_i}{\mu}} + e^{-\frac{s_i}{\mu}} - 1\right) = K, i = 1, 2, ...N$$

for some constant K.

At this time, the maximum value of SR is the upper bound of the service ratio of the $\mathcal{D}*\mathcal{S}/\mathcal{N}$ scheme.

5.3 The implementation of $\mathcal{D}*\mathcal{S}/\mathcal{N}$

The $\mathcal{D}*\mathcal{S}/\mathcal{N}$ scheme can be implemented using a similar dual-list data structure as in $\mathcal{D}*\mathcal{S}$ to reduce the computation complexity. The difference is that each entry records the information for each request group for the same data item instead of individual requests in $\mathcal{D}*\mathcal{S}$. Because there are three parameters (deadline, data size and pending requests (N)) in consideration, we need to combine the S value and N value in advance to form a single S/N list. Since the S/N value of the corresponding data item can be updated when a new request comes, this change does not bring much maintenance overhead. At each scheduling decision tick, the same pruning process is executed alternatively between D-List and S/N-List until the request with minimal $DSN_$value is found.

5.4 The selection of representative deadline

With the broadcast optimization, several requests can be served simultaneously in a single broadcast, which leads to more efficient use of shared bandwidth and a higher service ratio. However, when calculating their $DSN_$value, we need to assign each pending request group a single deadline to estimate the urgency of the

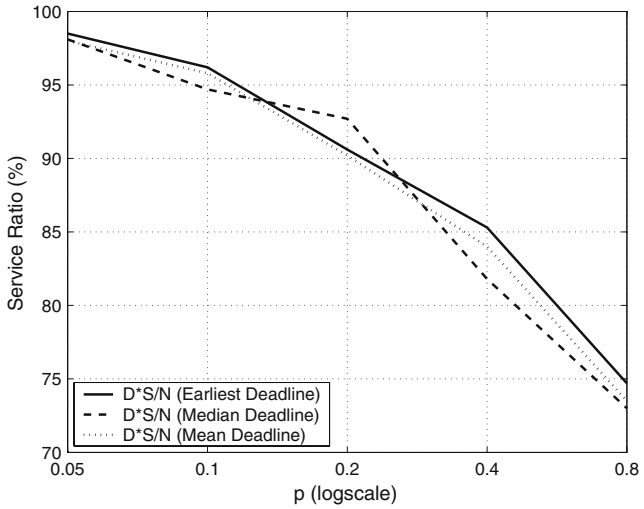


Fig. 5 The service ratio of $\mathcal{D} * \mathcal{S}/\mathcal{N}$ with earliest, median, and mean deadline

whole group. If there are more than one request waiting for the service, we can use the earliest, the median or mean deadline of the group to represent the group urgency. The earliest deadline reflects the urgency of satisfying all requests in the group and the mean and/or median deadline reflects the average urgency of the requests group. With the same setting in Section 7, we compare the performance under different representative deadlines. The simulation results (Fig. 5) show that selecting different representative deadline does not have too much impact on the scheduling performance. The earliest, median, and mean deadlines lead to similar scheduling performance. Therefore, in our later simulation settings, we choose the earliest deadline to represent the urgency of all pending requests in the same group.

6 Upload optimization: two-step scheduling

$\mathcal{D} * \mathcal{S}/\mathcal{N}$ can improve the service ratio, but it sacrifices the service opportunity of the upload (update) requests. For upload request, it is not necessary to maintain several update requests for one data item since only the last update is useful. As a result, the N value of upload request is always 1 in the $\mathcal{D} * \mathcal{S}/\mathcal{N}$ scheme, and hence it is not fair for update requests to compete for the service bandwidth. Clearly, more update requests have to be dropped and the data quality for downloading degrades as lots of later arriving download requests will get the stale data. A possible improvement is to incorporate a weight value to update requests to help them get a higher priority in scheduling. However, it is quite difficult to say how much weight should be given to the update requests since in a dynamic system the degradation of service quality by staleness and service ratio are incomparable. Therefore, it is impossible to use one single queue for both update and download [20]. Next, we propose a new scheduling scheme with two separate queues and a Two-Step scheduling approach to achieve the balance of data quality and service ratio.

6.1 The basic idea

We use two priority queues: one for the update requests and the other for the download requests. The RSU server provides two queues with different bandwidth (i.e., service probability). The benefit of using two separate priority queues is that we only need to compare the download queue and update queue instead of individual update and download requests. The scheduling goes through two steps: the first step chooses the service queue and the second step chooses the most suitable service request. Because of their specific concerns, update and download queues have their own priority scheduling schemes, which makes the scheduling more flexible.

6.2 Step I: update queue or download queue

Here we give a new definition, Service_Profit, as the sum of the profit gained from update and download requests. Suppose the download requests share ρ ($0 \leq \rho \leq 1$) of the bandwidth and the update requests share the rest: 1-ρ. We need to set ρ to the best value to achieve the maximum Service_Profit. To do that, we need to find the relationship between bandwidth allocation and the Service_Profit that the system can have.

The download requests can be served with fresh data or stale data, and hence the profits they bring to the system are different. Formally, Service_Profit can be presented as

$$\begin{aligned}\text{Service\_Profit} = {} & \text{Update\_Profit} \\ & + \text{FreshDownload\_Profit} \\ & + \text{StaleDownload\_Profit}\end{aligned}$$

We use the number of successfully served requests to represent the system profit. Then, one update request contributes the same profit as one download request with fresh data. However, if one update request is dropped, all the following download requests on that data item can only get the stale data. Therefore, their service qualities degrade with a coefficient, say α. The service degrades until the data item is updated by the next update.

We use r_u to denote the service rate of update requests. Then the update profit rate depends on its service rate (r_u) and the bandwidth allocated to it ($1-\rho$). After a time period t, the update profit can be approximated as:

$$\text{Update\_Profit} \simeq r_u \cdot (1-\rho) \cdot t$$

Similarly, we use r_d to denote the service rate of the download requests. The download profit relies on its service rate (r_d), the bandwidth allocation, and the quality for each download. Note that the data quality is related to uploads. The more bandwidth allocated to the update queue, the more requests will be served with fresh data. Therefore, the download profit can be approximated by:

$$\text{FreshDownload\_Profit} \simeq r_d \cdot \rho \cdot (1-\rho) \cdot t$$

and

$$\text{StaleDownload\_Profit} \simeq r_d \cdot \rho^2 \cdot \alpha \cdot t$$

Then the total profit can be given by

$$\begin{aligned}\text{Service\_Profit} &\simeq r_u \cdot (1-\rho) \cdot t + r_d \cdot \rho \cdot (1-\rho) \cdot t \\ &\quad + r_d \cdot \rho^2 \cdot \alpha \cdot t \\ &= r_d \cdot (\alpha - 1) \cdot t\rho^2 + (r_d - r_u) \cdot t \cdot \rho \\ &\quad + f_u \cdot t \qquad (0 \le \rho \le 1).\end{aligned}$$

We can calculate the optimal ρ to maximize the Service_Profit by solving the quadratic function with the lineal constraint on ρ. The optimal solution is:

$$\rho = \begin{cases} \min\left(\frac{r_d - r_u}{2(1-\alpha)r_d}, 1\right) & (0 \le \alpha < 1) \wedge (r_u < r_d) \\ 0 & r_u \ge r_d \\ 1 & (a = 1) \wedge (r_u < r_d) \end{cases}$$

When $\alpha = 1$, which means that the stale data does not have any negative impact on the service quality, the bandwidth allocation totally depends on the service rate of update and download requests. Either update or download request that has a higher service rate can take the whole bandwidth.

Since the value of ρ depends on the service rate of update and download requests, which are related to the workload, it should be able to adjust adaptively with the workload. When an accident happens, there will be more update requests. Also, the request workload at daytime and night should be different. The workload is examined with a time period τ, which is referred to as the *adaptation window*. At the beginning of each τ, ρ is re-calculated. To learn about the gradual change in workload over a period of time by utilizing some history information, we record the information of two time windows and get:

$$\rho_{\text{new}} = \begin{cases} \min\left(\frac{r_{d,k-1} - r_{u,k-1}}{2(1-\alpha)r_{d,k-1}}, 1\right) & (0 \le \alpha < 1) \wedge (r_{u,k-1} < r_{d,k-1}) \\ 0 & r_{u,k-1} \ge r_{d,k-1} \\ 1 & (a = 1) \wedge (r_{u,k-1} < r_{d,k-1}) \end{cases}$$

$$\rho_k = (1-\beta) * \rho_{k-1} + \beta * \rho_{\text{new}}$$

where $r_{d,k-1}$ and $r_{u,k-1}$ are the service rates of the previous time period and β is a parameter to measure the importance of the most recent value in comparison with the old value.

6.3 Step II: $\mathcal{D} * \mathcal{S}/\mathcal{N}$ and $\mathcal{D} * \mathcal{S}/\mathcal{R}$

As discussed in Section 5, entries in the download queue can be sorted based on their priority values calculated by the $\mathcal{D} * \mathcal{S}/\mathcal{N}$ scheme. In the update queue, we calculate the priority values with another scheme, called $\mathcal{D} * \mathcal{S}/\mathcal{R}$, where R is the service rate of the download requests in the download queue of one data item. The basic idea of $\mathcal{D} * \mathcal{S}/\mathcal{R}$ is that we use the service rate of download requests in the download queue to optimize the scheduling in the update queue. For example, given two update requests with the same DS_value, the request that updates hot data should have a higher service priority since when the data item is updated, more download requests can get the fresh data thus improving the system profit. Therefore, we add the service rate, denoted by R, as a weight factor to the priority calculation, that is:

$$DSR\text{\_value} = (\text{Deadline} - \text{CurrentClock}) * \text{DataSize}/R$$

The pruning process in $\mathcal{D} * \mathcal{S}/\mathcal{R}$ can also be optimized by using the dual-list data structure, similar to that in $\mathcal{D} * \mathcal{S}/\mathcal{N}$. The S/R-List is updated at the beginning of each adaptation time window and the R value of each entry can be updated similar to ρ, that is

$$R_k = (1-\beta) * R_{k-1} + \beta * R_{\text{new}}$$

where R_{new} is the number of served download requests of one data item in the last adaptation window.

6.4 The implementation of the two-step scheduling

According to previous discussions, the scheduling is based on two separate priority queues. The bandwidth

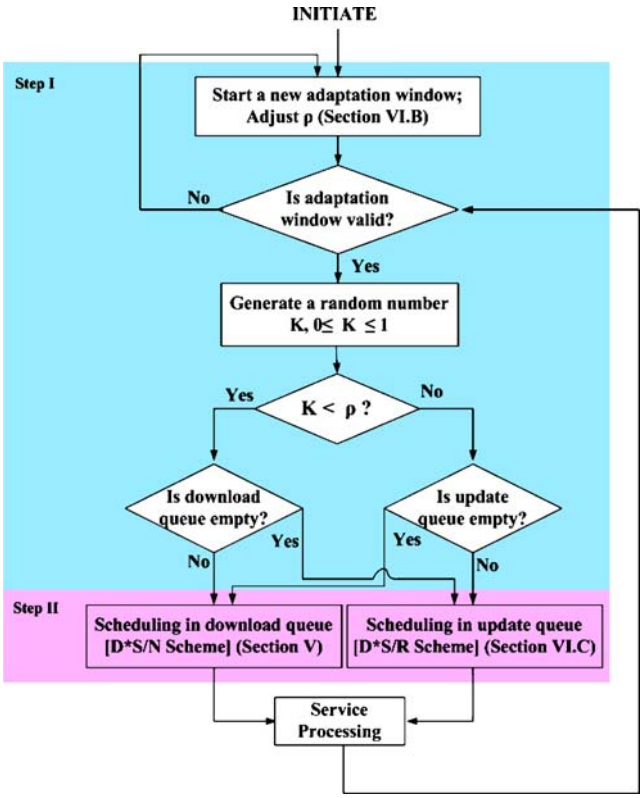


Fig. 6 Flow chart of the two-step scheduling

allocation for the update queue and the download queue depends on the periodically evaluated ρ. At each decision tick, the scheduler decides what queue (update or download) can get the service. After that, it serves a request in the specific queue. If the update queue is chosen, the $\mathcal{D} * \mathcal{S}/\mathcal{R}$ scheme is used to pick the next request to serve. If the bandwidth is assigned to the download queue, the $\mathcal{D} * \mathcal{S}/\mathcal{N}$ scheme is used. If one queue is empty, its service opportunity will be given to the other queue immediately. Figure 6 describes the flow chart of the Two-Step scheduling.

7 Performance evaluation

7.1 Experimental setup

We developed an ns-2 [25] based simulator to evaluate the proposed scheduling schemes. The experiment is based on a 400 m*400 m square street scenario. One RSU server is put at the center of the area. It is also the intersection of one horizontal road and one vertical road, where each two-way road has four lanes (Fig. 7). To simulate the vehicle traffic, we randomly deploy 40

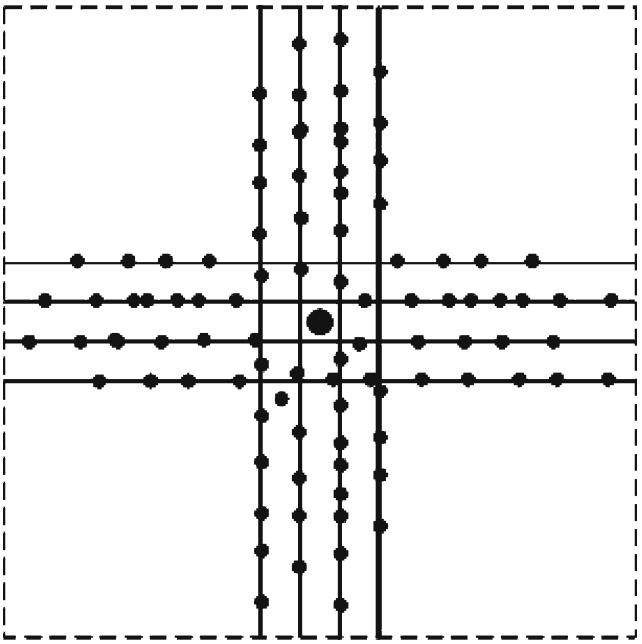


Fig. 7 The simulation scenario layout

vehicles in each lane initially, i.e., a total of 160 vehicles. All vehicles move towards either end of the road. They are moving forth and back during the simulation to mimic the continuous traffic flow in the intersection area. When one vehicle reaches the end of the road, which means the vehicle will move out of the RSU area, its request not serviced will be dropped. Each vehicle issues service requests with a probability p, $(0 < p \leq 1)$. A larger p is used to simulate a heavy service workload and a smaller p is for low workload. When one vehicle is served or reaching the end of the road, it waits some time to issue a new request. The inter-arrival time of each request follows an exponential distribution with a mean of λ. Its density function is:

$$f(t) = \lambda e^{-\lambda t}$$

Similar to [26], the access pattern of each data item follows $Zipf$ distribution. In the $Zipf$ distribution, the access probability of the i^{th} data item is represented as follows:

$$P_i = \frac{1}{i^{\theta} \sum_{j=1}^{n} \frac{1}{j^{\theta}}}$$

where $0 \leq \theta \leq 1$, n is the database size. When $\theta = 1$, it is the strict $Zipf$ distribution. When θ=0, it becomes the uniform distribution. The data item size randomly distributes between $s_{\min}$ and $s_{\max}$. Most of the system parameters and their default values are listed in Table 1.

Table 1 Simulation Setup

Parameter	Value
Simulation time	900 s
Transmission rate	5 Mbit/s = 625 Kbyte/s[1]
Vehicle velocity	15 m/s
Wireless coverage	200 m
Data size	50 K ∼ 5 M, average 2.5 M
Vehicle-vehicle space	20 m
Data set size	25
$Zipf$ parameter θ	0.8
Update percentage	10%
Adaptation window	40 s

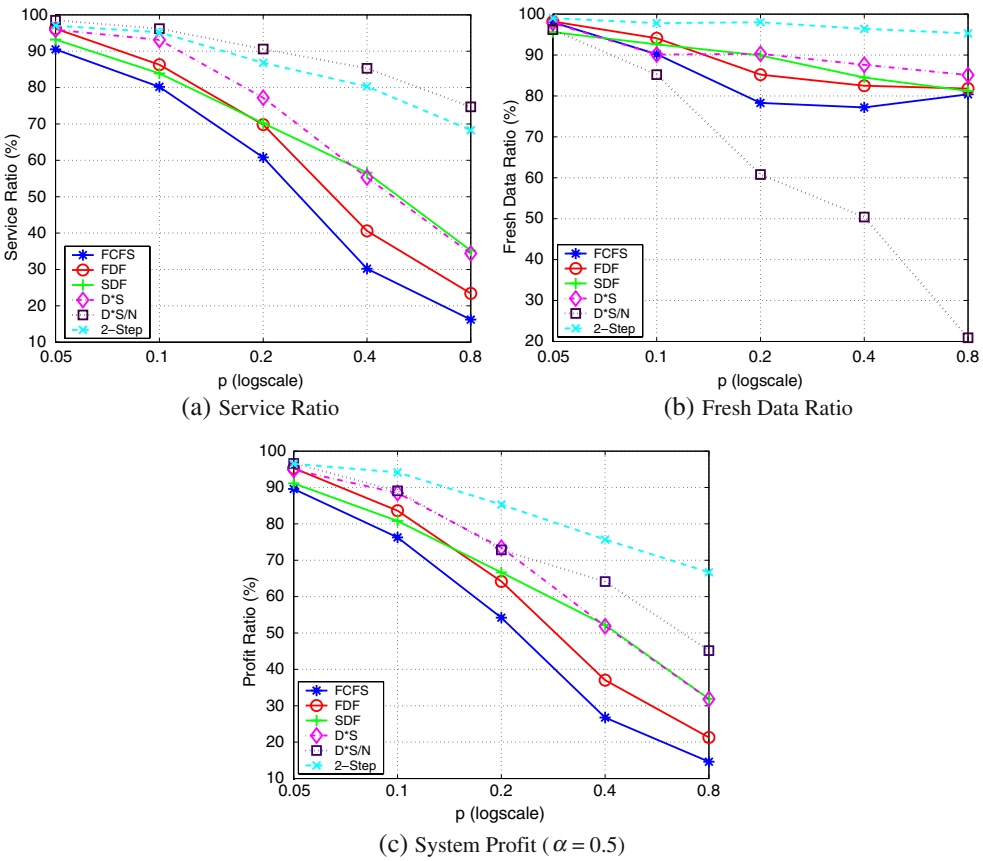
7.2 The effect of workload

Figure 8 shows the effect of the request arrival rate to the scheduling performance for the six schemes discussed in this paper. As shown in Fig. 8a, more requests have to be dropped as the request arrival rate increases. Since FCFS, FDF, SDF and $\mathcal{D} * \mathcal{S}$ serve each request individually, their service ratios decrease very quickly with the increasing of workload. $\mathcal{D} * \mathcal{S}/\mathcal{N}$ and the Two-Step scheme use broadcast to optimize the service. With this technique, they can achieve much higher service ratio than the other four schemes because several download requests for the same data item can be served simultaneously using a single broadcast. Since $\mathcal{D} * \mathcal{S}/\mathcal{N}$ scheduling is not fair to update requests, the data quality cannot be guaranteed. As Fig. 8b shows, when 80% of vehicles issue requests for service (heavy workload), only about 20% of all served requests can get the up-to-date data, which is much lower than other schemes. Figure 8c compares the total Service_Profit ratios of different schemes. Here, we set $\alpha = 0.5$, which means if one download request is served with stale data, it only contributes a 50% profit compared with other requests that are served with fresh data. As can be seen, the Two-Step scheme has a much higher service profit ratio than other schemes.

7.3 The effect of access pattern

7.3.1 θ

Figure 9 shows the performance as a function of the access skew parameter θ. In $Zipf$ distribution, when $\theta = 0$, the access pattern is uniformly distributed, and different data items have similar popularity. As θ increases, the access pattern becomes more skewed. Since FCFS, FDF, SDF and $\mathcal{D} * \mathcal{S}$ make the scheduling



(a) Service Ratio (b) Fresh Data Ratio

(c) System Profit ($\alpha = 0.5$)

Fig. 8 The effect of workload (**a–c**)

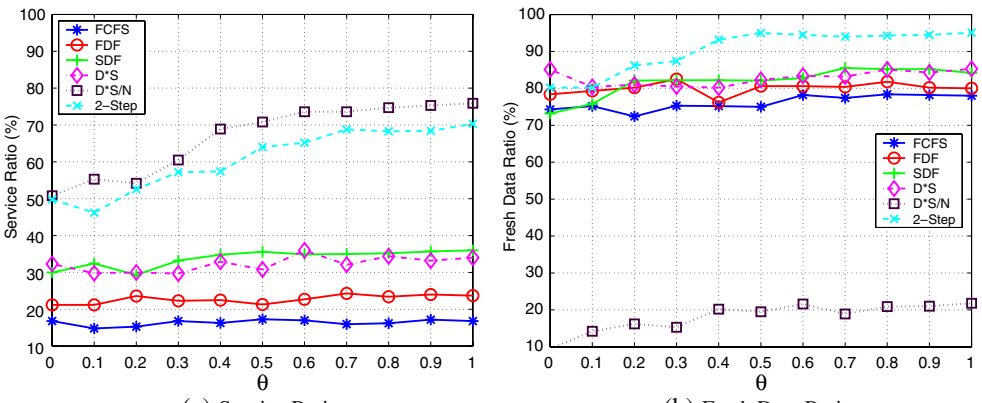
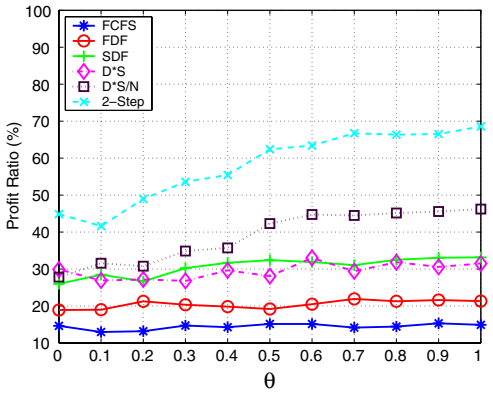


Fig. 9 The effect of $Zipf$ distribution parameter θ (**a**–**c**)

(a) Service Ratio (b) Fresh Data Ratio



(c) System Profit ($\alpha = 0.5$)

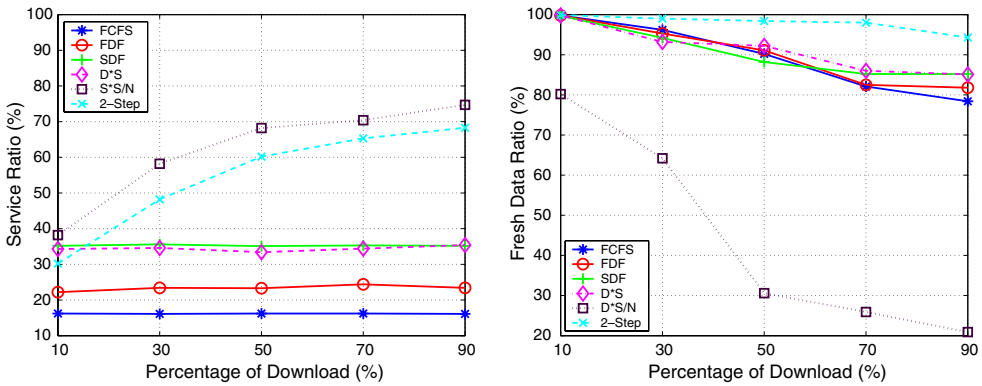
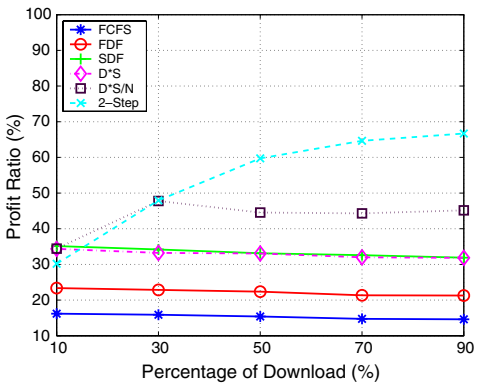
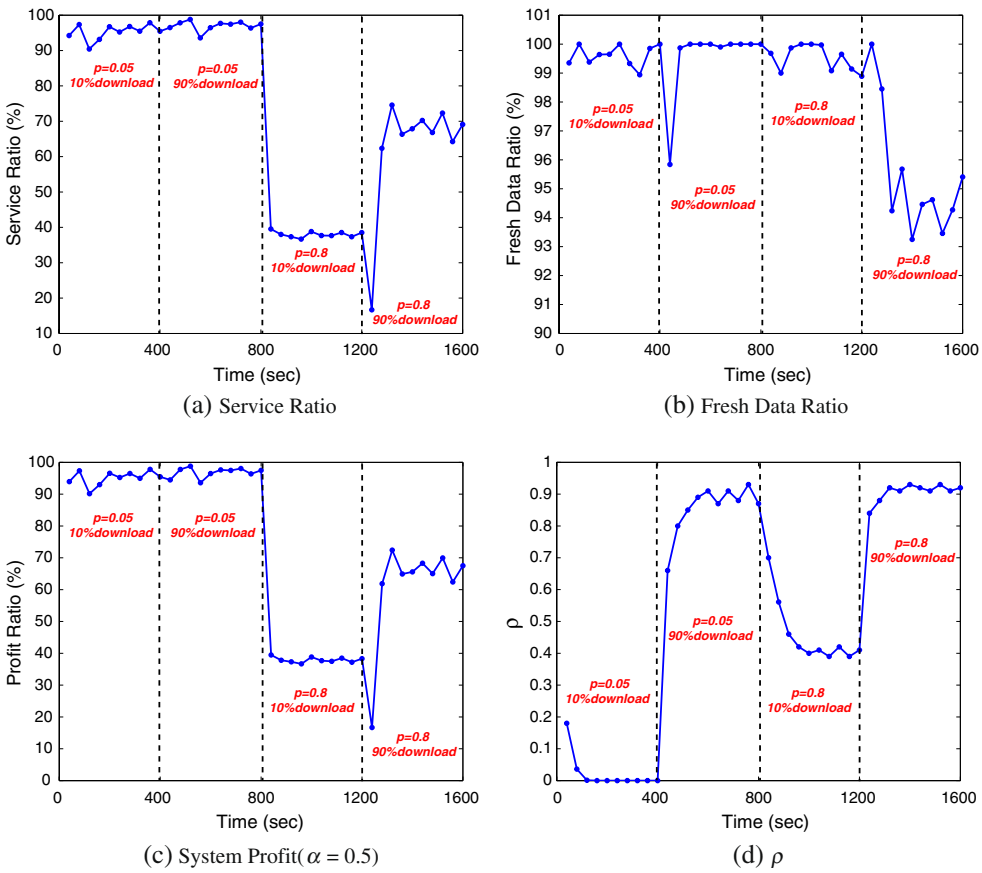


Fig. 10 The effect of download/update ratio (**a**–**c**)

(a) Service Ratio (b) Fresh Data Ratio



(c) System Profit ($\alpha = 0.5$)



(a) Service Ratio (b) Fresh Data Ratio

(c) System Profit($\alpha = 0.5$) (d) ρ

Fig. 11 The study of the adaptivity to traffic condition (**a**–**d**)

decision based on individual request, the change of θ does not have too much impact on their performance. For $\mathcal{D} * \mathcal{S}/\mathcal{N}$ and the Two-Step scheme that use broadcast optimization, they can benefit from the skewness of the data access pattern with the increase of θ. When the data access pattern becomes asymmetrical, popularity becomes the major performance factor. The popular data can have a high service priority weight in $\mathcal{D} * \mathcal{S}/\mathcal{N}$ and Two-Step scheme, which helps improve the service ratio.

7.3.2 Download/update ratio

Figure 10 compares the performance of different schemes when the download/upload ratio changes. When more download requests come, the service ratio of $\mathcal{D} * \mathcal{S}/\mathcal{N}$ scheme and Two-Step scheme increases quickly. This performance improvement comes from the benefit of download broadcasting. Because $\mathcal{D} * \mathcal{S}/\mathcal{N}$ prefers service ratio rather than data quality, as download rate increases, its fresh data ratio decreases. This is consistent with our previous discussion. The Two-Step scheme can achieve relatively good performance on both service ratio and data quality. Therefore, it has the highest profit ratio.

7.4 Adaptivity to traffic condition

To study the adaptivity of the Two-Step scheme. We divide the experiment period evenly into four intervals. The first interval and the second interval have low workloads ($p = 0.05$) while the third and fourth have heavy workloads ($p = 0.8$). At the same time, the download ratios of the first and third interval are low (10% download) while the second and fourth intervals have high download ratios (90% download). We create a sudden change at the start of each interval. This experiment is to show how the Two-Step scheme can react to the changes to different scenarios and adjust ρ accordingly.

As expected, the Two-Step scheme can achieve good performance in almost all scenarios. Here, note that in the third time interval with high workload and low download ratio, the service ratio is relatively low compared with that in other intervals (Fig. 11a). This is because in this service scenario, most arrivals are update requests. They need to be serviced one by one and hence the advantage of broadcasting cannot be well utilized. Also, although there may be sudden performance drops at the start of a switch interval, the Two-Step scheme adjusts ρ quickly and keeps a

high performance. Figure 11d shows the value of ρ over time. ρ is the probability that download requests have higher priority than upload requests. As shown in Fig. 11d, ρ is small when the download rate is low, and ρ adapts quickly when the download rate becomes high.

8 Conclusions

In the paper, we addressed some challenges in vehicle-roadside data access. We proposed a basic scheduling scheme called $\mathcal{D} * \mathcal{S}$ to consider both service deadline and data size when making scheduling decisions. An efficient search space pruning technique is presented to reduce the computation complexity for making scheduling decisions. To make use of the wireless broadcasting, we proposed a new scheduling scheme called $\mathcal{D} * \mathcal{S}/\mathcal{N}$ to serve multiple requests with a single broadcast. We also identified the effects of upload requests on data quality, and proposed a Two-Step scheduling scheme to provide a balance between serving download and update requests. Simulation results show that the Two-Step scheduling scheme outperforms other scheduling schemes. Further, the Two-Step scheduling scheme is adaptive to different workload scenarios.

This paper focuses on service scheduling issues in vehicle-roadside data access. In this paper, we assume the all vehicles have the same transmission rates with RSU as they are connected. However, [1] has shown that the transmission rate may vary as the distance between vehicle and RSU changes. Meanwhile, the number of successfully served requests, i.e. uploading and fresh downloading, is used to represent the scheduling performance, which may bring in some fairness issues. As future work, we will take multi-rate and fairness issues into consideration on scheduling. Further, other unique challenges in vehicular networks such as group mobility and platoon effect [27] will motivate further research in this area.

Acknowledgement This work was supported in part by the US National Science Foundation under grant CNS-0721479.

References

1. Ott J, Kutscher D (2004) Drive-thru internet: Ieee 802.11b for automobile users. In: Proceedings of INFOCOM 04'
2. Bychkovsky V, Hull B, et al (2006) A measurement study of vehicular internet access using in situ wi-fi networks. In: Proceedings of MobiCom'06, pp 50–61
3. Hull B, Bychkovsky V, Zhang Y, et al (2006) Cartel: a distributed mobile sensor computing system. In: Proceedings of SenSys'06, pp 125–138
4. Hadaller D, Keshav S, Brecht T, Agarwal S (2007) Vehicular opportunistic communication under the microscope. In: Proceedings of MobiSys'07, pp 206–219
5. Zhang Y, Zhao J, Cao G (2007) On scheduling vehicle-roadside data access. In: Proceedings of VANET'07, pp 9–18
6. Department of Transportation (2009) Intelligent transportation systems. http://www.its.dot.gov/vii/
7. Zhao J, Cao G (2006) VADD: vehicle-assisted data delivery in vehicular ad hoc networks. In: Proceedings of IEEE INFOCOM'06, pp 1–12
8. Zhang Y, Zhao J, Cao G (2009) Roadcast: a popularity aware content sharing scheme in vanets. In: Proceedings of ICDCS'09, Montreal
9. Lee U, Magistretti E, Gerla M, Bellavista P, Corradi A (2009) Dissemination and harvesting of urban data using vehicular sensing platforms. IEEE Trans Mob Comput 58(2):882–901
10. Zhao J, Zhang Y, Cao G (2007) Data pouring and buffering on the road: a new data dissemination paradigm for vehicular ad hoc networks. IEEE Trans Veh Technol 56(6):3266–3277
11. Balasubramanian A, Levine BN, Venkataramani A (2008) Enhancing interactive web applications in hybrid networks. In: Proceedings of MobiCom'08, pp 70–80
12. Enkelmann W (2003) Fleetnet-applications for intervehicle communication. In: Proceedings of IEEE IV, pp 162–167
13. Ernst T, Uehara K, Mitsuya K (2003) Network mobility from the internetcar perspective. In: Proceedings of the 17th international conference on advanced information networking and applications, Washington, DC
14. Wong J (1988) Broadcast delivery. In: Proceeding of the IEEE, pp 1566–1577
15. Su C, Tassiulas L (1997) Broadcast scheduling for information distribution. In: Proceeding of INFOCOM 97'
16. Vaidya N, Hameed S (1999) Scheduling data broadcast in asymemetric communication environments. Wirel Netw 5:183–193
17. Gandhi R, Khuller S, Kim Y, Wan Y (2004) Algorithms for minimizing response time in broadcast scheduling. Algorithmica 38(4):597–608
18. Aksoy D, Franklin M (1999) R*w: a scheduling approach for large-scale on-demand data broadcast. IEEE/ACM Trans Netw 7:846–860
19. Acharya S, Muthukrishnan S (1998) Scheduling on-demand broadcasts: new metrics and algorithms. In: Proceeding of MobiCom 98'
20. Qu H, Labrinidis A (2007) Preference-aware query and update scheduling in web-databases. In: Proceedings of ICDE'07, pp 356–365
21. Jiang S, Vaidya N (1999) Scheduling data broadcast to "impatient" users. In: Proceedings of MobiDE'99, pp 52–59
22. Rajan D, Sabharwal A, Aazhang B (2004) Power efficient broadcast scheduling with delay deadlines. In: Proceedings of the first international conference on broadband networks (BROADNETS'04), Washington, DC, pp 439–448
23. Xu J, Tang X, Lee W (2006) Time-critical on-demand data broadcast: algorithms, analysis, and performance evaluation. IEEE Trans Parallel Distrib Syst 17:3–14
24. Wu Y, Cao G (2001) Stretch-optimal scheduling for on-demand data broadcasts. In: Proceeding of the 10th international conference on computer communications and networks, pp 500–504
25. The Network Simulator (2009) The Network Simulator homepage. http://www.isi.edu/nsnam/ns
26. Yin L, Cao G (2006) Supporting cooperative caching in ad hoc networks. IEEE Trans Mob Comput 5(1):77–89
27. Gerlough D, Huber M (1975) Traffic flow theory—a monograph. Special Report 165, Transporation Reseaerch Board