Scheduling Temporal Data with Dynamic Snapshot Consistency Requirement in Vehicular Cyber-Physical Systems

Liu, Kai; Lee, Victor C. S.; Ng, Joseph K. Y.; Son, Sang H.; Sha, Edwin H.-M.

Published in:
Transactions on Embedded Computing Systems

DOI:
10.1145/2629546

Published: 06/10/2014

Document Version:
Version created as part of publication process; publisher's layout; not normally made publicly available

Scheduling Temporal Data with Dynamic Snapshot Consistency Requirement in Vehicular Cyber-Physical Systems

Kai Liu, Chongqing University
Victor C.S. Lee, City University of Hong Kong
Joseph K.Y. Ng, Hong Kong Baptist University
Sang H. Son, DGIST, Daegu, Korea
Edwin H.-M. Sha, Chongqing University

Timely and efficient data dissemination is one of the fundamental requirements to enable innovative applications in vehicular cyber-physical systems (VCPS). In this work, we intensively analyze the characteristics of temporal data dissemination in VCPS. On this basis, we formulate the static and the dynamic snapshot consistency requirements on serving real-time requests for temporal data items. Two on-line algorithms are proposed to enhance the system performance with different requirements. In particular, a reschedule mechanism is developed to make the scheduling be adaptable to the dynamic snapshot consistency requirement. A comprehensive performance evaluation demonstrates the superiority of the proposed algorithms.

Categories and Subject Descriptors: H.3.5 [**Online Information Services**]

General Terms: Algorithms, Design, Performance

Additional Key Words and Phrases: Real-time scheduling, temporal data dissemination, vehicular cyber-physical systems

ACM Reference Format:
Kai Liu, Victor C.S. Lee, Joseph K.Y. Ng and Sang H. Son, 2013. Scheduling temporal data with dynamic snapshot consistency requirements in vehicular cyber-physical systems. *ACM* V, N, Article A (January YYYY), 20 pages.
DOI: http://dx.doi.org/10.1145/0000000.0000000

1. INTRODUCTION

The vehicular cyber-physical system (VCPS) is a promising approach to achieving breakthroughs in transportation safety, efficiency and sustainability [Miloslavov and Veeraraghavan 2012]. With the latest advances in communications, computing, electronics, sensing, and control, etc., a variety of innovative applications are envisioned in the near future, such as collision avoidance systems [Gehrig and Stein 2007], roadway reservation systems [Liu et al. 2013], and autonomous intersection management systems [Lee and Park 2012], to name but a few. Obviously, efficient data dissemination in

This work was supported in part by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China [Project No. CityU 115312] and a grant from the HKBU Research Centre for Ubiquitous Computing (RCUC), the Institute of Computational and Theoretical Studies (ICTS) and the HKBU Strategic Development Fund [Grant No. HKBU SDF 10-0526-P08]. It was partially supported by the Ministry of Science, ICT and Future Planning (Cyber-Physical Systems Global Center) and the National Research Foundation of Korea Global Research Laboratory Program under Grant 2013K1A1A2A02078326. It was partially supported by National 863 Program 2013AA013202, Chongqing cstc2012ggC40005, NSFC 61173014, NSF CNS-1015802.
Author's addresses: Kai Liu, College of Computer Science, Chongqing University, Chongqing, China; email: liukai0807@gmail.com; Joseph K.Y. Ng, Department of Computer Science, Hong Kong Baptist University, Kowloon, Hong Kong; email:jng@comp.hkbu.edu.hk; Victor C.S. Lee, Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong; email: csvlee@cityu.edu.hk; Sang H. Son, Department of Information and Communication Engineering, Daegu Gyeongbuk Institute of Science and Technology, Korea; email: son@dgist.ac.kr; Edwin H.-M. Sha, College of Computer Science, Chongqing University, Chongqing, China; email: edwinsha@gmail.com.

DOI: http://dx.doi.org/10.1145/0000000.0000000

terms of providing timely and fresh information services is one of the fundamental requirements to enable these applications.

The dedicated short range communication (DSRC) [FCC 2006] is a key technology to support vehicular communications. The term DSRC commonly refers to a suite of standards, including IEEE 802.11p, IEEE 1609.1/.2/.3/.4 protocol family and SAE J2735 message set dictionary, etc. [Morgan 2010]. The U.S. Federal Communication Commission (FCC) allocates 75MHz of spectrum in the 5.9GHz band for DSRC to be used exclusively for infrastructure-to-vehicle (I2V) and vehicle-to-vehicle (V2V) communications. The road-side unit (RSU) is a fixed infrastructure installed along the road to provide information services to passing vehicles, while the on-board unit (OBU) is mounted on vehicles for both I2V and V2V communications. Different parties, including automotive OEMs, governments, research universities, etc., are actively engaged into the development and the deployment of VCPSs based on vehicular communications, such as the *Connect Vehicle* research program [Con 2013], the *Vehicle Infrastructure Integration* project [VII 2013], the MIT*CarTel* project [Car 2013], and the Berkeley *PATH* project [ITS-Berkeley 2013], etc. With the well support of cutting-edge communication techniques and the great need of emerging transportation applications, it is imperative to develop an efficient data dissemination system for providing real-time services in vehicular networks.

Although great efforts have been devoted to investigating data dissemination in conventional mobile computing systems [Hu and Chen 2009; Aksoy and Franklin 1999; Xu et al. 2006; Chen et al. 2013], they are not sufficient when considering the unique requirements and challenges of data scheduling arising in VCPSs. First, the time-constraint on information retrieval in vehicular networks is more stringent than many mobile applications due to the high mobility of vehicles. Second, the traffic information such as road conditions is highly dynamic, and it is challenging to guarantee the delivery of up-to-date information. Third, there are many applications where multiple dependent data items are required for query processing, and it is critical to ensure the consistency of these dependent data items retrieved by vehicles. We take the navigation system as an example to elaborate the above requirements. When a driver inquires the best route to a destination, the navigation system needs to obtain the traffic conditions of different roads, which correspond to different temporal data items. The best route can be computed only when all these data items are retrieved. In addition, the most up-to-date values of different pieces of information are expected to make the query result meaningful. Finally, the query result has to be returned in time (e.g. before the vehicle passing the intersection). Otherwise, it would be too late for the driver to make decisions. With the above analysis, it is critical yet challenging to schedule real-time requests for multiple dependent temporal data items in VCPS, and existing scheduling algorithms are not applicable in such an environment.

The main contributions of this work are outlined as follows. Firstly, we present the system architecture and analyze the characteristics of data dissemination in vehicular environments. Secondly, we formulate the problem by analyzing two types of requirements on serving real-time requests for temporal data items, namely, the static and the dynamic snapshot consistency requirements. Thirdly, we propose two on-line algorithms, SSCS (Static Snapshot Consistency oriented Scheduling) and DSCS (Dynamic Snapshot Consistency oriented Scheduling), which are dedicated to scheduling under different consistency requirements. In particular, a reschedule mechanism is developed for DSCS to make it be adaptable to the dynamic snapshot consistency requirement. Finally, we build the simulation model and evaluate the algorithm performance under a variety of circumstances. The simulation results demonstrate the superiority of the proposed algorithms.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the system architecture and analyzes the characteristics of data dissemination. Section 4 formulates the problem on serving real-time requests for temporal data items. Two scheduling algorithms, SSCS and DSCS, are proposed in Section 5. In Section 6, we build the simulation model and give a comprehensive performance evaluation. Last, we conclude this work and discuss future research directions in Section 7.

2. RELATED WORK

Current research on data dissemination in vehicular networks largely focuses on improving the communication quality at the MAC layer [Fujimura and Hasegawa 2004; Maeshima et al. 2007; Jhang and Liao 2010; Mak et al. 2005]. In [Fujimura and Hasegawa 2004], the authors proposed a Vehicle and Roadside Collaborative MAC protocol (VRCP) to support both V2V and I2V communications. In particular, two MAC modes are designed. One is the ad-hoc mode (Mode-A) for decentralized V2V communication. The other is infrastructure mode (Mode-I) for centralized I2V communication. [Maeshima et al. 2007] proposed a MAC protocol for emergency message delivery in I2V-based communication systems. In order to ensure timely delivery of emergency messages, whenever an emergency notification occurred on the control channel, the transmission of general information on the service channel would be suspended. [Jhang and Liao 2010] proposed a Proxy-based Vehicle to RSU (PVR) communication protocol based on IEEE 802.11 Distributed Coordination Function (DCF). It is designed to mitigate query upload contentions by electing proxy vehicles to help the data upload for other vehicles. Non-proxy vehicles which attempt to communicate with the RSU must forward their data items to a proxy vehicle. The proposed solution relieves the contention for the uplink channel and improves the throughput of the system. [Mak et al. 2005] proposed a coordinated MAC mode with the RSU. It aims to enhance system performance for both safety and non-safety applications by designing a multi-channel coordination mechanism, which is used to minimize collisions between V2V and I2V communications. These studies provided a solid basis for enhancing vehicular communication qualities at The MAC layer. Nevertheless, none of them addressed data scheduling problems at the application layer.

The data scheduling problem has been extensively studied in conventional mobile computing environments. A number of classical scheduling algorithms have been proposed for non-real-time data dissemination systems. FCFS (First Come First Served) [Wong and Ammar 1985] broadcasts data items sequentially according to the arrival order of requests. SRPT (Shortest Remaining Processing Time) [Acharya and Muthukrishnan 1998] chooses the data item with the shortest service time to broadcast. [Wong 1988] proposed two well-known algorithms for scheduling requests in on-demand broadcast systems: MRF (Most Requested First) and LWF (Longest Wait First). MRF broadcasts the data item which has the largest number of pending requests to account for the productivity of broadcast. LWF computes the total time that all pending requests for a data item have been waiting. The data item with the longest total waiting time is chosen to broadcast. [Aksoy and Franklin 1999] proposed a low overhead and scalable scheduling algorithm called RXW (Number of pending Requests Multiply Waiting time). It calculates the number of pending requests for a data item multiplied by the amount of time that the oldest outstanding request for that data item has been waiting. At each broadcast tick, the request with the maximum RXW value will be chosen. RXW combines the benefits of MRF and FCFS to enhance the scheduling performance.

In real-time scheduling, EDF (Earliest Deadline First) is one of the foremost classical scheduling algorithms, and it has been adopted into on-demand broadcast systems [Xuan et al. 1997]. EDF schedules the data item with the shortest remaining lifetime to cater for the urgency of requests. [Xu et al. 2006] proposed an on-line scheduling algorithm called SIN (Slack time Inverse Number of pending requests). It is designed for scheduling real-time requests in on-demand broadcast environments. The design of SIN is motivated by two strategies: EDF, which considers the urgency of requests, and MRF, which considers the productivity of data broadcast. It has been demonstrated that SIN significantly outperformed other existing real-time scheduling algorithms.

3. SYSTEM CHARACTERISTICS

The roadside-to-vehicle data dissemination system is shown in Figure 1, and the system characteristics are summarized as follows.

— Timeliness of data dissemination: The RSU is installed at a fixed position (e.g. at a road intersection) to provide a variety of services to passing vehicles. Vehicles are able to submit requests and retrieve data items within the service region of the RSU, which is represented by the dotted circle.

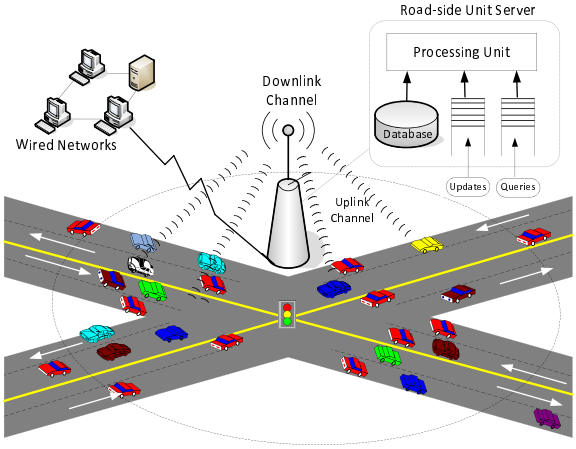

Fig. 1. System model of roadside-to-vehicle data dissemination

Specifically, requests are submitted via the uplink channel (i.e. the control channel as defined in IEEE 1609.4 [IEEE 2010]). According to a certain scheduling policy, the RSU selects data items from the local database for broadcasting via the downlink channel (i.e. the service channel as defined in IEEE 1609.4 [IEEE 2010]). This is known as the *on-demand broadcast* environment [Liu and Lee 2010a]. The limited radio coverage of the RSU and the high mobility of vehicles impose stringent time constraint on serving requests in such an environment [Liu and Lee 2012]. Therefore, it is critical to consider the timeliness of data dissemination for enabling time-critical applications.

— Temporality of data items: In order to maintain dynamic information in VCPS, such as traffic conditions, parking information and vehicle locations, etc., the RSU is connected to a *backbone network*, so that the data items stored in the local database are kept up-to-date by the sensors and information providers from the backbone network. The values of data items will be updated periodically, and only the latest version of each data item is maintained in the database. In other words, each version of the data item is only valid for a period of time. This is called the *temporal data item* [Lam et al. 2000]. Therefore, it is critical to consider the temporality of data items in scheduling to prevent receiving outdated information.

— Dependency of requested data items: For many applications in VCPS, it is necessary to retrieve multiple dependent data items. The query cannot be fully processed if only part of the requested data items are retrieved. In addition, it is expected that the versions of retrieved data items should be correlated closely in time so that the query result could be meaningful. For example, when the navigation system requests for the traffic conditions of several roads to make a routing decision, it is necessary to make sure that the retrieved information of different roads is up-to-date and comparable in terms of data versions. Otherwise, the query result would likely become useless or even have negative impact.

4. PROBLEM FORMULATION

The set of temporal data items in the database is denoted by $D = \{d_1, d_2, \cdots, d_{|D|}\}$, where $|D|$ is the total number of data items. The jth version of a temporal data item d_i $(1 \leq i \leq |D|)$ is denoted by $d_{i(j)}$ $(j = 1, 2, 3\cdots)$, and it is characterized by a 3-tuple: $< V(d_{i(j)}), U(d_{i(j)}), E(d_{i(j)}) >$, where $V(d_{i(j)})$ is the value of $d_{i(j)}$, while $U(d_{i(j)})$ and $E(d_{i(j)})$ represent the update time and the expiration time of $d_{i(j)}$, respectively. After time $E(d_{i(j)})$, a new version $d_{i(j+1)}$ will be created, and the values of $V(d_{i(j+1)})$, $U(d_{i(j+1)})$ and $E(d_{i(j+1)})$ are updated accordingly. Given the update interval of d_i, denoted by $l(d_i)$, the expiration time $E(d_i)$ can be represented by $E(d_i) = U(d_i) + l(d_i)$. To facilitate

Table I. Summary of notations

Notations	Descriptions	Notes
D	set of temporal data items	$D = \{d_1, d_2, \cdots, d_{\|D\|}\}$
$V(d_i\|t)$	value of d_i in its version at t	
$U(d_i\|t)$	update time of d_i in its version at t	$U(d_i\|t) \leq t$
$E(d_i\|t)$	expiration time of d_i in its version at t	$E(d_i\|t) > t$
$l(d_i)$	update interval of d_i	$U(d_i\|t) + l(d_i) = E(d_i\|t)$
τ	transmission time of a data item	
$RD(Q_m)$	set of data items requested by Q_m	$RD(Q_m) = \{d_m^1, d_m^2, \cdots d_m^{\|RD(Q_m)\|}\}$
$DL(Q_m)$	deadline of Q_m	
d_m^n	the n th data item requested by Q_m	$d_m^n \in D$
$BT_{Q_m}(d_m^n)$	broadcast time of d_m^n for Q_m	
FBT_{Q_m}	time to broadcast the first data item for Q_m	$FBT_{Q_m} = \min(BT_{Q_m}(d_m^n))$
LBT_{Q_m}	time to broadcast the last data item for Q_m	$LBT_{Q_m} = \max(BT_{Q_m}(d_m^n))$
$EE_{Q_m}(t)$	earliest expiration time of the requested data items	$EE_{Q_m}(t) = \min(E(d_m^n\|t))$
$US_{Q_m}(t)$	unserved set of Q_m	$US_{Q_m}(t) \subseteq RD(Q_m)$
$TTB_{Q_m}(t)$	tentative time bound for an unserved Q_m	$TTB_{Q_m}(t) = \min(DL(Q_m), EE_{Q_m}(t))$
DTB_{Q_m}	determined time bound for a partially-served Q_m	$DTB_{Q_m} = \min(DL(Q_m), EE_{Q_m}(FBT_{Q_m}))$
$SQ(t)$	set of schedulable requests	
$EDP_{d_m^n}(t)$	effective data productivity of d_m^n	
$ERP_{Q_m}(t)$	effective request productivity of Q_m	
$RR_{Q_m}(t)$	remaining ratio of Q_m	
$FSP_{Q_m}(t)$	feasible scheduling period for Q_m	the length is denoted by $\|FSP_{Q_m}(t)\|$
$RSQ(t)$	set of reschedulable requests	

the analysis, the notations $U(d_i|t)$ and $E(d_i|t)$ are adopted to represent the update time and expiration time of d_i in its version at time t, respectively. A request Q_m ($m = 1, 2, 3\cdots$) is characterized by a 3-tuple: $< RD(Q_m), ST(Q_m), DL(Q_m) >$. Concretely, $RD(Q_m)$ is the set of requested data items, and it is represented by $RD(Q_m) = \{d_m^1, d_m^2, \cdots d_m^{|RD(Q_m)|}\}$, where $|RD(Q_m)|$ is the number of requested data items, and $d_m^n \in D$ ($1 \leq n \leq |RD(Q_m)|$). $ST(Q_m)$ is the time when Q_m is submitted. $DL(Q_m)$ is the deadline of Q_m. Outstanding requests are pended in the service queue at the server, which is denoted by $Q(t)$. Based on a certain scheduling algorithm, the server broadcasts data items to serve pending requests. The time taken to broadcast a data item is denoted by τ, which is referred to as the *transmission time*. A request will be removed from $Q(t)$ when it is satisfied or its deadline expires. The primary notations are summarized in Table I.

In the following, we first recapitulate the key idea of serving requests with *static snapshot consistency requirement*, which has been intensively investigated in our previous work [Liu et al. 2013]. On this basis, we introduce the *dynamic snapshot consistency requirement* on serving real-time requests for temporal data items.

4.1. Static snapshot consistency requirement

At time t, the database maintains the latest version of each temporal data item, which forms the current *snapshot* of the database. Consider a request asking for multiple dependent temporal data items. In order to make the query result meaningful, it is expected that the versions of all the retrieved data items should coexist in the database when the query is processed. In this regard, we introduce the *static snapshot consistency requirement*. Generally, when all the data items are retrieved by a request, they should be in the same snapshot with respect to a particular time instance, while this time instance is determined by the time when the first data item for this request is disseminated. In other words, when the version of the first retrieved data item is determined, the versions of remaining

data items should be in the same snapshot regarding to the first one. Specifically, there are three conditions to serve a request Q_m with the static snapshot consistency requirement.

— The broadcast time of each d_m^n for Q_m, denoted by $BT_{Q_m}(d_m^n)$, should be later than the time when Q_m is submitted, because the vehicle only monitors and retrieves data items after submitting the request. The first condition is represented by:

$$FBT_{Q_m} > ST(Q_m) \tag{1}$$

where FBT_{Q_m} is the time to broadcast the first data item for Q_m, which is computed by $FBT_{Q_m} = \min(BT_{Q_m}(d_m^n))$, $\forall d_m^n \in RD(Q_m)$.

— Each d_m^n has to be retrieved before the request deadline $DL(Q_m)$. The second condition is represented by:

$$LBT_{Q_m} + \tau \leq DL(Q_m) \tag{2}$$

where LBT_{Q_m} is the time to broadcast the last data item for Q_m, which is computed by $LBT_{Q_m} = \max(BT_{Q_m}(d_m^n))$, $\forall d_m^n \in RD(Q_m)$, and τ is the transmission time of a data item.

— The version of each retrieved data item should be in the same snapshot regarding to the time when the first data item is disseminated for Q_m (i.e. FBT_{Q_m}). In order to meet this requirement, Q_m has to retrieve all its required data items before any of them is updated since time FBT_{Q_m}. Given the expiration time of d_m^n at time FBT_{Q_m}, which is denoted by $E(d_m^n|FBT_{Q_m})$, the third condition is represented by:

$$LBT_{Q_m} + \tau \leq EE_{Q_m}(FBT_{Q_m}) \tag{3}$$

where $EE_{Q_m}(FBT_{Q_m})$ represents the earliest expiration time of the requested data items, which is computed by $EE_{Q_m}(FBT_{Q_m}) = \min(E(d_m^n|FBT_{Q_m}))$, $\forall d_m^n \in RD(Q_m)$.

The first two conditions are straightforward. The third condition defines the static snapshot consistency requirement. Clearly, it is more challenging to schedule real-time requests for temporal data items. This is because in addition to the request deadline, the temporality of data items imposes strict timing constraints on satisfying the static snapshot consistency requirement.

4.2. Dynamic snapshot consistency requirement

The static snapshot consistency requirement restricts that the versions of all the retrieved data items for a request have to be in a particular snapshot, while this snapshot has been fixed since the first data item of this request is disseminated. In other words, there is only one valid snapshot for serving a request, which determines the versions of all the requested data items. Nevertheless, in practice, it may not necessary to have such a strict constraint. Consider the case where Q_m asks for data items a and b. At t_1, the ith version of a (i.e. a_i) is scheduled to broadcast. Meanwhile, b is in its jth version (i.e. b_j) in the database. As a matter of fact, instead of having to retrieve b_j in regard to the snapshot of t_1, it is also viable to serve Q_m by retrieving a newer version of b (e.g. b_{j+1}), as long as a_i is still valid by the time b_{j+1} is retrieved. In such a case, although a_i and b_{j+1} are retrieved in different snapshots, they still coexist in the database when the query is processed. Therefore, it is reasonable to have such a schedule to serve Q_m. To sum up, with a proper schedule, the time instance of the snapshot can be dynamic when serving a request. We give the following example to better illustrate this idea.

As shown in Figure 2, suppose Q_m is submitted at t_0 ($ST(Q_m) = t_0$), and it requests data items a, b and c. The deadline of Q_m is t_6 ($DL(Q_m) = t_6$). At t_0, the versions of each requested data item are a_i, b_j and c_k, respectively. The data item b is updated at t_1, when a new version b_{j+1} is generated. Similarly, data items c and a are updated at t_2 and t_3, respectively. According to the schedule shown in Figure 2, a is selected to broadcast at $BT_{Q_m}(a)$, and accordingly, the version of a_i is retrieved. With the static snapshot consistency requirement, the versions for b and c are determined in this snapshot (i.e. b_j and c_k). That is, both b and c have to be retrieved before t_1. Apparently, in this example, there is no chance to satisfy Q_m with such a requirement. Nevertheless, with dynamic

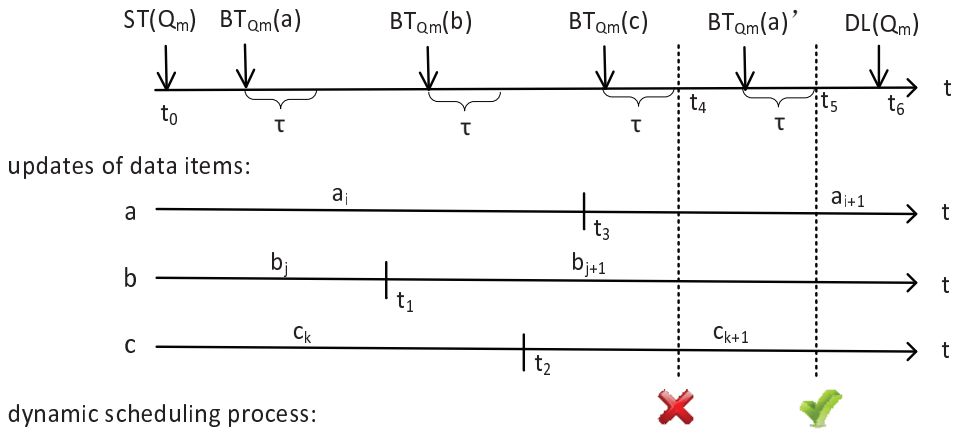

Fig. 2. An example of dynamic snapshot consistency requirement

snapshot consistency requirement, since a_i is still valid after t_1 but before t_3, Q_m can wait to retrieve a later version of b after t_1 (i.e. b_{j+1}). In such a case, a_i and b_{j+1} coexist in another snapshot at time $BT_{Q_m}(b) + \tau$. Therefore, even though b has been updated since the retrieval of a, the dynamic snapshot consistency requirement still holds. However, if c is selected to broadcast at $BT_{Q_m}(c)$, when it is retrieved at t_4, a new version of a (i.e. a_{i+1}) has been generated. In such a case, the retrieved data items a_i, b_{j+1} and c_{k+1} are not in the same snapshot at t_4, which violates the dynamic snapshot consistency requirement. Since the request deadline (t_6) has not yet expired, a can be rescheduled to broadcast at time $BT_{Q_m}(a)'$. Accordingly, a_{i+1} is retrieved at t_5. At this time, all the requested data items (a_{i+1}, b_{j+1} and c_{k+1}) have been retrieved before the deadline, and they are in the same snapshot at t_5. Therefore, Q_m is satisfied.

With the above analysis, the dynamic snapshot consistency requirement for serving Q_m is defined as follows.

— The version of each retrieved data item should be in the same snapshot at the time when all the data items are retrieved (i.e. $LBT_{Q_m} + \tau$). In other words, each retrieved data item has to be in its latest version at time $LBT_{Q_m} + \tau$. This condition is represented by:

$$BT_{Q_m}(d_m^n) > U(d_m^n | (LBT_{Q_m} + \tau)), \forall d_m^n \in RD(Q_m) \quad (4)$$

where $BT_{Q_m}(d_m^n)$ is the broadcast time of d_m^n, and $U(d_m^n|(LBT_{Q_m}+\tau))$ is the update time of d_m^n at time $LBT_{Q_m}+\tau$.

With such a dynamic snapshot consistency requirement, the time bound for serving Q_m cannot be determined when any individual requested data item is retrieved. This is because the snapshot can be adjusted dynamically with the broadcast of each requested data item for Q_m.

5. PROPOSED ALGORITHMS

Based on the above two types of requirements on serving real-time requests for temporal data items in VCPS, we propose two on-line scheduling algorithms accordingly, which are SSCS (Static Snapshot Consistency oriented Scheduling) and DSCS (Dynamic Snapshot Consistency oriented Scheduling).

5.1. Static snapshot consistency oriented scheduling

In general, the objectives of SSCS include exploiting the broadcast effect, improving the bandwidth utilization and enhancing the request service. Before introducing the detailed steps of SSCS, several preliminary definitions are stated as follows.

Definition 5.1. Unserved set of a request: At time t, the set of unserved data items of a request Q_m is represented by $US_{Q_m}(t)=\{d_m^{1\prime},d_m^{2\prime},\cdots d_m^{|US_{Q_m}(t)|\prime}\}$, where $|US_{Q_m}(t)|$ is the number of unserved data items ($0\leq|US_{Q_m}(t)|\leq|RD(Q_m)|$), and $US_{Q_m}(t)\subseteq RD(Q_m)$.

Given a pending request Q_m, if none of its requested data items have been retrieved ($US_{Q_m}(t)=RD(Q_m)$), Q_m is considered as the *unserved request*. In contrast, if part of the requested data items have been retrieved ($US_{Q_m}(t)\subset RD(Q_m)$ and $US_{Q_m}(t)\neq\emptyset$), Q_m is considered as the *partially-served request*. Recall that Q_m has to be served before its request deadline. Meanwhile, each requested data item is associated with an expiration time, which changes dynamically with the update of the data item. Therefore, there is a practical time bound to serve Q_m when considering both the request deadline and the update of each requested data item. With the static snapshot consistency requirement, the snapshot is determined when the first data item is disseminated for Q_m. This gives different attributes of time bounds for unserved requests and partially-served requests. Specifically, the time bound is dynamic for unserved requests, because the time to broadcast its first data item is unknown. In contrast, the time bound is determined for partially-served requests, since the snapshot to satisfy this request has been fixed. In this regard, we define the *tentative time bound* and the *determined time bounds* for unserved requests and partially-served requests, respectively.

Definition 5.2. Tentative time bound: At time t, if Q_m is an unserved request, the tentative time bound for Q_m ($TTB_{Q_m}(t)$) is either its request deadline ($DL(Q_m)$), or the earliest expiration time of its requested data items at time t ($EE_{Q_m}(t)$), whichever is earlier. That is,

$$TTB_{Q_m}(t)=\min(DL(Q_m),EE_{Q_m}(t)) \tag{5}$$

where $EE_{Q_m}(t)=\min(E(d_m^n|t))$, $\forall d_m^n\in RD(Q_m)$.

$TTB_{Q_m}(t)$ is not the finalized time bound of Q_m, and the value of $TTB_{Q_m}(t)$ may change with t. The reason is that, although the request deadline $DL(Q_m)$ is a constant, the value of $EE_{Q_m}(t)$ may vary with time. Therefore, at different scheduling points, the dynamic value of $EE_{Q_m}(t)$ may result in different $TTB_{Q_m}(t)$.

With the static snapshot requirement, as soon as the service starts for Q_m, the versions of its requested data items, as well as the corresponding expiration time of these data items will be determined. In other words, for any requests which have been scheduled to serve, the time bounds for serving these requests have been determined. In view of this, we define the *determined time bound* for the partially-served request as follows.

Definition 5.3. Determined time bound: For a partially-served request Q_m, given the time FBT_{Q_m} when the first data item is broadcast, the determined time bound for serving Q_m (DTB_{Q_m}) is either its request deadline $DL(Q_m)$, or the earliest expiration time of its requested data items at time FBT_{Q_m} ($EE_{Q_m}(FBT_{Q_m})$), whichever is earlier. That is,

$$DTB_{Q_m} = \min(DL(Q_m), EE_{Q_m}(FBT_{Q_m})) \tag{6}$$

where $EE_{Q_m}(FBT_{Q_m}) = \min(E(d_m^n | FBT_{Q_m}))$, $\forall d_m^n \in RD(Q_m)$.

At each scheduling point, the system may have both unserved and partially-served pending requests. In order to check whether a request has the chance to be satisfied at time t, we define the *schedulable request* ($SQ(t)$) as follows.

Definition 5.4. Schedulable request: At time t, the set of schedulable requests, denoted by $SQ(t)$, is the union of the set of *schedulable unserved requests* ($SQ_u(t)$) and the set of *schedulable partially-served requests* ($SQ_p(t)$). That is,

$$SQ(t) = SQ_u(t) \cup SQ_p(t) \tag{7}$$

where $SQ_u(t)$ and $SQ_p(t)$ are obtained as follows.

— $SQ_u(t)$: For any unserved request Q_m, it is schedulable if Q_m can retrieve all of its requested data items before the tentative time bound $TTB_{Q_m}(t)$. That is, $SQ_u(t) = \{Q_m | US_{Q_m}(t) = RD(Q_m) \wedge t + |RD(Q_m)| \cdot \tau \leq TTB_{Q_m}(t)\}$, where $|RD(Q_m)|$ is the total number of requested data items, and τ is the transmission time of a data item.
— $SQ_p(t)$: For any partially-served request Q_m, it is schedulable if Q_m can retrieve the remaining requested data items before the determined time bound DTB_{Q_m}. That is, $SQ_p(t) = \{Q_m | US_{Q_m}(t) \subset RD(Q_m) \wedge US_{Q_m}(t) \neq \emptyset \wedge t + |US_{Q_m}(t)| \cdot \tau \leq DTB_{Q_m}\}$, where $|US_{Q_m}(t)|$ is the number of unserved data items.

Based on the above definitions, the operations of SSCS along with the designing rationales are presented as follows.

Unlike the conventional unicast, where a data item must be transmitted many times to serve multiple requests, one broadcast data item has the potential to serve all its pending requests. This is called the *data productivity* [Aksoy and Franklin 1999; Xu et al. 2006]. In order to exploit the broadcast effect, intuitively, the algorithm should schedule the data item with high productivity. In view of this, we define the *effective data productivity* as follows.

Definition 5.5. Effective data productivity. At time t, denote $Q_{d_i}(t)$ as the set of requests formed by any Q_m which satisfies the following two conditions:

— d_i is in the unserved set of Q_m, namely, $d_i \in US_{Q_m}(t)$.
— Q_m is schedulable at time t, namely, $Q_m \in SQ(t)$.

The effective data productivity of d_i ($1 \leq i \leq |D|$), denoted by $EDP_{d_i}(t)$, is defined as the number of requests in $Q_{d_i}(t)$, which is represented by:

$$EDP_{d_i}(t) = \left|Q_{d_i}(t)\right| \tag{8}$$

Accordingly, for a multi-item request, the *effective request productivity* is defined as follows.

Definition 5.6. Effective request productivity. At time t, the effective request productivity of Q_m, denoted by $ERP_{Q_m}(t)$, is the average of the effective data productivity for its unserved data items, which is computed by:

$$ERP_{Q_m}(t) = \Big(\sum_{d_i \in US_{Q_m}(t)} EDP_{d_i}(t)\Big) / |US_{Q_m}(t)| \tag{9}$$

With the above analysis, in order to exploit broadcast effect, the algorithm should give higher priority to a request with higher value of $ERP_{Q_m}(t)$.

Due to the broadcast effect, it is possible that even though some requests have not yet been scheduled to serve, they may have retrieved part of their data items due to the service for other previously selected requests. As time passes, it is likely that there are some requests which have retrieved a large percentage of its required data items. Pushing these requests towards completion will be helpful for improving the bandwidth utilization. To this end, we define the *remaining ratio* to capture the serving status of requests.

Definition 5.7. Remaining ratio. At time t, the remaining ratio of Q_m, denoted by $RR_{Q_m}(t)$, is the number of its unserved data items over the total number of requested data items, which is computed by:

$$RR_{Q_m}(t) = \frac{|US_{Q_m}(t)|}{|RD(Q_m)|} \tag{10}$$

A small value of the remaining ratio implies that a large percentage of the requested data items have been retrieved, and the request is close to be satisfied. Giving higher priority to a request with a smaller value of remaining ratio will help to improve the bandwidth utilization.

In order to enhance the service chance for real-time requests, we introduce the concept of *feasible and non-feasible scheduling segments* and design a new metric called the *feasible scheduling period* to capture the request urgency. First, depending on the serving status of Q_m at time t, there are two cases for identifying feasible/non-feasible scheduling segments.

— Case a): Q_m is an unserved request: if Q_m is schedulable (i.e. $t + |RD(Q_m)| \cdot \tau \leq TTB_{Q_m}(t)$), then $[t, TTB_{Q_m}(t)]$ is a *feasible scheduling segment*. Otherwise, $[t, TTB_{Q_m}(t)]$ is a *non-feasible scheduling segment*.
— Case b): Q_m is a partially-served request: if Q_m is schedulable (i.e. $t + |US_{Q_m}(t)| \cdot \tau \leq DTB_{Q_m}$), then $[t, DTB_{Q_m}]$ is a *feasible scheduling segment*. Otherwise, $[t, DTB_{Q_m}]$ is a *non-feasible scheduling segment*.

It is worth noting that in Case b), $[t, DTB_{Q_m}]$ is the only possible feasible scheduling segment if Q_m is schedulable at time t. However, in Case a), Q_m may have multiple feasible/non-feasible scheduling segments. Details regarding to this concept can be referred to [Liu et al. 2013]. With the above knowledge, we define the *feasible scheduling period* for Q_m as follows.

Definition 5.8. Feasible scheduling period. At time t, the feasible scheduling period for Q_m, denoted by $FSP_{Q_m}(t)$, is the union of the feasible scheduling segments (FSS_{Q_m}) in $[t, DL(Q_m)]$, which is represented by:

$$FSP_{Q_m}(t) = \cup FSS_{Q_m} \ in \ [t, DL(Q_m)] \tag{11}$$

For a partially-served Q_m, there is only one possible feasible scheduling segment FSS_{Q_m}, which is $[t, DTB_{Q_m}]$. Therefore, $FSP_{Q_m}(t)$ is either $[t, DTB_{Q_m}]$ or $\emptyset$, depending on whether $[t, DTB_{Q_m}] \subseteq FSS_{Q_m}$. For an unserved Q_m, the $FSP_{Q_m}(t)$ is computed by cumulating all feasible scheduling segments. Note that these feasible scheduling segments may not be consecutive. The length of feasible scheduling period $|FSP_{Q_m}(t)|$ reflects the actual duration in which a request can be served. Therefore, in order to enhance request service chance, the algorithm should give higher priority to a request with shorter length of $|FSP_{Q_m}(t)|$.

To sum up, for the sake of exploiting the broadcast effect, improving the bandwidth utilization, and enhancing the service chance, the request priority is defined as follows.

Definition 5.9. Request priority. At time t, the priority of Q_m ($Priority_{Q_m}(t)$), which is a compound effect of the effective request productivity ($ERP_{Q_m}(t)$), the remaining ratio ($RR_{Q_m}(t)$), and

Algorithm 1: SSCS

Input: pending requests in the service queue $Q(t)$
Output: the set of data items for disseminating
// Construct the set of schedulable requests $SQ(t)$
$SQ(t) \leftarrow \emptyset$;
for each $Q_m \in Q(t)$ **do**
  if $US_{Q_m}(t) == RD(Q_m)$ **then**
    Compute the tentative time bound of Q_m ($TTB_{Q_m}(t)$);
    if $t + |RD(Q_m)| \cdot \tau \leq TTB_{Q_m}(t)$ **then**
      $SQ(t) \leftarrow SQ(t) \cup \{Q_m\}$;
    end if
  else
    if $t + |US_{Q_m}(t)| \cdot \tau \leq DTB_{Q_m}$ **then**
      $SQ(t) \leftarrow SQ(t) \cup \{Q_m\}$;
    end if
  end if
end for
// Find the request with the highest priority
$maxPriority \leftarrow 0$;
for each $Q_m \in SQ(t)$ **do**
  $ERP_{Q_m}(t) \leftarrow 0$;
  for each $d_i \in US_{Q_m}(t)$ **do**
    $ERP_{Q_m}(t) \leftarrow ERP_{Q_m}(t) + EDP_{d_i}(t)$;
  end for
  $ERP_{Q_m}(t) \leftarrow ERP_{Q_m}(t) / |US_{Q_m}(t)|$;
  $RR_{Q_m}(t) \leftarrow \frac{|US_{Q_m}(t)|}{|RD(Q_m)|}$;
  Compute the length of the feasible scheduling period $|FSP_{Q_m}(t)|$;
  $Priority_{Q_m}(t) \leftarrow \frac{ERP_{Q_m}(t)}{RR_{Q_m}(t) \cdot |FSP_{Q_m}|(t)}$;
  if $maxPriority < Priority_{Q_m}(t)$ **then**
    $maxPriority \leftarrow Priority_{Q_m}(t)$;
    $Q_{selected} \leftarrow Q_m$;
  end if
end for
// Broadcast the set of scheduled data items and update the service queue
for each $d_i \in US_{Q_{selected}}(t)$ **do**
  Broadcast d_i;
  for each $Q_s \in Q(t)$ **do**
    if $t > DL(Q_s)$ **then**
      $Q(t) \leftarrow Q(t) - \{Q_s\}$;
    else
      if $d_i \in US_{Q_s}(t)$ **and** $Q_s \in SQ(t)$ **then**
        $US_{Q_s}(t) \leftarrow US_{Q_s}(t) - \{d_i\}$;
        if d_i is the first retrieved data item for Q_s **then**
          Compute the determined time bound of Q_s (DTB_{Q_s});
        end if
        if $US_{Q_s}(t) == \emptyset$ **then**
          $Q(t) \leftarrow Q(t) - \{Q_s\}$;
        end if
      end if
    end if
  end for
end for

the length of feasible scheduling period ($|FSP_{Q_m}(t)|$), is computed by:

$$Priority_{Q_m}(t) = \frac{ERP_{Q_m}(t)}{RR_{Q_m}(t) \cdot |FSP_{Q_m}(t)|} \tag{12}$$

At each scheduling point, SSCS first constructs the set of schedulable requests $SQ(t)$. Then it computes the priority $Priority_{Q_m}(t)$ for each $Q_m \in SQ(t)$ and selects the one with the highest priority. Finally, all the unserved data items of the selected request are broadcast successively, and the status of the pending requests are updated accordingly. The pseudo code of SSCS is shown in Algorithm 1.

5.2. Dynamic snapshot consistency oriented scheduling

When considering the dynamic snapshot consistency requirement, as analyzed in Section 4.2, the time bound for serving a request cannot be determined even when some data items have been retrieved by this request. In other words, the determined time bound for partially-served requests in Definition 5.3 is exclusively defined for scheduling with the static snapshot consistency requirement, whereas it cannot be adopted in scheduling with this new requirement. For example, as shown in Figure 2, when a is broadcast at time $BT_{Q_m}(a)$, the determined time bound for serving Q_m will be set to t_1 according to Definition 5.3. Nevertheless, with the dynamic snapshot consistency requirement, as shown in this example, Q_m can be served in another snapshot at the time instance of t_5 rather than the one at the time instance of $BT_{Q_m}(a)$. Due to such an uncertainty of time bounds for serving requests with the dynamic snapshot consistency requirement, it is desirable to have a more sophisticated scheduling algorithm to enhance the system performance. With this motivation, we propose DSCS by designing a *reschedule mechanism* based on SSCS.

The key observations on designing the reschedule mechanism are introduced as follows. First, we note that the static snapshot consistency requirement is a special case of the dynamic one. Therefore, for any requests which can be satisfied under the scheduling of SSCS, they will be certainly satisfied with the dynamic snapshot consistency requirement. Second, for those requests which can not be served by SSCS due to the constraint that there is only one viable snapshot for serving each request, they may still have chance to be served with the dynamic snapshot consistency requirement. This is because as long as the request deadline has not yet expired, the request always has chance to be served by retrieving the versions of data items in the current snapshot (but the request may have to discard some of its previously retrieved data items which have become outdated). For example, at the snapshot of t_5 as shown in Figure 2, Q_m can be served by discarding the outdated version of a (i.e. a_i) and retrieving the current version of a (i.e., a_{i+1}). In such a case, the retrieved data items a_{i+1}, b_{j+1} and c_{k+1} are still in the same snapshot at t_5, which satisfy the dynamic snapshot consistency requirement. Based on this observation, we define the *reschedulable request* as follows.

Definition 5.10. Reschedulable request. At time t, if the determined time bound for Q_m has passed (i.e. $t > DTB_{Q_m}$), but the deadline of Q_m has not yet expired (i.e. $t < DL(Q_m)$), then Q_m is a reschedulable request. The set of reschedulable requests is represented by $RSQ(t) = \{Q_m | t > DTB_{Q_m} \wedge t < DL(Q_m)\}$.

With the above knowledge, the reschedule mechanism consists of three steps, which are introduced as follows.

— Step 1: Given the set of reschedulable requests $RSQ(t)$, update the consistency requirement for each $Q_m \in RSQ(t)$ according to the versions of the requested data items at the current snapshot.
— Step 2: Compare the versions of the retrieved data items with the updated consistency requirement for each $Q_m \in RSQ(t)$. In case there is any retrieved data item which has become outdated, remove this data item from Q_m and update the unserved set of Q_m (i.e. $US_{Q_m}(t)$) accordingly.
— Step 3: Recompute the time bound for each $Q_m \in RSQ(t)$ based on expiration time of data items at the current snapshot (i.e. $E(d_i|t)$ for each $d_i \in RD(Q_m)$) and the request deadline (i.e. $DL(Q_m)$).

Algorithm 2: DSCS

Input: pending requests in the service queue $Q(t)$
Output: the set of data items for disseminating
// Find the set of reschedulable requests
1: $RSQ(t) \leftarrow \emptyset$;
2: **for** each $Q_s \in Q(t)$ **do**
3: **if** $US_{Q_s}(t) \subset RD(Q_s)$ **then**
4: **if** $t > DTB_{Q_s}$ **and** $DTB_{Q_s} < DL(Q_s)$ **then**
5: $RSQ(t) \leftarrow RSQ(t) + \{Q_s\}$;
6: **end if**
7: **end if**
8: **end for**
// Update the unserved set and the time bound for each reschedulable request
9: **for** each $Q_m \in RSQ(t)$ **do**
10: $newTimeBound \leftarrow DL(Q_m)$;
11: **for** each $d_i \in RD(Q_m)$ **do**
12: **if** $d_i \notin US_{Q_m}(t)$ **and** $V(d_i|BT_{Q_m}(d_i)) \neq V(d_i|t)$ **then**
13: $US_{Q_m}(t) \leftarrow US_{Q_m}(t) + \{d_i\}$;
14: **end if**
15: **if** $E(d_i|t) < newTimeBound$ **then**
16: $newTimeBound \leftarrow E(d_i|t)$;
17: **end if**
18: **end for**
19: $DTB_{Q_m} \leftarrow newTimeBound$;
20: **end for**
// Invoke SSCS
21: SSCS();

For a reschedulable request, since its previously determined time bound has expired, this request can be served if it could retrieve the version of each requested data item in the current snapshot. Therefore, in Step 1, the algorithm should update the consistency requirement according to the current version of each requested data item. For example, at time $BT_{Q_m}(b)$ as shown in Figure 2, Q_m is a reschedulable request because its previously determined time bound (i.e. t_1 which is determined when a is broadcast at $BT_{Q_m}(a)$) has expired, but it is still before the request deadline (i.e. t_6). Therefore, to satisfy the snapshot consistency requirement at time $BT_{Q_m}(b)$, the versions of data items to be retrieved by Q_m should be updated to a_i, b_{j+1} and c_k accordingly.

Due to the update of the required version for each requested data item, some of the previously retrieved data items by the request may have become outdated. Therefore, in Step 2, the algorithm checks the version of each retrieved data item and discards the invalid data items. As a result, the unserved set of this request should be updated accordingly by adding the discarded data items. For example, at time $BT_{Q_m}(b)$ as shown in Figure 2, a_i has been retrieved and its version is still valid in the current snapshot. In this case, the unserved set of Q_m remains the same. However, at time $BT_{Q_m}(c)$, which is later than t_3, the required version of each data item becomes a_{i+1}, b_{j+1} and c_{k+1}, respectively. In such a case, the previously retrieved a_i becomes outdated, and it should be discarded. Accordingly, the data item a will be added into the unserved set of Q_m.

Since the required versions of data items and the unserved set of the request may have changed with above operations, the time bound for serving a reschedulable request will be changed accordingly. Therefore, in Step 3, the algorithm checks the expiration time of the current required versions of data items and the request deadline to recompute the time bound. Specifically, the Equation 5 can be adopted to compute the new time bound. Note that in such a context, this equation is no longer exclusively used for computing the tentative time bound for unserved requests, but also for computing the new time bound for reschedulable requests. For example, at time $BT_{Q_m}(c)$ as shown in Figure 2, the new time bound for Q_m is computed by $\min(DL(Q_m), EE_{Q_m}(t))$, which equals the

Table II. Default setting

Parameter	Default	Description
$\|D\|$	100	number of temporal data items
$\frac{1}{\lambda}$	0.6	mean inter-arrival time of requests
s	3	request size
L_{min}	70	minimum tolerated latency
L_{max}	90	maximum tolerated latency
T_{min}	200	minimum update period
T_{max}	300	maximum update period
θ	0.6	Zipf distribution parameter

request deadline t_6. To be compatible with the scheduling of SSCS, the previously determined time bound of Q_m is replaced by this new time bound for priority computing.

Based on the above reschedule mechanism, the procedures of DSCS are described as follows. First, it examines each partially-served request in the service queue and constructs the set of reschedulable requests $RSQ(t)$ based on Definition 5.10. Second, it updates the required versions of data items and recomputes the time bound for each $Q_m \in RSQ(t)$ based on the reschedule mechanism. Finally, it invokes SSCS to select the most rewarding request for serving. The pseudo code of DSCS is shown in Algorithm 2.

Note that the implementation of both SSCS and DSCS rely on the centralized control at the RSU. Specifically, RSU maintains the temporal database and it is aware of the update of each data item. Meanwhile, vehicles submit their requests via the uplink channel, and the RSU maintains the service status for each pending request. Accordingly, the scheduler adopted at the RSU is able to make scheduling decisions with the proposed algorithms.

6. PERFORMANCE EVALUATION

6.1. Simulation model

We build the simulation model based on the system architecture shown in Section 3. The model is implemented by C programming with CSIM19 [Schwetman 2001]. The arrival of vehicles follows the Poisson process with the parameter λ, which is widely assumed in vehicular networks [Liu and Lee 2010b; Comert and Cetin 2011; Fallah et al. 2011]. In order to evaluate algorithm performance under stressful settings, we consider a saturated scenario where every passing vehicle will submit a request. Therefore, the inter-arrival time of requests follows the Exponential distribution with mean value of $\frac{1}{\lambda}$. Each request may ask for multiple dependent data items and the request size (s) is the number of required data items. Each request is associated with a deadline, which is obtained by $t\_arrival + t\_relative$, where $t\_arrival$ is the request submission time, and $t\_relative$ is the relative deadline. The value of $t\_relative$ is uniformly selected from the range $(L_{\min}, L_{\max})$, where L_{min} and L_{max} represent the minimum and the maximum tolerated latency for serving requests, respectively. The database D consists of $|D|$ temporal data items. Each d_i $(1 \leq i \leq |D|)$ has an update interval of $l(d_i)$, which is uniformly generated from the range $(T_{\min}, T_{\max})$, where T_{min} and T_{max} represent the minimum and the maximum update periods of data items, respectively. The data access pattern follows the commonly used Zipf distribution [Zipf 1949] with the parameter θ. The time unit adopted in the simulation is the transmission time of a data item, which is also called the *broadcast tick*. The main parameters and their corresponding descriptions are summarized in Table II. Unless stated otherwise, the simulations are conducted under the default setting.

In the simulation, we do not specify the absolute values of the data size and the broadcast bandwidth, but rather use the broadcast tick as the time unit for performance evaluation with the purpose of emphasizing the general applicability of our analysis. Nevertheless, we justify the practicality of the chosen parameters with the following example. Suppose the data transmission rate is 6 Mbps and the I2V communication radius is 300 m. Such settings have been demonstrated reliable in vehicular communications with DSRC [Bai and Krishnan 2006]. Suppose the data size is 250 KB (i.e.

$250 * 8 = 2\,\text{Mb}$), which is sufficiently large for normal data items. Accordingly, a broadcast tick is computed by $\frac{2\,\text{Mb}}{6\,\text{Mbps}} = 0.33\,\text{s}$. In the default setting, the mean tolerated latency of requests is 80 broadcast ticks, which is 26.4 s. Consider the relative deadline of requests is bounded by the dwell time of the corresponding vehicles in the communication range of the RSU, namely, the mean dwell time of vehicles is 26.4 s. Given the communication radius of 300 m, it implies that the mean velocity of vehicles is $\frac{600\,\text{m}}{26.4\,\text{s}} = 22.7\,\text{m/s}$ (i.e. 81.7 km/h). In addition, the mean inter-arrival time of requests is 0.6 broadcast ticks in the default setting, which equals 0.2 s in this example. We consider a stressful environment where every passing vehicle will submit a request. This implies that, on average, there are 5 newly coming vehicle per second. Considering a 4-lane road on each direction, there are 16 lanes in a 4-way intersection. Accordingly, the inter arrival time of vehicles on each lane is around 3 s, which is reasonable in realistic traffic environments.

For performance comparison, we implement two well-known real-time scheduling algorithms. One is EDF (Earliest Deadline First) [Xuan et al. 1997], and the other is SIN (Slack time Inverse Number of pending requests) [Xu et al. 2006]. The simulations are conducted in an ideal environment without packet loss. Note that we compare the performance of algorithms under the same setting. Therefore, when other factors such as packet loss are taken into consideration, the relative performance of algorithms is supposed to be the same, and the observations from our simulation results will still hold. Detailed characteristics of DSRC-based communication can be referred to [Bai et al. 2010; Bai and Krishnan 2006], where the lower layer communication characteristics have been extensively examined in a variety of realistic driving environments. Finally, in order to give a quantitative performance analysis, the following metrics are adopted.

— Service ratio: It is the ratio of the number of satisfied requests to the total number of submitted requests. In real-time data dissemination systems, the foremost goal of a scheduling algorithm is to maximize the service ratio.
— Bandwidth saving ratio: It is designed to measure the bandwidth utilization in broadcast environments, which is computed by $\frac{N_{rev}-N_{bst}}{N_{req}}$, where N_{rev} is the total number of data items received by satisfied requests; N_{bst} is the number of broadcast data items; N_{req} is the total number of data items required by all the requests. A high bandwidth saving ratio indicates an efficient utilization of the broadcast bandwidth for satisfying requests.

6.2. Simulation results

In the following, we evaluate the algorithm performance for serving requests with the dynamic snapshot consistency requirement by examining the effect of three critical factors which have significant impact on system performance, including the request deadline, the data update period and the system workload.

6.2.1. Effect of request deadline. Figure 3 shows the service ratio of each algorithm under different relative deadlines. As expected, all the algorithms achieve a higher service ratio when the relative deadline increases. It is noted that the advantages of SSCS and DSCS are more significant in a looser request deadline environment, especially for DSCS. This is because all the algorithms consider the request deadline in scheduling, which are supposed to have reasonable performance in a tight request deadline environment. Nevertheless, with the increase of the relative deadline, the factor of data expiration gradually dominates the algorithm performance. Because SSCS considers the tentative and determined request time bounds in scheduling, it can address the snapshot consistency requirement to a certain extent, which gives its better performance comparing with EDF and SIN. Nevertheless, the time bound derived by SSCS has a strict constraint because there is only one viable snapshot for serving a request. In contrast, DSCS incorporates the reschedule mechanism in scheduling, which makes it adapt to the dynamic snapshot consistency requirement. As demonstrated, DSCS achieves the best performance. Figure 4 shows the bandwidth saving ratio of each algorithm under different relative deadlines. As shown, SSCS and DSCS achieve much higher bandwidth utilization than EDF and SIN. The relative performance of algorithms is consistent with the

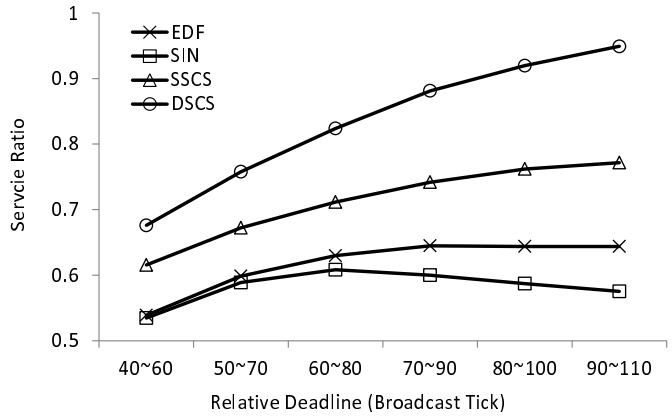

Fig. 3. Service ratio under different relative deadlines

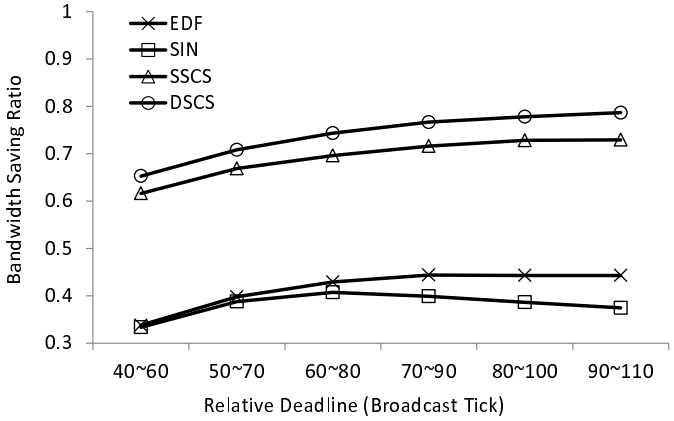

Fig. 4. Bandwidth saving ratio under different relative deadlines

result shown in Figure 3. This observation confirms the claim that a higher bandwidth saving ratio indicates that the algorithm is better at exploiting the broadcast bandwidth on satisfying requests.

6.2.2. Effect of data update period. The update period of data items is the most critical factor which influences the system performance on serving requests for temporal data items. Figure 5 shows the service ratio of each algorithm under different data update periods. A large range of data update periods are examined for comprehensive performance evaluation. When the data update period is small (i.e. $50 \sim 100$ broadcast ticks), it implies that the timing is stringent on satisfying the dynamic snapshot consistency requirement. In other words, the dynamic snapshot consistency requirement dominates the algorithm performance in such an environment. As shown, both SSCS and DSCS have much higher service ratios than other algorithms. Specifically, SSCS makes an initial effort on satisfying the dynamic snapshot consistency requirement by restricting the service for a request in a single snapshot. On this basis, DSCS adopts the reschedule mechanism for serving requests with more viable snapshots. As observed, the shorter of the data update period is, the more significant advantage is achieved by DSCS, which demonstrates the its superiority on satisfying stringent consistency requirements. On the other hand, when the data update period is getting longer, the performance of all the algorithms gradually converge at a certain service ratio, where the scheduling performance is dominated by the request deadline. As shown, DSCS achieves the

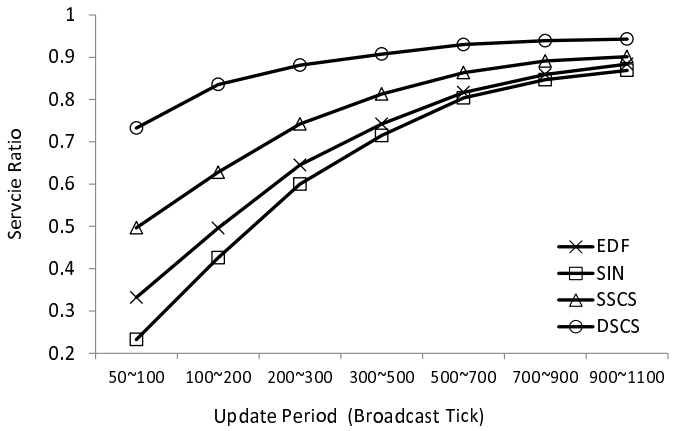

Fig. 5. Service ratio under different data update periods

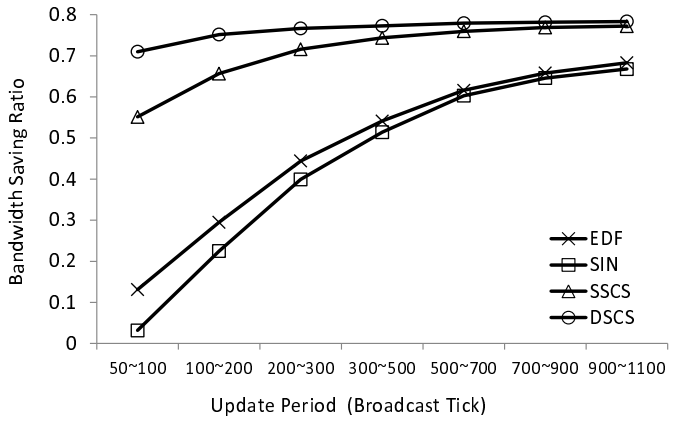

Fig. 6. Bandwidth saving ratio under different data update periods

best performance across the whole range. Similarly, Figure 6 shows the bandwidth saving ratio of each algorithm under different data update periods. The observations are consistent with that in Figure 5. This set of simulation results conclusively demonstrates that no matter whether the consistency requirement or the request deadline dominates the scheduling performance, DSCS always outperforms other algorithms.

6.2.3. Effect of system workload. In order to show the scalability of the system, we examine the scheduling performance of algorithms under different system workloads. In the default setting, the service rate is 1 (i.e. the system broadcasts one data item in each broadcast tick) and the data request rate is 5, which is computed by $\lambda \cdot s$ (i.e. five data items are requested in each broadcast tick on average). Therefore, the default system workload, which is the ratio of the data request rate to the service rate, equals 5. The following experiments examine the scheduling performance of algorithms under different system workloads by adjusting the request arrival rate (λ). As shown in Figure 7, when the data request rate is twice of the service rate (i.e. the workload is 2), both SSCS and DSCS can serve nearly 100% of requests under the default setting. When the system workload increases, although the performance of all the algorithms drops to varying degrees, DSCS maintains a decent service ratio, which is over 80%. Figure 8 shows the bandwidth saving ratio of each algorithm under different system workloads. Normally, in a heavier system workload environment, broadcasting a

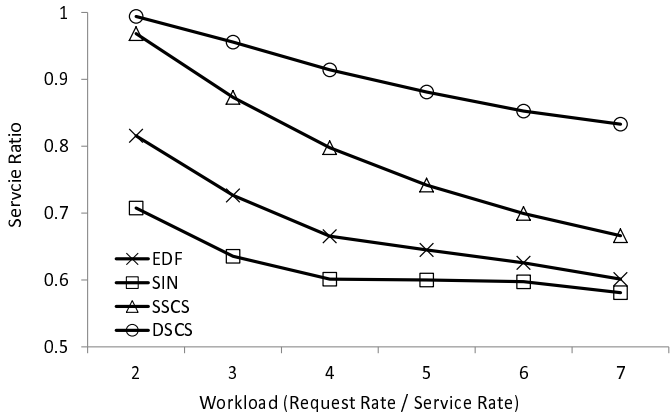

Fig. 7. Service ratio under different system workloads

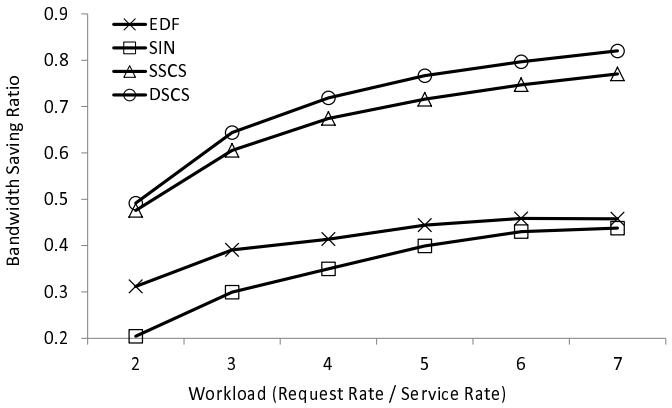

Fig. 8. Bandwidth saving ratio under different system workloads

data item has the potential to be retrieved by more requests, because it is likely that there are more pending requests for a particular data item. As shown in Figure 8, both SSCS and DSCS achieve a higher bandwidth utilization ratio with an increasing of the request arrival rate, which indicates that they can effectively exploit the broadcast effect. In contrast, the bandwidth saving ratio of SIN and EDF almost remain the same, even though the request arrival rate is getting higher, which indicates the inefficiency of bandwidth utilization.

7. CONCLUSIONS AND FUTURE WORK

Efficient data dissemination is one of the fundamental requirements to enable a variety of innovative applications in VCPS. In this work, we analyze the system characteristics of data dissemination via I2V communication, including the timeliness of data dissemination, the temporality of data items and the dependency of requested data items. On this basis, we formulate the static and the dynamic snapshot consistency requirements for serving real-time requests asking for multiple dependent temporal data items. We propose an algorithm called SSCS, which analyzes the time bound for serving requests based on the static snapshot consistency requirement. It considers the request characteristics including the productivity, serving status and urgency in scheduling with the purpose of exploiting the broadcast effect, improving the bandwidth utilization and enhancing the request service chance. The other algorithm called DSCS is proposed for scheduling requests with the dy-

namic snapshot consistency requirement. In particular, we design a reschedule mechanism, which is adopted to update the consistency requirement for those requests which cannot be served by SSCS and recompute the time bound for serving these requests. Therefore, DSCS enhances the chance of serving requests with the dynamic snapshot consistency requirement. Finally, we build the simulation model for performance evaluation. The experimental results under a variety of circumstances demonstrate the superiority of the proposed algorithms.

This work focuses on data dissemination via I2V communication with the presence of a single RSU. It is desirable to scale the system by considering the cooperation of multiple RSUs. In addition, leveraging V2V communication to assist and supplement the I2V-based data dissemination is a promising approach to enhancing the robustness and increasing the scalability of the system, which also deserves future efforts.

REFERENCES

2013. MIT CarTel. [Online]: http://cartel.csail.mit.edu/doku.php. (2013).

2013. USDOT - Research and Innovative Technology Administration (RITA): Connected vehicle research. [Online]: http://www.its.dot.gov/connected_vehicle/connected_vehicle.htm. (2013).

2013. USDOT - Research and Innovative Technology Administration (RITA): Vehicular infrastructure integration. [Online]: http://www.its.dot.gov/vii. (2013).

Swarup Acharya and S Muthukrishnan. 1998. Scheduling on-demand broadcasts: New metrics and algorithms. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM'98)*. ACM, 43–54.

Demet Aksoy and Michael Franklin. 1999. R×W: a scheduling approach for large-scale on-demand data broadcast. *IEEE/ACM Transactions on Networking* 7, 6 (1999), 846–860.

Fan Bai and Hariharan Krishnan. 2006. Reliability analysis of DSRC wireless communication for vehicle safety applications. In *Proceedings of the 9th International IEEE Conference on Intelligent Transportation Systems (ITSC'06)*. IEEE, 355–362.

Fan Bai, Daniel D Stancil, and Hariharan Krishnan. 2010. Toward understanding characteristics of dedicated short range communications (DSRC) from a perspective of vehicular network engineers. In *Proceedings of the 16th Annual International Conference on Mobile Computing and Networking (MobiCom'10)*. ACM, 329–340.

Jun Chen, Victor Lee, Kai Liu, GG Md Nawaz Ali, and Edward Chan. 2013. Efficient processing of requests with network coding in on-demand data broadcast environments. *Information Sciences* (2013).

Gurcan Comert and Mecit Cetin. 2011. Analytical evaluation of the error in queue length estimation at traffic signals from probe vehicle data. *Intelligent Transportation Systems, IEEE Transactions on* 12, 2 (2011), 563–573.

Yaser P Fallah, Ching-Ling Huang, Raja Sengupta, and Hariharan Krishnan. 2011. Analysis of information dissemination in vehicular ad-hoc networks with application to cooperative vehicle safety systems. *IEEE Transactions on Vehicular Technology* 60, 1 (2011), 233–247.

FCC. 2006. FCC report and Order 06-110. Amendment of the commission's rules regarding dedicated short-range communication services in the 5.850-5.925GHz band. (2006).

Kaichi Fujimura and Takaaki Hasegawa. 2004. A collaborative MAC protocol for inter-vehicle and road to vehicle communications. In *Proceedings of the 7th International IEEE Conference on Intelligent Transportation Systems (ITSC'04)*. IEEE, 816–821.

Stefan K Gehrig and Fridtjof J Stein. 2007. Collision avoidance for vehicle-following systems. *IEEE Transactions on Intelligent Transportation Systems* 8, 2 (2007), 233–244.

Chih-Lin Hu and Ming-Syan Chen. 2009. Online scheduling sequential objects with periodicity for dynamic information dissemination. *IEEE Transactions on Knowledge and Data Engineering* 21, 2 (2009), 273–286.

IEEE. August 2010. IEEE Standard for Wireless Access in Vehicular Environments (WAVE)-Multi-channel Operation. http://www.sae.org/standardsdev/dsrc. (August 2010).

ITS-Berkeley. 2013. PATH: Partners for advanced transportation technology. [Online]: http://www.path.berkeley.edu/. (2013).

Ming-Fong Jhang and Wanjiun Liao. 2010. Cooperative and opportunistic channel access for vehicle to roadside (V2R) communications. *Mobile Networks and Applications* 15, 1 (2010), 13–19.

Kam-Yiu Lam, Edward Chan, and Joe Chun-Hung Yuen. 2000. Approaches for broadcasting temporal data in mobile computing systems. *Journal of Systems and Software* 51, 3 (2000), 175–189.

Joyoung Lee and Byungkyu Park. 2012. Development and evaluation of a cooperative vehicle intersection control algorithm under the connected vehicles environment. *IEEE Transactions on Intelligent Transportation Systems* 13, 1 (2012), 81–90.

Kai Liu, , Victor Lee, Joseph Ng, and Sang Son. 2013. Scheduling temporal data for real-time requests in roadside-to-vehicle communication. In *Proceedings of the 19th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'13)*. IEEE.

Kai Liu, Edward Chan, Victor Lee, Krasimira Kapitanova, and Sang H Son. 2013. Design and evaluation of token-based reservation for a roadway system. *Transportation Research Part C: Emerging Technologies* 26 (2013), 184–202.

Kai Liu and Victor Lee. 2010a. On-demand broadcast for multiple-item requests in a multiple-channel environment. *Information Sciences* 180, 22 (2010), 4336–4352.

Kai Liu and Victor Lee. 2012. Adaptive data dissemination for time-constrained messages in dynamic vehicular networks. *Transportation research part C: emerging technologies* 21, 1 (2012), 214–229.

Kai Liu and Victor CS Lee. 2010b. RSU-based real-time data access in dynamic vehicular networks. In *Proceedings of the 13th International IEEE Conference on Intelligent Transportation Systems (ITSC'10)*. IEEE, 1051–1056.

Osamu Maeshima, Shengwei Cai, Teruhiko Honda, and Hirofumi Urayama. 2007. A roadside-to-vehicle communication system for vehicle safety using dual frequency channels. In *Proceedings of the 10th International IEEE Conference on Intelligent Transportation Systems (ITSC'07)*. IEEE, 349–354.

Tony K Mak, Kenneth P Laberteaux, and Raja Sengupta. 2005. A multi-channel VANET providing concurrent safety and commercial services. In *Proceedings of the 2nd ACM International Workshop on Vehicular Ad Hoc Networks (VANET '05)*. ACM, 1–9.

Adelin Miloslavov and Malathi Veeraraghavan. 2012. Sensor data fusion algorithms for vehicular cyber-physical systems. *IEEE Transactions on Parallel and Distributed Systems* 23, 9 (2012), 1762–1774.

Yasser L Morgan. 2010. Notes on DSRC & WAVE standards suite: Its architecture, design, and characteristics. *IEEE Communications Surveys & Tutorials* 12, 4 (2010), 504–518.

H. Schwetman. 2001. CSIM19: a powerful tool for building system models. In *Proceedings of the 33nd Conference on Winter Simulation (WSC'01)*. IEEE, 250–255.

JW Wong and Mostafa H. Ammar. 1985. Analysis of broadcast delivery in a videotex system. *IEEE Trans. Comput.* 100, 9 (1985), 863–866.

John W Wong. 1988. Broadcast delivery. *Proc. IEEE* 76, 12 (1988), 1566–1577.

Jianliang Xu, Xueyan Tang, and Wang-Chien Lee. 2006. Time-critical on-demand data broadcast: Algorithms, analysis, and performance evaluation. *IEEE Transactions on Parallel and Distributed Systems* 17, 1 (2006), 3–14.

Ping Xuan, Subhabrata Sen, Oscar Gonzalez, Jesus Fernandez, and Krithi Ramamritham. 1997. Broadcast on demand: Efficient and timely dissemination of data in mobile environments. In *Proceedings of the 3rd IEEE Real-Time Technology and Applications Symposium (RTAS'97)*. IEEE, 38–48.

G.K. Zipf. 1949. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Cambridge, Mass.: Addison-Wesley Press.