Speed Control and Scheduling of Data Mules in Sensor Networks

RYO SUGIHARA and RAJESH K. GUPTA
University of California, San Diego

Unlike traditional multihop forwarding among stationary sensor nodes, use of mobile devices for data collection in wireless sensor networks has recently been gathering more attention. The use of mobility significantly reduces the energy consumption at sensor nodes, elongating the functional lifetime of the network. However, a drawback is an increased data delivery latency. Reducing the latency through optimizing the motion of data mules is critical for this approach to thrive. In this article, we focus on the problem of motion planning, specifically, determination of the speed of the data mule and the scheduling of the communication tasks with the sensors. We consider three models of mobility capability of the data mule to accommodate different types of vehicles. Under each mobility model, we design optimal and heuristic algorithms for different problems: single data mule case, single data mule with periodic data generation case, and multiple data mules case. We compare the performance of the heuristic algorithm with a naive algorithm and also with the multihop forwarding approach by numerical experiments. We also compare one of the optimal algorithms with a previously proposed method to see how our algorithm improves the performance and is also useful in practice. As far as we know, this study is the first of a kind that provides a systematic understanding of the motion planning problem of data mules.

Categories and Subject Descriptors: C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Wireless communication*

General Terms: Algorithm, Design, Performance

Additional Key Words and Phrases: Controlled mobility, motion planning, scheduling, linear programming, simulation

ACM Reference Format:

Sugihara, R. and Gupta, R. K. 2010. Speed control and scheduling of data mules in sensor networks. ACM Trans. Sensor Netw. 7, 1, Article 4 (August 2010), 29 pages.
DOI = 10.1145/1806895.1806899 http://doi.acm.org/10.1145/1806895.1806899

This work has been partially supported by the DMEA research program under contract H94003-07-C-0702 and the US National Science Foundation under grants CCF-0702792 and SRS-0820034.
Authors' addresses: Department of Computer Science and Engineering, University of California, San Diego, 9500 Gilman Drive, La Jolla, CA 92093; email: {ryo,rgupta}@ucsd.edu.

DOI 10.1145/1806895.1806899 http://doi.acm.org/10.1145/1806895.1806899

1. INTRODUCTION

Controlled mobility presents an attractive alternative to multihop forwarding for efficient data collection in a sensor field. In particular, we consider collecting data from stationary sensor nodes using "data mules" via wireless communication. A data mule is a mobile node with radio and sufficient amount of storage to store the data from the sensors in the field. Data mules have been used in recent sensor network applications, e.g., a robot in underwater environmental monitoring [Vasilescu et al. 2005] and a UAV (unmanned aerial vehicle) in structural health monitoring [Todd et al. 2007]. A data mule travels across the sensor field and collects data from each sensor node while the distance is short, and later deposits all the data to the base station. In this way, each sensor node can conserve energy, since it only needs to send the data over a short distance and has no need to forward other sensors' data all the way to the base station. Note that energy issue is critical for sensor nodes as opposed to the data mule that returns to the base station after the travel. However, one disadvantage of this approach is that it generally takes more time to collect data, which in turn incurs larger data delivery latency. Thus optimizing the data delivery latency is vital for the data mule approach to be useful in practice.

Optimizing the motion of data mule is a hard problem in general, since we need to find a trajectory satisfying both spatial constraints (i.e., wireless communication range) and temporal constraints (i.e., time required for data collection), where the spatial constraints alone is analogous to traveling salesman problem. To deal with the hardness, previous studies on data mule approaches [Kansal et al. 2004; Somasundara et al. 2004; Ma and Yang 2006; Xing et al. 2007] simplified the problem by using simple models for mobility and communications. These simplifications lead to suboptimal solutions with unnecessarily large latency. Moreover, since these algorithms are often tailored for specific settings, it is difficult to compare them and use them for similar problems in different application scenarios.

To address these limitations, in this article we describe a problem framework that we call the *data mule scheduling* (DMS) *problem*. The DMS problem captures the motion planning problem as one composed of loosely connected subproblems of path selection, speed control, and job scheduling. This framework enables us to identify the cases of the problem that we can optimally solve, as well as the hard cases that we explore heuristically.

Our focus in this article is on the one-dimensional case of the DMS problem (1-D DMS). The 1-D DMS problem is to determine the speed change and data collection schedule to minimize the data delivery latency. This applies to many cases in which the data mules move on the fixed paths. We first discuss the basic case, where a single data mule collects a given amount of data from each sensor node. We show we can give an optimal speed change and data collection schedule when the data mule's mobility model is either constant speed or variable speed. For a case when there is an acceleration constraint, we design a heuristic algorithm. Later we discuss the cases of periodic data generation and multiple data mules. In the periodic data generation case, each sensor node generates

data at a certain rate and the data mule travels repetitively. In the multiple data mules case, data mules move either along an identical path or arbitrary paths. For both of these cases, we design optimal and heuristic algorithms in a similar way as the basic case. By numerical experiments, the performance of the proposed algorithms is evaluated against multihop forwarding approach and a previously proposed speed control algorithm.

Our contributions include the following:

—Formulate the data mule scheduling (DMS) problem, a problem framework for optimal control of data mule for minimizing the data delivery latency;

—Present optimal and heuristic algorithms for the one-dimensional case of the DMS problem (1-D DMS) in various problem settings including single data mule, multiple data mules, and periodic data generation, each for three different mobility models.

The rest of this article is organized as follows. In Section 2 we give an overview of the DMS problem framework. We also introduce an application example and related work. In Section 3 we define the 1-D DMS problem in more details and introduce mobility models we consider. We also discuss the feasibility testing by processor demand analysis that we use in designing the optimal algorithms. We present optimal and heuristic algorithms for the basic cases of single data mule in Section 4. A periodic case with single data mule is discussed in Section 5. In Section 6, we discuss multiple data mules cases. Section 7 shows the results of the simulation experiments and Section 8 concludes the article.

2. PROBLEM FRAMEWORK FOR MOTION PLANNING OF DATA MULE

In this section we present the data mule scheduling (DMS) framework for optimizing the motion of data mule. As we discussed in the introduction, motion planning of data mule is a hard problem. Communications with sensor nodes need to take place in the proximity of each node and will take certain time duration, whereas the motion of data mule is possibly governed by dynamics constraints. There is also a prioritization problem when the data mule is in the communication ranges of multiple nodes.

To deal with this complexity, in the DMS framework, we decompose the problem into the following three subproblems as shown in Figure 1:

(1) Path selection: which trajectory the data mule follows;
(2) Speed control: how the data mule changes the speed while moving along the path;
(3) Job scheduling: from which sensor the data mule collects data at each time point.

Path selection is to determine the trajectory of the data mule in the sensor field. To collect data from each particular sensor, the data mule needs to enter the sensor's communication range at least once. Depending on the capability of data mule, there may be some constraints on path selection, such as minimum turning radius.

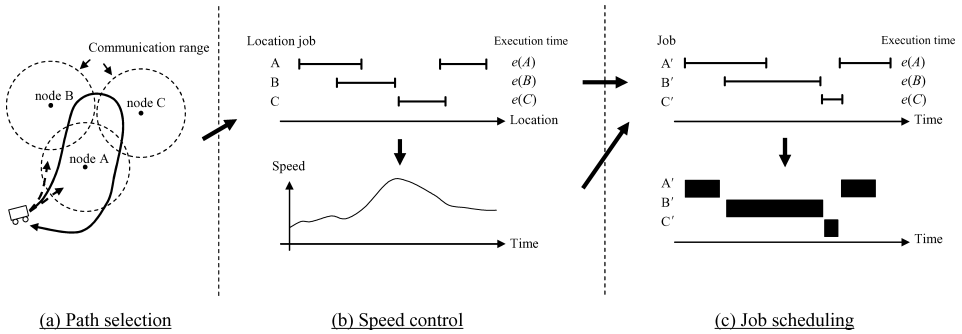

Fig. 1. Subproblems of data mule scheduling.

Speed control is the second subproblem to determine how the data mule changes its speed along the chosen path. The data mule needs to change speed so that it stays within each sensor's communication range long enough to collect all the data from it.

The final subproblem is job scheduling. Once the time-speed profile is determined, we get a mapping from each location to a time point. Thus we get a scheduling problem by regarding data collection from each sensor as a job. Each job has one or more intervals in which it can be executed. Job scheduling is to determine the allocation of time to jobs so that all jobs can be completed.

The DMS problem is general and can be used to express several earlier problems in the area. For instance, the assumption of no wireless communication (as in [Somasundara et al. 2004, 2007]) is easily expressed by setting the communication range to zero in the path selection subproblem. The constant-speed assumption (as in [Ma and Yang 2006, 2007; Xing et al. 2007]) and variable-speed assumption (as in [Zhao and Ammar 2003; Kansal et al. 2004]) are handled in the speed control subproblem.

In this article, we focus on the subproblems of speed control and job scheduling. These subproblems constitute the one-dimensional case of the data mule scheduling problem (1-D DMS). The 1-D DMS problem is important in many cases when the data mule needs to move along a given path.

A solution for the 1-D DMS problem is twofold. One element is a "time-speed profile," which determines the speed changes of the data mule. With a time-speed profile, each point on the location axis can be mapped onto a point on the time axis. Then we obtain a scheduling problem having a set of jobs, each of which has an execution time and feasible intervals. For this problem, we need to determine the "job schedule" that defines when the data mule communicates with each node.

The objective of the 1-D DMS problem is to find a speed control plan and a feasible job schedule so that the total travel time of the data mule is minimized.

2.1 Why Do We Minimize the Latency?

As mentioned earlier, data mules can be used as an alternative to multi-hop forwarding in sensor networks. The use of data mules in collecting data

introduces the tradeoff between energy consumption and data delivery latency. Our goal is to optimize this tradeoff through either minimizing the energy consumption under some latency constraint or minimizing the latency under some constraint on energy consumption.

Protocol designers have tried to optimize the multihop forwarding in both energy and latency through sophisticated MAC protocols [Ye et al. 2002; Polastre et al. 2004; Rhee et al. 2005]. Data mule, or its combination with multihop forwarding, is a relatively nascent area. In this article, we focus on the pure data mule approach, in which each node uses only direct communication with the data mule and no multihop forwarding. Energy consumption related to communication is already minimized in this case, since each node only sends its own data and does not forward others' data. Naturally, our objective is to minimize the data delivery latency by minimizing the travel time of the data mule.

2.2 Example Application: SHM with UAV

Our problem formulation is based on our experience with the example application described in Todd et al. [2007]. It is a structural health monitoring (SHM) application to do postevent (e.g., earthquakes) assessments for large-scale civil infrastructure such as bridges. Automated damage assessment using sensor systems is much more efficient and reliable than human visual inspections.

In this application, the sensor nodes operate completely passively and do not carry batteries, for the sake of long-term measurement and higher maintainability. Upon data collection, an external mobile element provides energy to each node via microwave transmission, wakes it up, and collects data from it. The prototype system uses a radio-controlled helicopter as the mobile element that is either remotely piloted or GPS-programmed. Each sensor node is equipped with ATmega128L microcontroller, a 2.4GHz XBee radio, antennas for transmission/reception, and a supercapacitor to store the energy. Each node has two types of sensors. One is a piezoelectric sensing element integrated with nuts and washers to check if the bolt has loosened. The other is capacitive-based sensors for measuring peak displacement and bolt preload. Since the size of data from these sensors is small, communication time is almost negligible; however, it takes a few minutes to charge a supercapacitor through microwave transmission in the current prototype. The team is currently investigating a new design to improve the charging time down to tens of seconds.

The data collected by the UAV is brought back to the base station and analyzed by researchers using statistical techniques for damage existence and its location/type. Since the primary purpose of this application is to assess the safety of large civil structures after a disaster such as an earthquake, every process including data collection and analysis needs to proceed as quickly as possible for prompt recovery. Furthermore, a shorter travel time is required in view of the limited fuel on the helicopter.

Thus the goal of our formulation is to achieve data collection from spatially distributed wireless sensors in the minimum amount of time. It also provides another reason for using the data mule approach instead of multihop

forwarding: simply because the SHM sensors are not capable of doing multihop communication. Furthermore, use of UAVs implies the need for more precise mobility model that takes the acceleration constraint into consideration, as opposed to the simple "move or stop" model used in the majority of the related work.

2.3 Related Work

Data mule approaches, or using controlled mobility for improving data collection performance in sensor or mobile ad hoc networks, have been discussed in various articles from both theoretical and applications perspectives [Zhao and Ammar 2003; Ho and Fall 2004; Vasilescu et al. 2005; Ma and Yang 2006; Somasundara et al. 2006, 2007; Xing et al. 2008; Chebrolu et al. 2008]. For an overview of data mule and similar approaches, see Ekici et al. [2006]. Our work on the DMS problem has been focused on identifying the combinatoric nature of the problem among its various variants in an attempt to develop a comprehensive understanding of the fundamental performance limits of the data mule approach.

Ma and Yang [2006, 2007] studied the problem of maximizing the network lifetime, which is defined as the time until the first node dies (i.e. the minimum of the lifetime of all nodes), by finding an optimal motion of a data mule. Speed control was not their focus and they just used a constant-speed mobility model. They also avoided the problem of scheduling by assuming either that data communication time is negligible (in Ma and Yang [2006]) or that the data mule stops while communicating with sensor nodes (in Ma and Yang [2007]). However, the former assumption does not apply when the data size is big and communication is slow, and the latter results in inefficiency when the data mule can actually communicate while it is moving.

Zhao and Ammar [2003] studied the use of controlled mobility in mobile ad hoc networks. They assumed a controllable mobile node (called a *ferry*) that mediates communications between sparsely deployed stationary nodes. The speed of the ferry is basically constant but can be reduced when it is necessary to communicate more data with a node. They formulated the problem as a linear programming problem. Our work is similar and we build our formulations upon theirs, but with extensions that include more general mobility models, as well as other scenarios including multiple data mules and periodic motions.

A more sophisticated speed controlling scheme has been proposed by Kansal et al. [2004] and Somasundara et al. [2006]. Assuming a data mule periodically travels across the sensor field along a fixed path, they proposed an adaptive speed control algorithm. In the algorithm, the data mule slows down when it encounters the nodes from which data collection has not been very successful in the previous period. Their speed control algorithm is reactive in the sense it is solely based on past performance. By contrast, we study a proactive approach by having a priori information on the communication range and data collection time.

For multiple data mules case, Jea et al. [2005] studied the case in which multiple data mules move on fixed paths. They presented a distributed

coordination scheme for allocating sensor nodes to each data mule to achieve a good distribution of the communications load. They just assumed movements at the given constant speed. Our work is complementary to their work and focused on identifying the performance limits of centralized schemes when we can freely determine the path and speed of each data mule. This would serve as a performance target for distributed schemes.

Our formulation of the DMS problem is closely related to scheduling. It is known that the earliest due date (EDD) rule, in which the earliest deadline among all available jobs is executed at any time slice, is optimal for single processor preemptive scheduling with release times [Stankovic et al. 1995]. On the other hand, our problem is not standard in that each job may have multiple feasible intervals. As far as we know, there are only a few studies of this case: for unitlength, non-preemptible jobs [Simons and Sipser 1984] and for preemptible, noncontinuable (i.e., jobs must be completed within one feasible interval) jobs [Shih et al. 2003; Chen et al. 2005], both of which are different from our problem of interest.

3. 1-D DMS PROBLEM

In this section, we define the 1-D DMS problem in a more formal way. We first introduce some terms and definitions and then give a formal definition. We also discuss the mobility models of data mules.

3.1 Terminology, Definitions, and Assumptions

For the job scheduling subproblem, a *job* τ_i has an execution time e_i and a set $\mathcal{I}_i$ of feasible intervals. A *feasible interval* $I \in \mathcal{I}_i$ is a time interval $[r(I), d(I)]$, where $r(I)$ is a *release time* and $d(I)$ is a *deadline*. A job can be executed only within its feasible intervals. A *simple job* is a job with one feasible interval, whereas a *general job* can have multiple feasible intervals. For instance, in Figure 1(c), job B′ and C′ are simple jobs and job A′ is a general job.

Similarly for the speed control subproblem, a *location job* τ_i has an execution time e_i and a set $\mathcal{I}_i$ of feasible location intervals. A *feasible location interval* $I \in \mathcal{I}_i$ is a location interval $[r(I), d(I)]$, where $r(I)$ is a *release location* and $d(I)$ is a *deadline location*. A location job can be executed only within its feasible location intervals. A *simple location job* is a location job with one feasible location interval, whereas a *general location job* can have multiple feasible location intervals. In Figure 1(b), location job B and C are simple location jobs and location job A is a general location job.

For an interval $I = [r, d]$ (also for a location interval), $|I|$ denotes the length $d - r$. We also define containment as follows: $I \subseteq I'$ if and only if $r' \leq r$ and $d \leq d'$ where $I' = [r', d']$.

We make the following assumptions. Each sensor node is stationary and the location is known. Communication range and execution time are known. Communication is always successful in the communication range. All location jobs are preemptible without any cost incurred and can be executed over multiple feasible location intervals. There is no dependency among the location jobs. Data mule can communicate with one node at a time. Depending on the

dynamics constraint, the data mule may have constraints on its maximum speed and maximum acceleration.

3.2 Mobility Models

In the speed control subproblem, we consider three different constraints on the dynamics of the data mule.

The first is "constant speed," where the data mule cannot change speed after it starts to move, that is, $v(t) = v_0$ for some constant v_0.

The second is "variable speed," where the data mule can instantaneously change speed within a speed range. that is, $v_{min} \leq v(t) \leq v_{max}$ for given constants v_{min}, v_{max}.

The third is the "variable-speed with acceleration-constraint" model. In this model, the data mule can change the speed, but the rate of change is within the maximum absolute acceleration, that is, $|dv(t)/dt| \leq a_{max}$ for given constant a_{max}.

The constant-speed and variable-speed models apply to ground vehicles such as Packbot,[1] which is commonly used as a data mule in actual deployments. The acceleration-constrained model captures mobility more precisely and is most appropriate when we cannot ignore the inertia, for example in the case where a helicopter is used as a data mule, as in Todd et al. [2007].

3.3 Problem Definition

An instance of the 1-D DMS problem is $(L, \mathcal{J})$, where

—$[0, L]$: total travel interval of the data mule on the location axis;
—$\mathcal{J}$: set of location jobs; i-th location job τ_i is characterized by;
  —$\mathcal{I}_i$: set of feasible location intervals;
  —e_i: execution time.

The solution to the problem is a pairing of time-speed profile $v(t)$ and job schedule. Let T denote the total travel time. Then the constraints on the motion of the data mule are $\int_0^T v(t) = L$ and the dynamic constraints are related to each mobility model. After $v(t)$ is determined, we can define a function $f(x)$ that maps location x to time t. Using $f(x)$, we obtain a job scheduling problem, which we call an *induced job scheduling problem*. We need to determine $v(t)$ such that the induced job scheduling problem has a "feasible schedule": a job schedule that finishes all the jobs before their deadlines. The objective of the 1-D DMS problem is to find a solution that minimizes T.

4. ALGORITHMS FOR BASIC CASES

In this section, we consider the 1-D DMS problem for the single data mule case. We first introduce a feasibility test by processor demand analysis. Using that, we design optimal algorithms for the constant-speed case and variable-speed case. Then we present the idea of a heuristic algorithm for the acceleration-constrained case.

[1] `http://www.irobot.com/`.

Algorithm 1. FIND-MIN-MAXSPEED.

1: **for** each location interval $I = [r(\tau'), d(\tau'')]$ s.t. $\tau', \tau'' \in \mathcal{J}, r(\tau') \leq d(\tau'')$ **do**
2: $\quad d = \sum_{\substack{\tau \in \mathcal{J}, \\ \mathcal{I}(\tau) \in I}} e(\tau)$ ▷ Processor demand for I
3: $\quad u[I] \leftarrow \frac{|I|}{d}$ ▷ Maximum speed allowed for I
4: **end for**
5: **return** $\min_I u[I]$

4.1 Preliminary: Feasibility Testing by Processor Demand Analysis

To design optimal algorithms for the 1-D DMS problem, we use processor-demand-based feasibility testing by Baruah et al. [1993]. Processor demand g in interval $[t_1, t_2]$ is the sum of the execution time of the tasks whose feasible interval is completely contained in the interval, and is defined as

$$g(t_1, t_2) = \sum_{\substack{\tau \in \mathcal{J} \\ I(\tau) \in [t_1, t_2]}} e(\tau).$$

Then the following theorem holds for periodic tasks with arbitrary relative deadline (i.e., the relative deadline of each task can be smaller than its period):

THEOREM 4.1 [BARUAH ET AL. 1993]. *Let $\tau = \{T_1, \ldots, T_n\}$ be a task system. τ is not feasible iff there exist natural numbers $t_1 < t_2$ such that $g(t_1, t_2) > t_2 - t_1$.*

Using the theorem, we can show that testing at each release time and deadline is sufficient for guaranteeing the feasibility. In other words:

THEOREM 4.2. *The task system is feasible iff $g(t_1', t_2') \leq t_2' - t_1'$ for any $t_1' \in \{r_i\}, t_2' \in \{d_i\}$ satisfying $t_1' < t_2'$.*

The proof of Theorem 4.2 is given in the Appendix. This feasibility testing is for a periodic task system in real time scheduling, but we can apply it to our case. Specifically, we can determine the speed for each location interval so that the processor demand for the interval is at most the time that the data mule stays in the interval.

4.2 Constant Speed

For the constant-speed case, the problem is to find the maximum speed v_0 such that all jobs can be finished. We present two optimal offline scheduling algorithms, for simple location jobs and general location jobs, respectively.

4.2.1 *Simple Location Jobs.* When each location job has one feasible location interval, the simple algorithm shown as Algorithm 1 finds the maximum possible v_0 such that all location jobs can be completed. It applies the processor-demand-based feasibility test (Theorem 4.2) for all possible pairs of a release location and a deadline location.

This algorithm runs in $O(n^3)$ time, where n is the number of location jobs in a naive implementation, but we can improve it to $O(n^2)$ by computing the

processor demand incrementally. Specifically, for each starting location, by having a list of jobs sorted by their deadline locations, we can incrementally extend the interval and calculate the processor demand in $O(1)$ time. Then it takes $O(n)$ time for each starting location, and since there are at most n starting locations, it takes $O(n^2)$ as a whole.

4.2.2 *General Location Jobs.* For the general location jobs case, we formulate the problem as a linear programming problem.

We split the total travel interval $[0, L]$ into $(2m + 1)$ location intervals $[l_0(= 0), l_1]$, $[l_1, l_2], \ldots, [l_{2m}, l_{2m+1}(= L)]$ $(l_i \leq l_{i+1})$, where m is the number of feasible location intervals of all location jobs, and each l_i is either a release location or a deadline location. For each location job $\tau \in \mathcal{J}$, we consider variables $p_0(\tau), \ldots, p_{2m}(\tau)$, in which $p_i(\tau)$ represents the time allocated to job τ within the location interval $[l_i, l_{i+1}]$. Let v_0 denote the speed of data mule. Then we have the following linear program:

Maximize v_0 subject to

—$0 \leq \forall i \leq 2m, \forall \tau \in \mathcal{J}, p_i(\tau) \geq 0$.
—(Feasible intervals) $0 \leq \forall i \leq 2m, \forall \tau \in \mathcal{J}$, if $[l_i, l_{i+1}] \notin \mathcal{I}(\tau)$, $p_i(\tau) = 0$.
—(Job completion) $\forall \tau \in \mathcal{J}, \sum_{i=0}^{2m} p_i(\tau) = e(\tau)$.
—(Processor demand) $0 \leq \forall i \leq 2m, \sum_{\tau \in \mathcal{J}} p_i(\tau) \leq (l_{i+1} - l_i)/v_0$.

The processor demand constraint becomes a linear constraint by introducing a new variable $u_0 = 1/v_0$.

For the solution obtained, we can make a job schedule in the following way. Location interval $[l_i, l_{i+1}]$ is mapped to the time interval $[\sum_{k=0}^{i-1} z_k, \sum_{k=0}^{i} z_k]$, where $z_k = (l_{k+1} - l_k)/v_0$. For each time interval, we allocate $p_i(\tau_1)$ for location job τ_1 from the start of the interval and $p_i(\tau_2)$ for τ_2 after that, and continue this for all location jobs.

4.3 Variable Speed

In the variable-speed case, the data mule can change its speed anytime. To make the problem realistic,[2] we enforce constraints on the speed for this case, and the data mule can choose its speed within the range $[v_{min}, v_{max}]$ for a given v_{min} and v_{max}.

4.3.1 *Simple Location Jobs.* For the simple location jobs case, we have the optimal offline algorithm shown as Algorithm 2. The algorithm is based on processor demand analysis and uses FIND-MIN-MAXSPEED internally. Interestingly, this algorithm is equivalent to the one for optimal dynamic voltage scaling presented in Yao et al. [1995].

Here is how this algorithm works. For the set of location jobs, FIND-MIN-MAXSPEED finds a tight interval and the corresponding speed v_c. It is the minimum speed that makes the processor demand for each location interval equal to or less than the time allocated to it. In other words, if the data mule

[2]Without speed constraints, the data mule can always minimize the total travel time simply by moving at infinite speed and stopping to execute a job (and repeat this for each job).

Algorithm 2. SEQUENTIAL-FIND-MIN-MAXSPEED.

```
1: loop
2:     v_c ← FIND-MIN-MAXSPEED
3:     if v_c < v_min then return INFEASIBLE
4:     else if v_c > v_max then
5:         Set v_max for all remaining intervals and finish
6:     else
7:         Set v_c for the current tight interval
8:         Remove jobs within the tight interval
9:         "Compress" remaining jobs
10:    end if
11: end loop
```

moves at the speed faster than v_c, there is at least one interval in which the allotted time is less than the processor demand (and thus violates the feasibility). Therefore, if $v_c < v_{min}$ (Line 3), it is infeasible. "Compress" (Line 9) is an operation used in Yao et al. [1995], which is to remove the tight interval from the feasible intervals of all jobs and to connect the remaining two intervals together to construct a new set of jobs. In each iteration, v_c is nondecreasing. This was shown by the same reasoning in Yao et al. [1995]. When v_c reaches v_{max}, we cannot increase the speed of any remaining location intervals. Thus we set the speed to v_{max} for all these intervals (Line 5).

The optimality of this algorithm is shown immediately by using the proof in Yao et al. [1995]. This algorithm runs in $O(n^3)$ time, since each iteration takes $O(n^2)$ time by using the improved implementation of FIND-MIN-MAXSPEED and at least one job is removed at each iteration.

As for online algorithms, when $v_{min} = 0$, the algorithm shown as Algorithm 3 is an optimal online algorithm that minimizes the total travel time. In the algorithm, EDD-WITH-STOP, the data mule moves at v_{max} while executing a job having the earliest deadline, in the same way as the ordinary Earliest Due Date algorithm. However, when a job is not completed by its deadline, the data mule stops until the job is completed and moves at v_{max} again.

We have the following theorem about the optimality of EDD-WITH-STOP. Its proof is given in the Appendix.

THEOREM 4.3. EDD-WITH-STOP *is optimal for the 1-D DMS problem for simple location jobs with the variable-speed model when* $v_{min} = 0$

4.3.2 *General Location Jobs.* For general location jobs, we design an offline algorithm by linear programming formulation. The formulation is similar to the one in the constant-speed case (Section 4.2.2), but we have additional variable z_i for each location interval $[l_i, l_{i+1}]$ to represent the time that the data mule spends in the interval. The linear program is as follows:

Minimize $\sum_{i=0}^{2m} z_i$ subject to

—$0 \le \forall i \le 2m, \forall \tau \in \mathcal{J}$, $p_i(\tau) \ge 0$, $z_i \ge 0$.

—(Feasible intervals) $0 \le \forall i \le 2m, \forall \tau \in \mathcal{J}$, if $[l_i, l_{i+1}] \notin \mathcal{I}(\tau)$, $p_i(\tau) = 0$.

Algorithm 3. EDD-WITH-STOP.

Init $\mathcal{J}_P$: set of "pending" location jobs, i.e., the location jobs that are currently executable and not finished yet, init with $\emptyset$
v: data mule's speed, init with v_{max}
$a(\tau)$: time allocated to job τ, init with 0
x_c: current location, init with X_s

On $\exists \tau \in \mathcal{J}_P,\ x_c = d(\tau)$ ▷ When a job is unfinished at its deadline location
1: $\mathcal{J}_u \leftarrow \{\tau | \tau \in \mathcal{J}_P, d(\tau) = x_c\}$ ▷ Set of jobs that needs to be finished here
2: $v \leftarrow 0$ ▷ Data mule stops
3: Complete each job in $\mathcal{J}_u$
4: $\mathcal{J}_P \leftarrow \mathcal{J}_P \setminus \mathcal{J}_u$
5: $v \leftarrow v_{max}$ ▷ Move at v_{max} again
6: $\tau_{ed} \leftarrow \arg\min_{\tau \in \mathcal{J}_P} d(\tau)$ ▷ Job with the earliest deadline location
7: Execute τ_{ed}

On $\exists \tau \in \mathcal{J},\ x_c = r(\tau)$ ▷ When jobs released
8: $\mathcal{J}_P \leftarrow \mathcal{J}_P \cup \{\tau | r(\tau) = x_c\}$
9: $\tau_{ed} \leftarrow \arg\min_{\tau \in \mathcal{J}_P} d(\tau)$
10: Execute τ_{ed}

On $\exists \tau \in \mathcal{J}_P,\ a(\tau) = e(\tau)$ ▷ When jobs finished
11: $\mathcal{J}_P \leftarrow \mathcal{J}_P \setminus \tau$
12: $\tau_{ed} \leftarrow \arg\min_{\tau \in \mathcal{J}_P} d(\tau)$
13: **if** $\tau_{ed} \neq \emptyset$ **then** Execute τ_{ed}
14: **end if**

—(Job completion) $\forall \tau \in \mathcal{J}$, $\sum_{i=0}^{2m} p_i(\tau) = e(\tau)$.
—(Processor demand) $0 \leq \forall i \leq 2m$, $\sum_{\tau \in \mathcal{J}} p_i(\tau) \leq z_i$.
—(Max/min speed) $0 \leq \forall i \leq 2m$, $v_{min} z_i \leq l_{i+1} - l_i \leq v_{max} z_i$.

4.4 Variable Speed with Acceleration Constraint

When there is a constraint on acceleration, the problem is NP-hard for the general location jobs case[3] [Sugihara and Gupta 2007]. Here we briefly present a heuristic algorithm that gives a feasible solution for both the simple and general location jobs cases. For more details on the algorithm, see Sugihara and Gupta [2010].

4.4.1 *Approach.* Figure 2 shows the idea of the heuristic algorithm. The algorithm works recursively, and in each recursion, we confine ourselves to the following three-phase speed-changing profile: first accelerate at the maximum acceleration, then move at the constant speed, and finally decelerate at the maximum negative acceleration. We call each of these intervals *accel interval*, *plateau interval*, and *decel interval*, respectively. Further we call the speed in the plateau interval the *plateau speed*.

[3]Complexity is unknown for simple location jobs case.

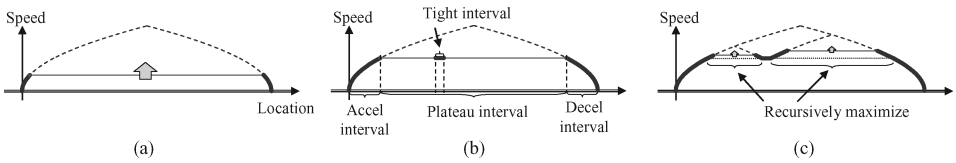

Fig. 2. Idea of the heuristic algorithm: (a) Increase the plateau speed. Dotted curves show the acceleration/deceleration for the fastest possible travel speed covering the whole interval. Bold and thin lines mean fixed intervals and free intervals, respectively. (b) A tight interval is found. (c) Recursively maximize the speed for the remaining free intervals.

As shown in Figure 2(b), the main idea of the algorithm is to maximize the plateau speed until we have a *tight interval*, which is defined as an interval whose length (in time) is equal to the processor demand for that interval. We can naturally extend the definition to define *tight location interval*, when we give the speed of the data mule for the interval.

For the intervals in the plateau interval but not in the tight interval, we can still increase the speed without violating the feasibility. As shown in Figure 2(c), we recursively apply the maximization procedure to them until there is no such interval.

4.4.2 *Details.* The heuristic algorithm consists of following four steps:

Step (1) Simplify: Converting general location jobs to simple location jobs. For each general location job, distribute the execution time to each feasible location interval proportionally to its length.

Step (2) Maximize: Find the maximum plateau speed and tight interval. In a similar way as for the SEQUENTIAL-FIND-MIN-MAXSPEED algorithm, for each interval of release location of the deadline location, calculate the plateau speed that makes it a tight interval. The maximum plateau speed is the minimum of these plateau speeds. From Theorem 4.2, this procedure does not destroy the feasibility, since the processor demand is equal to or less than the time given for each location interval of a release location and a deadline location.

Step (3) Trim: Trim feasible location intervals of each location job. We first process the accel interval and the decel interval, and then the tight interval. For each job, we calculate the time that needs to be allocated to the remaining location interval after trimming. We use the EDD algorithm and the LRT (latest release time) algorithm [Liu 2000], which is equivalent to the EDD in reversed time axis and is optimal.

Step (4) Recursion: By the previous step, all remaining jobs are separated into two disjoint groups. One group of jobs populate the location interval from the end of the accel interval to the beginning of the tight interval, and the other group populate the location interval from the end of the tight interval to the beginning of the decel interval. The speed for these intervals are not fixed yet, that is, there may still be some room for increasing the speed without destroying the feasibility. Thus, we recursively maximize the speed

by repeating from Step 2 for these intervals. The number of recursions varies from zero to two depending on the configuration of the tight interval.

5. PERIODIC 1-D DMS PROBLEM

In the periodic case of the 1-D DMS problem, each sensor node generates data at a given rate and a data mule travels across the sensor field periodically. This models a common type of sensor network application that continuously monitors the field in the long term. The objective is to minimize the period, that is, the time the data mule takes for each travel period, since it largely affects the data delivery latency.

We introduce some notations for the periodic case. Let T_t denote the travel time of one period. The data mule needs to stay at the base station for constant time T_b to deposit the data to the base station and refuel, etc. Thus the length T of one period is $T_t + T_b$. For the system to be stable, in each period of travel, the data mule needs to collect the data generated in one period.

5.1 Algorithms

5.1.1 *Processor Demand Analysis.* First we present an algorithm based on processor demand analysis. This algorithm applies to the constant-speed model with simple location jobs, that is, each location job has only one feasible location interval.

Let $e(\tau)$ denote the execution time of location job τ for one period. It is defined as follows:

$$e(\tau) \equiv \frac{\lambda(\tau)}{R}(T_t + T_b), \tag{1}$$

where $\lambda(\tau)$ is the data generation rate of the node represented by τ and R is the bandwidth, both of which are known constants.

Processor demand $g(I)$ for location interval I for one period is defined as $g(I) \equiv \sum_{\tau:I(\tau)\subseteq I} e(\tau)$, where $I(\tau)$ is the feasible location interval of location job τ. Let $g'(I)$ denote the processor demand for I for unit time, which is defined as follows:

$$g'(I) \equiv \frac{g(I)}{T_t + T_b} = \sum_{\tau:I(\tau)\subseteq I} \frac{\lambda(\tau)}{R}. \tag{2}$$

The set of location jobs is feasible if and only if the speed v of data mule satisfies

$$v \leq \min_{I\subseteq I_0} \frac{|I|}{g(I)} = \frac{1}{T_t + T_b} \min_{I\subseteq I_0} \frac{|I|}{g'(I)}, \tag{3}$$

where I_0 is the total travel interval.

When $T_b > 0$, we obtain the following constraint using $T_t = |I_0|/v$:

$$v \leq \left(\min_{I\subseteq I_0} \frac{|I|}{g'(I)} - |I_0|\right) \frac{1}{T_b}. \tag{4}$$

For a feasible solution to exist, the following must be satisfied:

$$|I_0| < \min_{I \subseteq I_0} \frac{|I|}{g'(I)}. \quad (5)$$

When this is satisfied, the maximum speed is the right-hand side of (4). When this is not satisfied, it is not possible to collect data without loss.

When $T_b = 0$, we obtain the following from Equation (3):

$$|I_0| \leq \min_{I \subseteq I_0} \frac{|I|}{g'(I)}. \quad (6)$$

Note that Equation (6) contains neither v nor T_t. What it implies is, when this is satisfied, the speed of data mule can be arbitrary. This validates the experimental observation in Kansal et al. [2004] that the speed of the data mule does not matter if the data mule travels the sensor field periodically.

5.1.2 *Online Algorithm.* For the variable-speed model with simple location jobs and $v_{min} = 0$, we can use an online algorithm almost the same as EDD-WITH-STOP. A data mule moves at v_{max} while executing a job with the earliest deadline location. When a node still has data at its deadline, the data mule stops there and collects the data until the node becomes empty. Here, *empty* refers to the state when the buffer of a node becomes empty as the data mule collects the data from it. Note that each node keeps generating data, so it will be the case that, after the buffer of a node becomes empty, it starts to fill again while the data mule is still in the communication range. In such a case, however, the data is collected in the next period.

5.1.3 *Linear Program Formulation.* For the constant- and variable-speed models where the above two approaches cannot be used (e.g., the general location jobs case), we can use a linear program formulation.

The formulation is almost the same as the ones for the non-periodic cases with general location jobs (Sections 4.2.2, 4.3.2), but the job completion constraint is replaced with the following:

—(Job completion) $\forall \tau \in \mathcal{J}$, $\sum_{i=0}^{2m} p_i(\tau) = (\sum_{i=0}^{2m} z_i + T_b)\lambda(\tau)/R$, where R is the bandwidth of communication from each node to the data mule.

The right-hand side is the amount of time to transmit the data generated in one period.

This linear program may be either infeasible or unbounded.[4] When it is infeasible, it is impossible to collect all data. When it is unbounded, the speed is arbitrary.

5.1.4 *Iterative Method.* When there is a constraint on acceleration, we can design a heuristic algorithm based on the one for the nonperiodic case (Section 4.4). Specifically, we can estimate T_t iteratively in the following manner:

—Solve the linear program for the variable-speed model by ignoring the acceleration constraint. Set the result to the initial value of $\hat{T}_t$, the estimate of T_t.

[4]It may be unbounded only in the constant-speed model.

—Repeat
  —Run the heuristic algorithm with setting $e(\tau) = \lambda(\tau)(\hat{T}_t + T_b)/R$. Denote the travel time as $\tilde{T}_t$.
  —If $|\tilde{T}_t - \hat{T}_t| < \epsilon$, break the loop. Otherwise, update $\hat{T}_t$ by $\tilde{T}_t$ and repeat.

5.2 Feasibility Analysis

For the constant-speed model with simple location jobs, we have discussed the feasibility conditions in Section 5.1.1.

For the variable-speed model, we have a feasibility condition when $v_{min} = 0$ for both the simple and general location jobs cases. Let's define the normalized total data generation rate α as follows: $\alpha = \sum_\tau \lambda(\tau)/R$. Then the following theorem holds.

THEOREM 5.1. *When $v_{min} = 0$, there exists a finite period T if $0 \leq \alpha < 1$.*

The proof is given in the Appendix.

6. 1-D DMS WITH MULTIPLE DATA MULES

In this section we discuss the one-dimensional DMS (1-D DMS) problem for the multiple data mules case.

6.1 Problem Definition

First we define k-DM 1-D DMS by extending the 1-D DMS problem:

> **k-DM 1-D DMS**
> INSTANCE: Set $\mathcal{J}$ of location jobs, for each location job $\tau \in \mathcal{J}$, an execution time $e(\tau)$, and a set $\mathcal{I}^{(k)}(\tau)$ of feasible location intervals, for each feasible location interval $I^{(k)} \in \mathcal{I}^{(k)}(\tau)$, a release location $r(I^{(k)})$ and deadline location $d(I^{(k)})$, a start $X_s^{(k)}$, a destination $X_d^{(k)}$, number of data mules K, and a constant T.
> QUESTION: Is there a feasible speed control plan and a feasible job schedule for K data mules satisfying $T_k \leq T$ for all k, where T_k is the travel time of kth data mule from $X_s^{(k)}$ to $X_d^{(k)}$?

Depending on the mobility assumption, there may be additional constraints (e.g., maximum speed, maximum acceleration) that determine the feasibility of a speed control plan. In the rest of the article, we focus on an optimization version of the problem, in which we minimize $\max_k T_k$.

For the optimization problem, we can use other objective functions as well. One example is the average travel time over all data mules $\sum_k T_k/n$. However, it does not appropriately reflect the overall data delivery delay, for example, when some of the data mules do not travel at all and have zero travel time. To avoid this, average travel time weighted by the amount of collected data ($\sum_k w_k T_k$) would be a more appropriate metric. However, this allows a small amount of data having a very large delay, which is also not preferable in some applications. It also has a practical issue that it is hard to be efficiently solved due to the nonlinear cost function. Maximum travel time $\max_k T_k$ gives a guarantee on the data delivery latency, and is appropriate for applications in which the maximum

delivery latency is important. Although this is also a nonlinear function, it can be converted to a linear function with additional constraints, as we will see later.

6.2 Constant Speed and Variable Speed

First we discuss the constant-speed and variable-speed cases. We consider two separate cases depending on whether the paths of data mules are identical or not.

6.2.1 *For Identical Paths.* When paths are identical for all data mules, feasible location intervals of each job are identical as well. We define the term *symmetric schedule* as follows. We call a schedule *symmetric* when the speed control plan and job schedule are identical in all data mules. Then we have the following theorem:

THEOREM 6.1. *For the optimization version of k-DM 1-D DMS with identical paths for the constant-speed or variable-speed models, there exists an optimal symmetric schedule.*

PROOF. We show there always exists a symmetric schedule whose speed is equal to or greater than that of the asymmetric one. Consider splitting the total travel interval into short location intervals by dividing it at locations which are either release or deadline locations of a job. Then, without loss of generality, we can assume each data mule moves at a constant speed in each of these short location intervals. For each of these intervals, let $v_1, v_2, \ldots, v_n$ denote the data mules' speed in the location interval. Without loss of generality, we assume $v_1 \leq v_2 \leq \cdots \leq v_n$. The total amount of collected data c satisfies $c \leq \sum_i l/v_i$, where l is the length of the location interval. The maximum time is l/v_1. Now, consider a symmetric schedule in which each data mule moves at $v' = n/(\sum_i 1/v_i)$, which is the harmonic mean of the original speed. Since $n(l/v') = \sum_i l/v_i \geq c$, it is possible to collect the same amount of data as in the original schedule. Maximum time is $l/v' \leq l/v_1$, where the equality is satisfied if and only if $v_1 = \cdots = v_n$. □

To find an optimal schedule, we first divide the execution time of each location job equally for each data mule and then apply the optimal algorithms for the single data mule case.

6.2.2 *For Arbitrary Paths.* When the path of each data mule is different, we formulate the problem as a linear program in the following way. The formulation is based on the same idea as the one in Section 4.3.2, but we introduce an additional variable z and constraints to convert the min-max objective into a linear objective. For the kth data mule, we split the location interval $[X_s^{(k)}, X_d^{(k)}]$ into $(2m^{(k)} + 1)$ location intervals $[l_0^{(k)}(= X_s^{(k)}), l_1^{(k)}]$, $[l_1^{(k)}, l_2^{(k)}], \ldots, [l_{2m}^{(k)}, l_{2m+1}^{(k)}(= X_d^{(k)})]$ $(l_i^{(k)} \leq l_{i+1}^{(k)})$, where $m^{(k)}$ is the number of feasible location intervals of all the location jobs that are executable at this data mule, and each $l_i^{(k)}$ is either a release location or a deadline location. Let $z_i^{(k)}$ denote the time that the kth data mule spends in location interval $[l_i^{(k)}, l_{i+1}^{(k)}]$, and $p_i^{(k)}(\tau)$

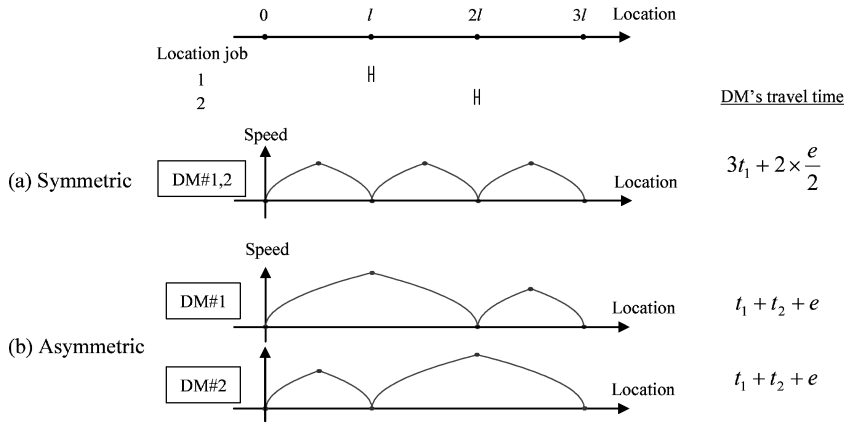

Fig. 3. Example of non-optimal symmetric schedule (in acceleration-constrained case): Two location jobs have zero-length feasible location intervals with execution time e.

denote the time it allocates to job τ in this interval. Using a variable Z to represent min-max objective, we have the following linear program:

Minimize Z subject to

—(Min-max objective) $\forall k$, $\sum_{i=0}^{2m^{(k)}} z_i^{(k)} \le Z$. Note that minimizing Z is equivalent to minimizing the maximum of the left-hand side over all k.

—(Max/min speed) $\forall k$, $0 \le \forall i \le 2m^{(k)}$, $v_{min} z_i^{(k)} \le l_{i+1}^{(k)} - l_i^{(k)} \le v_{max} z_i^{(k)}$ for the variable-speed model. For the constant-speed model, $(l_{i+1}^{(k)} - l_i^{(k)})/z_i^{(k)} = (l_{j+1}^{(k)} - l_j^{(k)})/z_i^{(j)}$ for all i, j satisfying $l_{i+1}^{(k)} - l_i^{(k)} > 0$, $l_{j+1}^{(k)} - l_j^{(k)} > 0$.

—(Positive allocation time) $\forall k$, $0 \le \forall i \le 2m^{(k)}$, $\forall \tau \in \mathcal{J}$, $p_i^{(k)}(\tau) \ge 0$.

—(Feasible intervals) $\forall k, 0 \le \forall i \le 2m^{(k)}, \forall \tau \in \mathcal{J}$, if $[l_i^{(k)}, l_{i+1}^{(k)}] \notin \mathcal{I}^{(k)}(\tau)$, $p_i^{(k)}(\tau) = 0$.

—(Job completion) $\forall \tau \in \mathcal{J}$, $\sum_k \sum_{i=0}^{2m^{(k)}} p_i^{(k)}(\tau) = e(\tau)$.

—(Processor demand) $\forall k$, $0 \le \forall i \le 2m^{(k)}$, $\sum_{\tau \in \mathcal{J}} p_i^{(k)}(\tau) \le z_i^{(k)}$.

6.3 Variable Speed with Acceleration Constraint

When there is a constraint on acceleration, the problem for the multiple data mules case is hard, as implied by the hardness of the single data mule case.

One interesting observation is that, for the identical paths case, the symmetric schedule is not always optimal. Figure 3 shows such an example. In this example, the data mules travel from location 0 to $3l$. Each of two location jobs (representing communication with two sensor nodes) have a zero-length feasible location interval at locations l and $2l$, respectively. All curves in the location-speed graphs represent acceleration/deceleration at the maximum rate a. Since $t_1 = 2\sqrt{l/a}$ and $t_2 = 2\sqrt{2}\sqrt{l/a}$, we have $3t_1 + e > t_1 + t_2 + e$ and thus the symmetric schedule is not optimal.

We design a heuristic algorithm based on an idea similar to List Scheduling [Graham 1969]. Figure 4 shows the algorithm. First the jobs are sorted in decreasing order of the number of executable data mules. This is to assign

—Sort the jobs in the ascending order of the number of executable data mules.
  —For the jobs with the same number of executable data mules, sort them by execution time in the descending order (i.e., long job first).
—For all jobs, from the head of the list,
  —Determine the strategy such that the maximum of current travel time is minimized. The strategy is one of the following:
    —**Assign**: Assign the job to one data mule that can execute it.
    —**Spread**: Divide the job equally among all the data mules that can execute it.
  —Remove the job from the list and update the travel time of each data mule.

Fig. 4. Heuristic algorithm for k-DM 1-D DMS problem with acceleration constraint.

the jobs that are executable by only one data mule first. In this way, there will be more freedom later in balancing the travel time of each data mule by appropriately allocating the jobs, which are executable by many data mules. The main idea of the algorithm is to assign jobs one by one to a data mule so that the maximum travel time is minimized. In addition to assigning a job to one of the data mules, we can also choose to spread the job to all data mules that can execute it by equally dividing the job's execution time, if the resulting maximum travel time is shorter than assigning the job to one data mule.

When each location job is not allowed to be executed by multiple data mules, we modify the heuristic algorithm by eliminating the "spread" option when determining the strategy. This modified version of the algorithm is also applicable to the case without acceleration constraint. Although it is not optimal anymore when applied to constant-speed or variable-speed cases, it is appropriate when it is infeasible for a single node to communicate with multiple data mules.

7. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed algorithms by numerical experiments. We first consider the variable-speed with acceleration constraint model and compare the performance of the proposed heuristic algorithm with multihop forwarding and a naive speed control algorithm. Then we compare one of the optimal algorithms with a previously proposed algorithm to examine how it can improve the performance and to argue how the proposed algorithms are useful in practice.

7.1 Performance of Heuristic Algorithm in Acceleration-Constrained Case

For the first experiment, we compare the data delivery latency and energy consumption between data mule approach and multihop forwarding approach. Then we compare the performance of the heuristic algorithm with a naive method the single and multiple data mules cases.

7.1.1 *Method.* We used Matlab for simulation experiments. Besides the heuristic algorithms, we have implemented a naive method for speed control. In the naive method, a data mule stops to collect data from each node one by

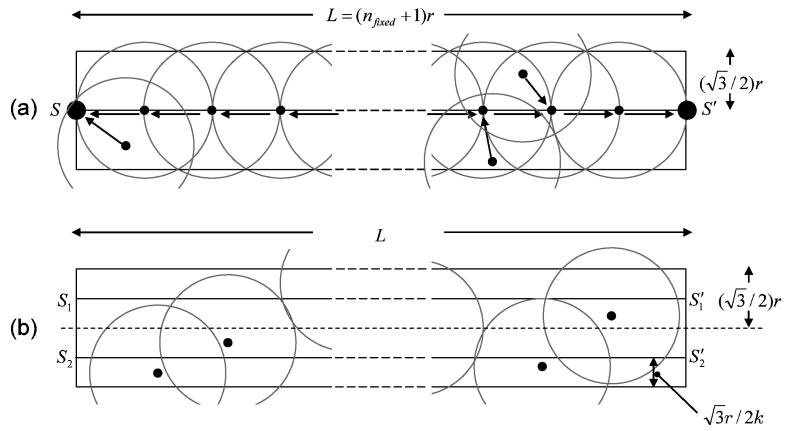

Fig. 5. Randomly generating test cases: (a) for comparison with multihop forwarding. Fixed nodes are aligned on line SS'; (b) For the multiple data mules case. An example with two data mules ($k = 2$) is shown.

one. Specifically, a data mule moves to the point on the path that is the closest to the node, collects data from the node while stopping, and moves to the next point. This is similar to the method used in [Ma and Yang 2007], though they did not assume an acceleration constraint. When there are multiple data mules, each node is assigned to the data mule whose path is the closest to it.

We randomly generated test cases in the following ways for each of the two experiments. For the first experiment, we needed to guarantee the connectivity between the base station and each node, since otherwise the multihop forwarding approach is not feasible. For this purpose, we generated node placements as shown in Figure 5(a), in which we arranged n_{fixed} "fixed" nodes on the horizontal dotted line SS', each of them being r apart, where r is the communication range. Base stations are located at S and S', assuming the data mule starts from and comes back to the base station. Then we put $n - n_{fixed}$ "random" nodes at random locations in the rectangle within the distance $(\sqrt{3}/2)r$ from the horizontal line. In this way we could guarantee that each of these random nodes had at least one fixed node or the base station within its communication range and thus was reachable by the base station via multihop forwarding. In the case of data mule approach, the data mule moved from S to S' on the straight line. All the nodes were within r from the trajectory and thus the data mule could collect data from them. We set $n_{fixed} = 20$, $n = \{20, 50, 80\}$, and $r = 100[m]$ in the experiment.

For the second experiment, on the multiple data mules case, we had random nodes only and no fixed nodes (Figure 5(b)). The vertical coordinate of the deployment area was $[-(\sqrt{3}/2)r, (\sqrt{3}/2)r]$, which was same as the first case. The path for ith data mule was a horizontal line on the vertical coordinate of $((2i - 1)\sqrt{3}/2k - \sqrt{3}/2)r$, where k is the total number of data mules. This setting made each data mule cover a strip of the same size. We used the same parameters as in the first experiment: $n = \{20, 50, 80\}$, $r = 100[m]$, and $L = 2100[m]$. The number of data mules was $k = 1, 2, 3$.

Table I. Comparison with Multihop Forwarding: Amount of Transmission is Relative to the Minimum Possible Amount. Maximum Latency with Asterisk is the Lower Bound Given in Gandham et al. [2006]

#Node (n)		Data Mule		Multihop
		Heuristic	Naive	Forwarding
20	Max. Latency (s)	220.0	620.0	(270)*
	Transmission (max./avg.)	1.0/1.0		10.0/5.5
50	Max. Latency (s)	502.1	1098.3	(720)*
	Transmission (max./avg.)	1.0/1.0		25.1/6.0
80	Max. Latency (s)	800.0	1538.9	(1170)*
	Transmission (max./avg.)	1.0/1.0		41.2/6.0

For both experiments, we set the execution time $e = 10$[sec], that is, a data mule needed 10 s to collect all the data from each node. Each experiment was repeated 10 times for each case and the average was used as the result. We used the variable-speed with acceleration constraint as the mobility model of the data mules and set $a_{max} = 1[m/s^2]$ and $v_{max} = 10[m/s]$ to roughly simulate the mobility capability of the radio-controlled helicopter used in Todd et al. [2007].

7.1.2 *Comparison with Multihop Forwarding.* First we compare the data mule and multihop forwarding approaches in terms of latency and energy consumption. We use one data mule in both the heuristic and the naive algorithms for the data mule approach. For the multihop forwarding algorithm, we use a simple example where each node forwards the data that it generated and received from other nodes to its neighbor node that is closest to the base station.

Table I shows the maximum latency and amount of transmission in these approaches averaged for 10 different node placements for each n. Latency is measured by the total travel time in case of the data mule approach. For the multihop forwarding approach, we use a lower bound derived in Gandham et al. [2006], where the authors studied the problem of minimum latency convergecast in several different network topologies. For a tree network, they gave a lower bound of $\max(3n_k - 3, N)$ where n_k is the size of the maximum branch and N is the total number of nodes. Since we can see the network used in our experiment as a tree network whose maximum branch has at least $n/2$ nodes, we use $3n/2 - 3$ as the lower bound of maximum latency in multihop forwarding. The amount of transmission is the amount of data that each node sends to the data mule (in the data mule approach) or a neighbor node (in multihop forwarding). It is normalized so that the value becomes 1.0 when a node only transmits all of its own data.

Maximum latency in the data mule approach with the heuristic algorithm turned out to be less than that of multihop forwarding. It is reasonable because in the data mule approach the communication is one way from each node to the data mule, whereas in multihop forwarding each node needs to do receiving as well as sending, which cannot be done at the same time. However, it should be noted that the results would be different if we compared the average latency instead of the maximum. In multihop forwarding, the average would be roughly

half the maximum since the data arrives at the base station in a trickle, whereas in the data mule approach the average is the same as the maximum since the data is delivered to the base station at a time when the data mule comes back. Within the data mule approach, latency in the heuristic algorithm was around half that in the naive method.

The amount of transmission is always minimum in the data mule approach, since each node sends its own data to the data mule and does not forward other nodes' data. On the other hand, it was around six on average in multihop forwarding and the maximum was proportional to the number of nodes. Since communications account for a major part of energy consumption at the nodes, these results suggest that the data mule approach is more energy efficient by roughly by six times compared to multihop forwarding. Moreover, since the maximum amount of transmission occurs at the node next to the base station and reachability to the base station is lost without this node, the functional lifetime in multihop forwarding is much shorter in this simple forwarding algorithm compared to the data mule approach.

7.1.3 *Multiple Data Mules Case.* Next we compare the heuristic algorithm and the naive method in a multiple data mules setting. Since the energy consumption at the nodes measured by the amount of transmission is identical in both methods, we compare the total travel time.

Figure 6 shows examples of speed control plans from these two methods. For random node placement as shown in Figure 6(a), we use two data mules that move along two dotted horizontal lines. Figure 6(b) shows the speed changes of two data mules in the heuristic algorithm. The travel times of the data mules are 264.55 s and 266.22 s, respectively. Figure 6(c) shows the ones in the naive method, in which the travel times are 727.75 s and 630.28 s, respectively. Not only is the maximum travel time much shorter in the heuristic algorithm, the travel time is also balanced between two data mules compared to the naive method.

Figure 7 shows the travel time for different numbers of data mules (k) and different numbers of nodes (n). For the $k = 2, 3$ cases, we use the maximum travel time among multiple data mules. In each case, the heuristic algorithm reduced the travel time by 40–60% from the naive method on average. The travel time does not improve for the case of the heuristic algorithm in $n = 20$ because it is almost the minimum possible travel time that is determined by L, a_{max}, and v_{max}.

7.2 Comparison with Adaptive Motion Control

Finally we compare the proposed algorithm with adaptive motion control (AMC) [Kansal et al. 2004; Somasundara et al. 2006] to see how it can improve the performance in data collection. AMC is a speed control algorithm designed for a periodic data collection scenario and the main idea is as follows: a data mule keeps a record of the data collection performance from each node in previous periods and slows down or stops to collect more data when it encounters a node with poor performance.

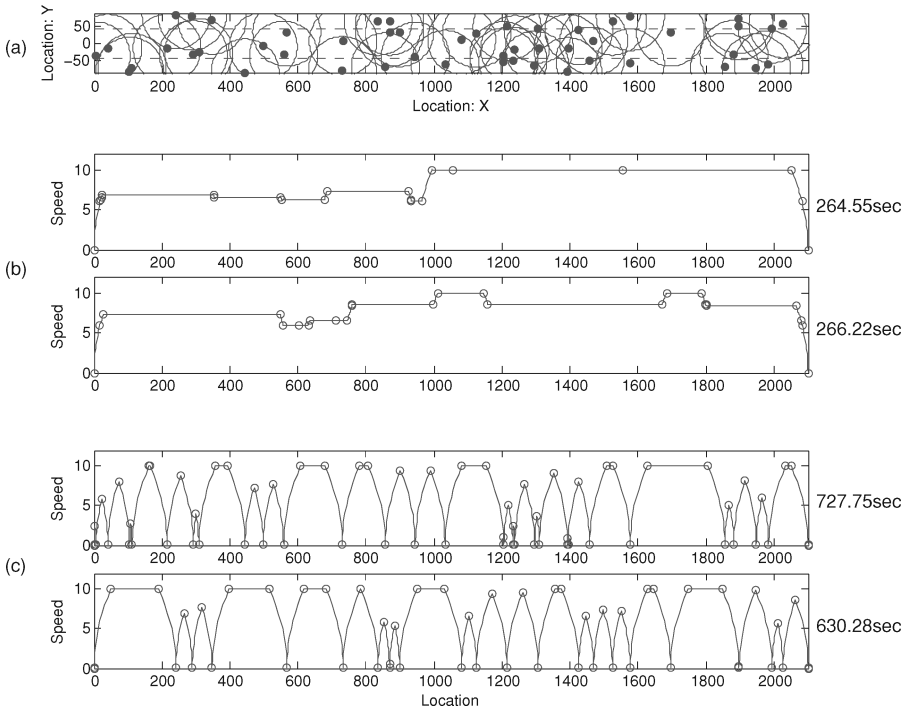

Fig. 6. Example of speed control plans (50 nodes; number of data mules $k = 2$): (a) Node placement. Filled circles are the node locations and large circles represent communication ranges. The two dotted lines correspond to the trajectories of two data mules; (b) speed control plans from the heuristic algorithm; (c) Speed control plans from the naive method.

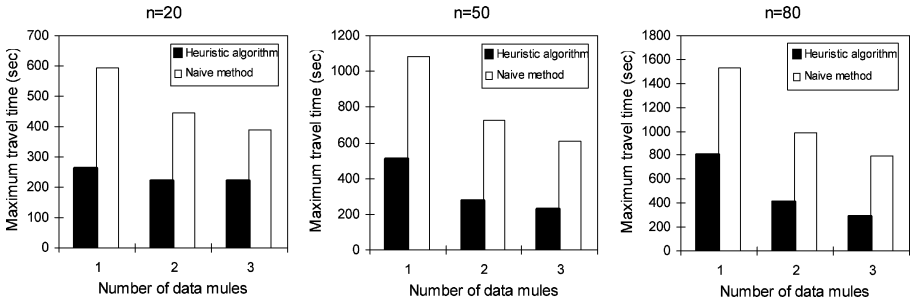

Fig. 7. Travel time for the multiple data mules case: Maximum of all data mules. Average of 10 experiments for each case.

Here is more detail of how AMC works. Given the latency requirement T_{req}, set the normal speed to $2s$, where $s = L/T_{req}$ and L is the total travel distance. In other words, the data mule can spend extra $T_{req}/2$ time to collect data from nodes with poor performance, since the travel time would be $T_{req}/2$ if it moved without stopping. Then the data mule chooses k worst-performing nodes based on previous performances, where k is a predetermined constant, and stops in

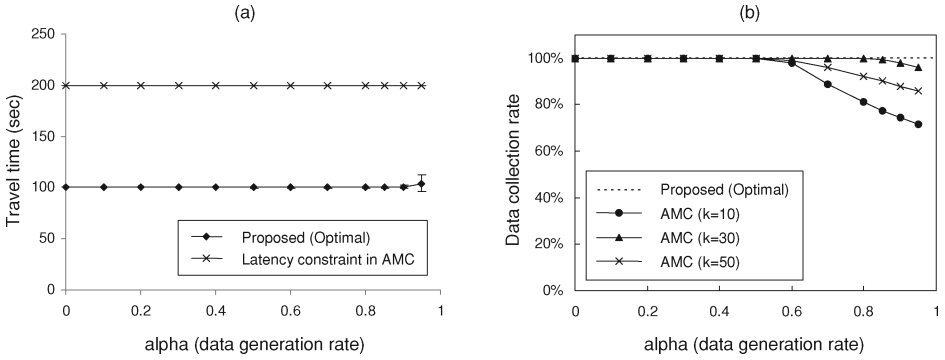

Fig. 8. Comparison with adaptive motion control (AMC): dense deployment ($L = 200$). (a) Travel time of each period (avg. $\pm$ s.d.); (b) data collection rate.

the communication range of each of these nodes for $T_{req}/2k$ time to collect more data.

To allow direct comparison between our algorithm and AMC, we tested a periodic data collection case and used the online algorithm based on EDD-WITH-STOP (details in Section 5.1.2) as the proposed method. We used the same parameters as in Somasundara et al. [2006], as follows: 50 nodes, speed range $[v_{min}, v_{max}] = [0, 2][m/s]$, and communication range $r = 25[m]$. Nodes were randomly deployed just as in Figure 5(b), except that the vertical range was $[-r, r]$ and there was only one data mule. We used two values for travel length L to make dense ($L = 200$) and sparse ($L = 2000$) deployments, where the dense deployment had the same node density as the one used in Somasundara et al. [2006]. For each case, we generated 20 random deployments and measured the travel time per period and the data collection rate, which is defined as the ratio of collected data among the generated data in a period. We tested various data generation rates by changing α (normalized total data generation rate: defined in Section 5.2) from 0 to 0.95. Each node generates data at the same rate.

There are some more parameters to choose for AMC. Since AMC takes required latency T_{req} as an input rather than finding the minimum latency, we set $T_{req} = 2L/v_{max}$. This is the minimum possible latency such that $2s$ does not exceed v_{max}; thus we can use AMC without modification. In addition, in Somasundara et al. [2006], how to choose k was not described and the value used in simulation experiments was not given. As a result, we used three different values ($k = 10, 30, 50$) to see the effect.

Figure 8 shows the results for dense deployments. Travel time is fixed to 200 s in AMC due to the reason explained above. For the proposed algorithm, the travel time is 100 s, which is the fastest possible for this deployment, except for cases of large α. As Figure 8(b) shows, the data collection rate decreased in AMC as α increased. The rate is 100% in the proposed algorithm by definition, whereas it was down to 72–96 % in AMC, depending on different values of k ($k = 30$ was the highest). Error bars are not shown in this graph for clarity, but the standard deviation was small ($< 1\%$ for all data points).

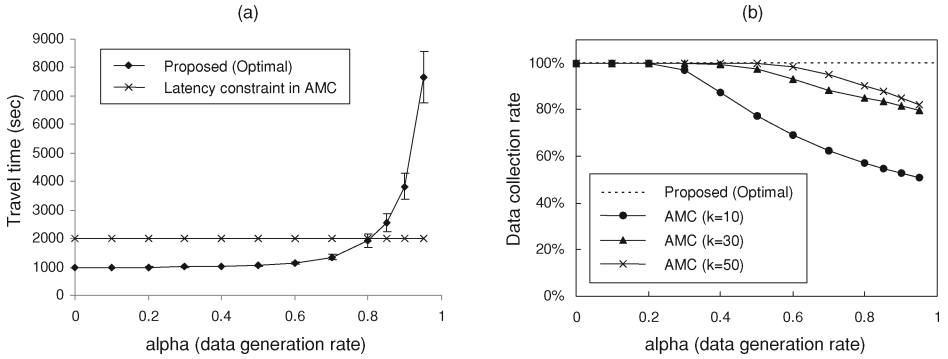

Fig. 9. Comparison with adaptive motion control (AMC): Sparse deployment ($L = 2000$).

Figure 9 shows the results for sparse deployments. Travel time rapidly increased in the proposed algorithm as α approached 1 and exceeded that for AMC. This means the specified latency requirement (in this case, $T_{req} = 2000$ s) cannot be satisfied theoretically without data loss. Figure 9(b) shows a similar trend as the dense deployment case for the data collection rate. The standard deviation was less than 4% for all data points. The rate got smaller in larger α and was down to 51–82%. A notable thing is that $k = 50$ yielded the best results, whereas $k = 30$ was the best in sparse deployments. This implies that, to maximize the data collection rate in AMC, it is critical to find an appropriate value of k for each different setting.

To summarize, the proposed algorithm improved the travel time, which is approximately equal to the data delivery latency, by up to 100% compared to AMC. More importantly, the apparently reasonable setting of latency requirement in AMC can be theoretically impossible to achieve in some cases. Not only giving a theoretical foundation for the speed control problem, the proposed algorithm is useful in practice as well, since it helps system designers to identify such cases and to choose appropriate parameters when they use AMC or other speed control algorithms.

8. CONCLUSIONS

We have presented the data mule scheduling (DMS) problem as a problem framework for the problem of motion planning for data mules. Then we focused on the 1-D DMS problem, which consists of the speed control and job scheduling subproblems, and presented optimal and heuristic algorithms for different mobility models and different problem settings including periodic data generation and multiple data mules. Through simulation experiments, we have shown that the data mule approach provides a large benefit over multihop forwarding in terms of energy efficiency and also that the proposed heuristic algorithm performs much better than the naive method. Further, by comparing our algorithm with a previously proposed method for speed control, we have demonstrated that our algorithm can be used in practice as well by providing theoretical limits to designers so that they can choose parameters accordingly.

APPENDIX

A.1 Proof of Theorem 4.2

We first show that, for any arbitrary interval, there exists a pair of release time and deadline that has the same processor demand:

LEMMA A.1. *For any t_1, t_2 satisfying $t_1 < t_2$ and $g(t_1, t_2) > 0$, there exist $t_1' \in \{r_i\}$ and $t_2' \in \{d_i\}$ such that $t_1 \le t_1' < t_2' \le t_2$, $g(t_1, t_2) = g(t_1', t_2')$.*

PROOF. Choose t_1', t_2' as follows:

$$t_1' = \min_{\tau_i \in T, r_i \ge t_1} \{r_i\},$$
$$t_2' = \max_{\tau_i \in T, d_i \le t_2} \{d_i\}.$$

Since $[t_1, t_2]$ contains at least one task, $t_1 \le t_1' < t_2' \le t_2$ is satisfied. Then $g(t_1, t_2) = g(t_1', t_2)$, since there is no task released in interval $[t_1, t_1')$. Similarly, since there is no task having a deadline in interval $(t_2', t_2]$, $g(t_1', t_2) = g(t_1', t_2')$. Therefore, $g(t_1, t_2) = g(t_1', t_2')$ for these t_1', t_2'. □

LEMMA A.2. *$g(t_1, t_2) \le t_2 - t_1$ for any t_1, t_2 satisfying $t_1 < t_2$ if and only if $g(t_1', t_2') \le t_2' - t_1'$ for any $t_1' \in \{r_i\}, t_2' \in \{d_i\}$ satisfying $t_1' < t_2'$.*

PROOF ("IF" PART). Proof by contrapositive: we first assume $\exists t_1, \exists t_2, (t_1 < t_2) \wedge (g(t_1, t_2) > t_2 - t_1)$ and prove $\exists t_1' \in \{r_i\}, \exists t_2' \in \{d_i\}, (t_1' < t_2') \wedge (g(t_1', t_2') > t_2' - t_1')$. Since $g(t_1, t_2) > t_2 - t_1 > 0$, by Lemma A.1, there exist $t_1' \in \{r_i\}$ and $t_2' \in \{d_i\}$ such that $g(t_1', t_2') = g(t_1, t_2) > t_2 - t_1$. Choose $t_1' = \min_{i:\tau_i \in T, r_i \ge t_1}\{r_i\}$ and $t_2' = \max_{i:\tau_i \in T, d_i \le t_1}\{d_i\}$. Since there is at least one task contained in $[t_1, t_2]$, $t_2' - t_1' > 0$ and $t_2 - t_1 \ge t_2' - t_1'$. Therefore, $\exists t_1' \in \{r_i\}, \exists t_2' \in \{d_i\}, (t_1' < t_2') \wedge (g(t_1', t_2') > t_2' - t_1')$.
("ONLY IF" PART). Obvious from Lemma A.1. □

The theorem follows from Theorem 4.1 and Lemma A.2. □

A.2 Proof of Theorem 4.3

Every valid schedule that uses the speed between 0 and v_{max} can be converted to one that only uses 0 and v_{max}. Thus, for a valid schedule, the time to stop is minimized if and only if the schedule is optimal. Further, for a valid and reasonable schedule that does not have idle time while stopping, the idle time while moving at v_{max} is minimized if and only if the schedule is optimal.

We consider another scheduling problem in which we want to maximize the allocated time, or equivalently to minimize the idle time. We do not allow the processor to execute a job after its deadline, and thus some jobs may be left unfinished. However, a nonstandard assumption is that partial job execution counts in this problem. We claim the following algorithm similar to EDD is optimal for the problem:

> Algorithm: At any time, execute a job with the earliest deadline from the set of available jobs.

Note that this algorithm is identical to the EDD algorithm when the system is underloaded. We show this algorithm minimizes the idle time by showing that an optimal schedule can be converted to it. Let A and A_{opt} denote the allocation by the algorithm and the optimal schedule, respectively. Allocation during time interval $[t_a, t_b]$ is denoted as $A(t_a, t_b)$. We compare A and A_{opt} from the beginning and swap allocations in A_{opt} as follows when they differ:

—*Case* 1: $A_{opt}(t_a, t_b) = \tau_1$, $A(t_a, t_b) = \tau_2$, $\tau_1 \neq \tau_2$. In this case, there exists a pair (t'_a, t'_b) such that $t'_a \geq t_b$ and $A_{opt}(t'_a, t'_b) = \tau_2$, since the time allocated to τ_2 by the time t_a in A_{opt} is shorter by $(t_b - t_a)$ than that in A, and thus τ_2 is not finished yet in A_{opt}. We can make a list of pairs $L = \{(t'_a, t'_b) | t'_a \geq t_b, A_{opt}(t'_a, t'_b) = \tau_2\}$ such that $\sum_{(t'_a, t'_b) \in L}(t'_b - t'_a) = t_b - t_a$. For all pairs (t'_a, t'_b) in L, we swap the allocation and obtain $A_{opt}(t'_a, t'_b) = \tau_1$ and $A_{opt}(t_a, t_b) = \tau_2$, which makes the allocation in (t_a, t_b) identical to A. It is possible because the time t'_b is before the deadline of τ_1, since $t'_b \leq d(\tau_2) \leq d(\tau_1)$ (because of EDD-based allocation).

—*Case* 2: $A_{opt}(t_a, t_b) = \emptyset$, $A(t_a, t_b) = \tau$. From the same argument as for Case 1, job τ is not finished in A_{opt} at t_a, and we can swap the allocation to obtain $A_{opt}(t'_a, t'_b) = \emptyset$ and $A_{opt}(t_a, t_b) = \tau$ for the pairs in $L = \{(t'_a, t'_b) | t'_a \geq t_b, A_{opt}(t'_a, t'_b) = \tau\}$ such that $\sum_{(t'_a, t'_b) \in L}(t'_b - t'_a) = t_b - t_a$.

—*Case* 3: $A_{opt}(t_a, t_b) = \tau$, $A(t_a, t_b) = \emptyset$. This does not happen for the following reason: since the allocation up to time t_a is identical, τ is not finished yet in A at t_a. However, this is a contradiction, since τ is available at t_a and the algorithm allocates time to a job whenever there are any available jobs.

EDD-WITH-STOP allocates exactly the same way as this algorithm when the data mule is moving (at v_{max}), and thus minimizes the idle time while moving. Therefore, EDD-WITH-STOP minimizes the total travel time. □

A.3 Proof of Theorem 5.1

Without loss of generality, we can assume that a data mule either moves at v_{max} or stops. Then the data mule is always in one of the following states:

(1) moving at v_{max} and executing a job (i.e., collecting data from a node);
(2) moving at v_{max} and not executing;
(3) stopping and executing a job;
(4) stopping at the base station (not executing).

Let T_1, T_2, T_3, T_b denote the time in each mode in a period. Since $T = T_1 + T_2 + T_3 + T_b$ and $T_1 + T_2 = L/v_{max}$, it is sufficient to show that T_3 is finite. Denoting the time the data mule spends on collecting data from this node by $t(\tau)$, we have

$$\begin{aligned} t(\tau)R &= \lambda(\tau)(T_1 + T_2 + T_3 + T_b), \\ \sum_{\tau} t(\tau) &= T_1 + T_3. \end{aligned}$$

From these equations, we have

$$\alpha(T_1 + T_2 + T_3 + T_b) = T_1 + T_3,$$

and thus, since $\alpha \neq 1$,

$$T_3 = \frac{\alpha}{1-\alpha}(T_2 + T_b) - T_1.$$

Therefore, if $0 \leq \alpha < 1$, T_3 is finite. □

REFERENCES

BARUAH, S. K., HOWELL, R. R., AND ROSIER, L. E. 1993. Feasibility problems for recurring tasks on one processor. *Theoret. Comput. Sci. 118*, 1, 3–20.

CHEBROLU, K., RAMAN, B., MISHRA, N., VALIVETI, P. K., AND KUMAR, R. 2008. BriMon: A sensor network system for railway bridge monitoring. In *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services* (*MobiSys*). 2–14.

CHEN, J.-J., WU, J., SHIH, C., AND KUO, T.-W. 2005. Approximation algorithms for scheduling multiple feasible interval jobs. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* (*RTCSA*). 11–16.

EKICI, E., GU, Y., AND BOZDAG, D. 2006. Mobility-based communication in wireless sensor networks. *IEEE Commun. Mag. 44*, 7, 56–62.

GANDHAM, S., ZHANG, Y., AND HUANG, Q. 2006. Distributed minimal time convergecast scheduling in wireless sensor networks. In *Proceedings of the 26th International Conference on Distributed Computing Systems* (*ICDCS*).

GRAHAM, R. L. 1969. Bounds on multiprocessing timing anomalies. *SIAM J. Appl. Math. 17*, 2, 416–429.

HO, M. AND FALL, K. 2004. Poster: Delay tolerant networking for sensor networks. In *Proceedings of the 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks* (*SECON*).

JEA, D., SOMASUNDARA, A. A., AND SRIVASTAVA, M. B. 2005. Multiple controlled mobile elements (data mules) for data collection in sensor networks. In *Proceedings of the 1st International Conference on Distributed Computing in Sensor Systems* (*DCOSS*). 244–257.

KANSAL, A., SOMASUNDARA, A. A., JEA, D. D., SRIVASTAVA, M. B., AND ESTRIN, D. 2004. Intelligent fluid infrastructure for embedded networks. In *Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services* (*MobiSys*). 111–124.

LIU, J. W. S. 2000. *Real-time Systems*. Prentice Hall.

MA, M. AND YANG, Y. 2006. SenCar: An energy efficient data gathering mechanism for large scale multihop sensor networks. In *Proceedings of the 2nd International Conference on Distributed Computing in Sensor Systems* (*DCOSS*). 498–513.

MA, M. AND YANG, Y. 2007. SenCar: An energy efficient data gathering mechanism for large-scale multihop sensor networks. *IEEE Trans. Parall. Distrib. Syst. 18*, 10, 1476–1488.

POLASTRE, J., HILL, J., AND CULLER, D. 2004. Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems* (*SenSys*). 95–107.

RHEE, I., WARRIER, A., AIA, M., AND MIN, J. 2005. Z-MAC: A hybrid MAC for wireless sensor networks. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems* (*SenSys*). 90–101.

SHIH, C., LIU, J. W. S., AND CHEONG, I. K. 2003. Scheduling jobs with multiple feasible intervals. In *Proceedings of the 9th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications* (*RTCSA*). 53–71.

SIMONS, B. AND SIPSER, M. 1984. On scheduling unit-length jobs with multiple release time/deadline intervals. *Operat. Res. 32*, 1, 80–88.

SOMASUNDARA, A. A., KANSAL, A., JEA, D. D., ESTRIN, D., AND SRIVASTAVA, M. B. 2006. Controllably mobile infrastructure for low energy embedded networks. *IEEE Trans. Mobile Comput. 5*, 8, 958–973.

SOMASUNDARA, A. A., RAMAMOORTHY, A., AND SRIVASTAVA, M. B. 2004. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *Proceedings of the 25th IEEE International Real-Time Systems Symposium* (*RTSS*). 296–305.

SOMASUNDARA, A. A., RAMAMOORTHY, A., AND SRIVASTAVA, M. B. 2007. Mobile element scheduling with dynamic deadlines. *IEEE Trans. Mobile Comput. 6*, 4, 395–410.

STANKOVIC, J. A., SPURI, M., NATALE, M. D., AND BUTTAZZO, G. C. 1995. Implications of classical scheduling results for real-time systems. *Comput. 28*, 6, 16–25.

SUGIHARA, R. AND GUPTA, R. K. 2007. Data mule scheduling in sensor networks: Scheduling under location and time constraints. UCSD Tech. rep. CS2007-0911. University of California, San Diego, La Jolla, CA.

SUGIHARA, R. AND GUPTA, R. K. 2010. Optimal speed control of mobile node for data collection in sensor networks. *IEEE Trans. Mobile Comput. 9*, 1, 127–139.

TODD, M., MASCARENAS, D., FLYNN, E., ROSING, T., LEE, B., MUSIANI, D., DASGUPTA, S., KPOTUFE, S., HSU, D., GUPTA, R., PARK, G., OVERLY, T., NOTHNAGEL, M., AND FARRAR, C. 2007. A different approach to sensor networking for SHM: Remote powering and interrogation with unmanned aerial vehicles. In *Proceedings of the 6th International Workshop on Structural Health Monitoring*.

VASILESCU, I., KOTAY, K., RUS, D., DUNBABIN, M., AND CORKE, P. I. 2005. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems* (*SenSys*). 154–165.

XING, G., WANG, T., JIA, W., AND LI, M. 2008. Rendezvous design algorithms for wireless sensor networks with a mobile base station. In *Proceedings of the 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (*MobiHoc*). 231–240.

XING, G., WANG, T., XIE, Z., AND JIA, W. 2007. Rendezvous planning in mobility-assisted wireless sensor networks. In *Proceedings of the 28th IEEE International Real-Time Systems Symposium* (*RTSS*). 311–320.

YAO, F., DEMERS, A., AND SHENKER, S. 1995. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society Press, Los Alamitos, CA, 374–382.

YE, W., HEIDEMANN, J., AND ESTRIN, D. 2002. An energy-efficient MAC protocol for wireless sensor networks. In *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies* (*INFOCOM*). 1567–1576.

ZHAO, W. AND AMMAR, M. 2003. Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks. In *Proceedings of the IEEE Workshop on Future Trends in Distributed Computing Systems*. 308–314.

Received March 2009; revised September 2009, December 2009; accepted December 2009