Efficient Path Planning for a Mobile Sink to Reliably Gather Data from Sensors with Diverse Sensing Rates and Limited Buffers

You-Chiun Wang and Kuan-Chung Chen

Abstract—Wireless sensor networks are vulnerable to energy holes, where sensors close to a static sink are fast drained of their energy. Using a *mobile sink (MS)* can conquer this predicament and extend sensor lifetime. How to schedule a traveling path for the MS to efficiently gather data from sensors is critical in performance. Some studies select a subset of sensors as *rendezvous points (RPs)*. Non-RP sensors send data to the nearest RPs and the MS visits RPs to retrieve data. However, these studies assume that sensors produce data with the same speed and have no limitation on buffer size. When the two assumptions are invalid, they may encounter serious packet loss due to buffer overflow at RPs. In the paper, we show that the path planning problem is NP-complete and propose an *efficient path planning for reliable data gathering (EARTH)* algorithm by relaxing these impractical assumptions. It forms a spanning tree to connect all sensors and then selects each RP based on hop count and distance in the tree and the amount of forwarding data from other sensors. An *enhanced EARTH (eEARTH)* algorithm is also developed to further reduce path length. Both EARTH and eEARTH incur less computationsl overhead and can flexibly recompute new paths when sensors change sensing rates. Simulation results verify that they can find short traveling paths for the MS to collect sensing data without packet loss, as compared with existing methods.

Index Terms—Data gathering, mobile sink, path planning, sensing rate, wireless sensor network.

1 Introduction

One core component of the emerging Internet of Things (IoT) technology is *wireless sensor network (WSN)* [1], which allows people to easily get environmental information. It contains a lot of small embedded devices, namely sensors, deployed in a region of interest. Each sensor keeps monitoring the surroundings and reporting its sensing data to a sink via multihop communications. Various WSN applications have been also developed, from air-quality monitoring [2] to health assessment [3], intelligent transportation [4], precision agriculture [5], and smart shopping [6].

Sensors are powered by small batteries, and it is uneconomic to replace batteries as the number of sensors is large [7]. Due to the nature of multihop communications, the sensors close to a static sink have to relay a great deal of data from others farther away from the sink. It results in non-uniform energy consumption of sensors, and those sensors near the sink will use up their energy quickly. Thus, there is a high possibility that the sink soon becomes disconnected from all sensors, thereby shortening WSN lifetime. This situation is called the *energy hole problem* [8].

Many research efforts have shown the effectiveness of using a *mobile sink (MS)* to solve the energy hole problem [9], [10]. The MS periodically moves along a preplanning route to visit sensors to gather their data. In this way, sensors need not relay data through many hops and thus save their energy on communications. Moreover, since the role of data forwarders (i.e., the sensors that connect to the sink) may change when the MS moves, the amount of energy consumed by each sensor could be more balanced.

The authors are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, 80424, Taiwan. Email: ycwang@cse.nsysu.edu.tw; m043040017@student.nsysu.edu.tw.

One critical issue is how to schedule a short traveling path for the MS to get data from sensors, as sensing data usually have timeliness (e.g., delay-sensitive applications [11] or event reports [12]). Intuitively, we can let the MS visit each sensor, and the problem will be same to the NP-complete *traveling salesman problem (TSP)* [13]. However, when the scale of WSN is large, such a solution may be infeasible since the MS is liable to violate the delay constraint of sensing data due to moving along a much longer path. Alternatively, one can select a subset of sensors as *rendezvous points (RPs)* to reduce path length [14]. In particular, non-RP sensors send data to nearby RPs and the MS moves to visit only RPs to collect all data. Thus, the problem becomes how to find RPs such that the amount of energy consumption due to multihop communications (from non-RP sensors to RPs) is minimized with the shortest traveling path.

Existing methods to find RPs assume that sensors produce data at the same speed (i.e., they have an equal sensing rate). Moreover, RPs have sufficiently large buffers to cache the forwarding data from neighbors and their sensing data before the MS comes to get them. In practice, these assumptions may not be always valid. For example, some sensors may be asked to accelerate their sensing rates once they detect abnormal events [12]. Besides, it is infeasible to let sensors possess large buffers due to their small-sized implementation [15], [16]. Thus, when the two assumptions are not available, existing methods will lead to packet loss at some RPs due to buffer overflow. We give an example in Fig. 1, where each sensor produces one packet of sensing data, and r_0 is the location of a base station (BS). To reduce the MS's traveling path while saving sensors' energy on communications, one will select sensors s_1, s_4, and s_6 to be RPs. Thus, each sensor can forward packets through just one hop and save its energy. In addition, the path's length is also minimized. However, suppose that each sensor can

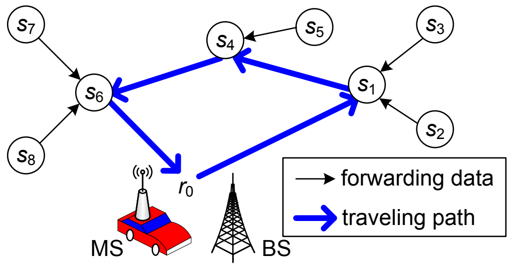


Fig. 1: An example to show the occurrence of packet loss when sensors have diverse sensing rates and limited buffers.

cache at most three packets in its buffer, and sensor s_8 now produces two packets. In this case, RP s_6 has to discard one packet with the original path. In fact, we can change the path to $r_0 \to s_1 \to s_4 \to s_6 \to s_8 \to r_0$ to avoid packet loss.

Motivated by the above observation, this paper proposes an *efficient path planning for reliable data gathering (EARTH)* algorithm to adaptively select RPs based on sensing rates and buffer sizes. The objectives are to 1) minimize the traveling path, 2) save the energy of sensors spent on data transmissions, and 3) prevent RPs from discarding packets due to buffer overflow. To do so, the EARTH algorithm constructs a *shortest-path (SP) tree* to connect all sensors. Starting from leaf nodes, it progressively finds candidate RPs according to the amount of relay data. Then, EARTH selects RPs from the candidates by referring to their hop counts and distances. Thus, each RP can have enough buffer space to cache all forwarding data and its own data, before the MS comes to retrieve them. Moreover, we also develop an *enhanced EARTH (eEARTH)* algorithm by efficiently replacing some RPs to further reduce path length, which helps speed up the MS's data gathering process.

Our contributions are threefold. First, we point out that existing methods do not consider the issues of diverse sensing rates and buffer limitation, so they may fail to support a reliable data gathering mechanism due to buffer overflow at RPs. Both the EARTH algorithm and its enhancement are proposed to address the practical issues. Second, many methods find a permanent path for the MS based on a constant sensing rate, so some of them involve in developing a complicated solution which incurs high computational overhead. Comparing with them, our proposed algorithms can adaptively compute new paths to deal with the case where sensors will need to adjust their sensing rates (e.g., increasing the rates when detecting events or decreasing the rates to save energy). Through complexity analysis, we verify that the proposed algorithms require less computational time. Third, with extensive simulation results, we show that our algorithms not only find a short traveling path, but also reduce energy consumption of sensors on communications. Moreover, they can efficiently solve the serious packet loss problem encountered by other methods.

We outline this paper as follows: Section 2 discusses related work. Section 3 formulates the path planning problem. In Section 4, we propose the EARTH algorithm and its enhancement. Afterwards, performance is evaluated in Section 5. Section 6 finally gives a conclusion.

2 Related Work

Our survey of related work has three parts. First, we study the issue of dispatching mobile nodes to visit sensors. Then, we address how to use one MS to directly gather data from every sensor. Finally, we discuss the research efforts that aim at selecting RPs to schedule the MS's traveling path.

2.1 Dispatching Multiple Mobile Nodes

Some studies use k *mobile ferries* to gather data from each sensor, whose objective is to decide the moving paths of ferries and make them meet at some points to exchange data. The work of [17] divides the sensing field into k grids and assigns a ferry to visit the sensors in each grid. Besides, some *contact points* are selected on grid boundary to make two ferries meet together. In [18], three path-planning methods are developed for ferries. The first method picks a contact point and cuts the sensing field into k regions which intersect at the point. Each ferry finds a Hamiltonian cycle to visit all sensors in a region and the contact point. The second method derives k cycles, where $k-1$ cycles are used to visit sensors while one cycle called *bus* is to connect all other cycles. Thus, ferries can exchange their data via the bus cycle. The third method uses one cycle to visit all sensors and each ferry moves along a segment of the cycle. When a ferry f_i contacts with the next ferry f_{i+1}, it sends data to f_{i+1}, just like a relay race.

A hybrid WSN is considered in [19], where static sensors detect events while mobile sensors move to visit event sites to gather data. Since mobile sensors have limited energy, the goal is to extend their lifetime by dispatching them in a round-by-round manner. Event sites are divided into clusters, and [19] assigns one mobile sensor to each group, which considers not only reducing the traveling paths of mobile sensors but also balancing their energy consumption. Then, the mobile sensor uses a TSP method to call on every site in the cluster. The work of [20] extends the dispatch problem with multi-attribute mobile sensors. Each event site has one attribute of sensing data but a mobile sensor can gather data with multiple attributes. To solve this problem, [20] pairs each mobile sensor with an event site to satisfy the Pareto optimality [21]. Then, each mobile sensor uses a spanning tree to connect unpaired event sites such that the mobile sensor has the same attribute with these event sites and the tree's weight is minimized. Thus, it can balance the amount of energy spent by mobile sensors.

Although the above studies deal with the issue of path planning, they have different objectives with our paper.

2.2 Using an MS to Visit Each Sensor

Given a set of sparsely scattered sensors, some studies schedule the traveling path of an MS to gather their data with the goal of minimizing path length. The work of [22] assumes that sensors have different communication ranges, and the MS can get data from a sensor once it touches any point within the sensor's communication range. Thus, [22] uses an evolutionary algorithm to find the touching points on the boundary of each sensor's communication range, and connects these points by a TSP method, so as to reduce the overall path. The work of [23] also finds touching points from the boundary of communication range, but it eliminates some redundant points (e.g., visiting the same sensor multiple times). In this way, [23] can further shorten the traveling path.

The work of [24] assumes that the MS's speed is adjustable and it keeps collecting data from a sensor when the MS roams within the sensor's communication range. Then, the objectives are to schedule the MS's traveling path and also change its speed along the path, such that the MS can gather data from each sensor with the shortest time. He et al. [25] use a progressive strategy to compute the traveling path with a delay constraint. They first use a TSP method to find a path to visit all sensors. By modeling the sensors' communication ranges as disks, the Welzl's method [26] is then used to find the smallest enclosing disks along the path. In this way, the path is iteratively reduced, as the MS can gather data from multiple sensors when it locates within the overlapping region of their communication ranges.

As discussed in Section 1, the traveling path becomes much longer when there are many sensors. Thus, the above schemes may not perform well in a large-scale WSN.

2.3 Selecting RPs to Gather Data

Both [27] and [28] assume that an MS keeps moving along a fixed and known trajectory, and sensors are scattered close to the MS's path. When a sensor resides within the MS's communication range (when the MS approaches), it becomes an RP to relay data for others. Thus, both studies focus on analyzing queuing delays and energy consumption of sensors in this network paradigm. Obviously, they deal with a different problem with our path planning problem.

The work of [29] considers a strip-shaped sensing field with both entrance and exit points r_e and r_x on its two sides, respectively. To find the traveling path of an MS, it divides the sensing field into two halves and finds a turning point r_t on the field's center line, such that when the MS moves along the path $r_e \to r_t \to r_x$, sensors in each half can spend less energy on sending data to the MS. Afterwards, [29] iteratively cuts each half into two equal parts and repeats the above process to find a turning point in each part, until the path's length reaches an upper bound. In this way, sensors near the traveling path will become RPs. However, this scheme may not be applied to a sensing field with arbitrary shape.

Some studies use a tree that link all sensors to find the traveling path and RPs. Specifically, [30] forms a minimum Steiner tree and traverses it in a preorder sequence, until the time that the MS moves to visit the selected vertices (i.e., RPs) exceeds the delay threshold. Since the Steiner tree may have *virtual* vertices (i.e., they do not correspond to sensors), it replaces such RPs by the nearest sensors. However, this method may cause longer data forwarding paths for some non-RP sensors, as the Steiner tree is traversed in preorder. So, it will force the sensors to spend more energy on communications. Xing et al. [14] create a routing tree in the WSN and assign each tree edge a weight based on the number of sensors using the edge to relay their data. Then, the MS selects the edges with larger weights under a delay constraint. Since the MS will move along tree edges, some RPs may be visited many times. Thus, [14] iteratively amends the traveling path via a TSP method. However, the TSP method is invoked n_{RP} times in every iteration, where n_{RP} is the number of RPs, so the scheme incurs high computational overhead of $O(n_{RP}^2\Upsilon)$, where Υ is the time used to run the TSP method.

A few studies seek to find the minimum RPs by limiting the hop count from each non-RP sensor to the nearest RP. The work of [31] forms an SP tree in the WSN. From each leaf node, the ancestor node k-hops away is selected as an RP. Then, each RP collects data from all nodes in its subtree, and the MS moves to visit RPs to gather data. The work of [32] proposes a single-hop data-gathering problem to find the fewest RPs such that every non-RP sensor can always find a one-hop RP to relay its data. This problem is formulated by a mixed-integer problem (i.e., NP-hard) and a spanning-tree covering heuristic is used to select RPs. Though non-RP sensors can spend less energy to send data to RPs, each RP has to collect data from relatively more sensors and spends more energy accordingly. Therefore, the energy hole problem may still occur at some RPs.

The work of [33] proposes a *cluster based (CB)* method to find RPs. It randomly selects some sensors as cluster heads, and each other sensor joins the cluster whose head is closest. Then, one RP is selected from each cluster. If the traveling path used to visit all RPs is shorter than a threshold $L_{\max}$, CB adds more cluster heads and repeats the procedure. However, since cluster heads are randomly selected, some clusters may have more sensors, which makes their sensors spend more energy on communications. In [34], a *weighted rendezvous planning (WRP)* scheme is proposed by forming a spanning tree in the WSN. Each sensor s_i is given a weight $W_i = N_i \times H(i, M)$, where N_i is the number of packets that s_i sends to an RP and $H(i, M)$ is the hop count between s_i and its closest RP. Since WRP assumes that each sensor produces one packet of sensing data, N_i is the number of s_i's children plus one. Then, WRP selects an RP with the maximum weight and uses a TSP method to schedule a path to visit each RP. The iteration is repeated until path length exceeds $L_{\max}$. However, since it uses the TSP method to recompute the path to visit RPs (including the new found one) in each iteration, WRP has time complexity of $O(n^2\Upsilon)$, which is higher than that in [14].

Comparing with existing methods, our paper considers the issues of diverse sensing rates and limited buffers, which are rarely studied in the literature. Thus, it can find a better traveling path to save sensors' energy on communications, avoid RPs dropping packets due to buffer overflow, and help the MS efficiently gather data from RPs.

3 Problem Definition

We first present the network architecture, followed by the energy model. Then, we depict the path planning problem. Finally, we discuss some practical issues.

3.1 Network Architecture

Consider a set of sensors $\mathcal{S} = \{s_1, s_2, \cdots, s_n\}$ and a BS r_0 that form a connected network. Sensors are homogeneous, so they have identical buffer capacity and communication range. Each sensor s_i has a sensing rate r_i (i.e., the number of packets produced per unit time), and it can store at most B packets of sensing data in its buffer. The destination of all sensing data is r_0. There is an MS that periodically makes a tour around the network to collect data from sensors. The MS has to come back to r_0 to offload its gathering data. We divide the time axis into multiple *rounds* with length T, where T is a given input (and we will discuss how to set its value in Section 3.4). In each round, the MS moves from r_0, visits some sensors, and goes back to r_0. Note that each sensor produces at most $m_i = \lfloor r_i \times T \rfloor$ packets in a round, where m_i is no larger than B, otherwise buffer overflow must occur at some sensors.

Based on [34], we make some assumptions on the MS. First, the location of each sensor is known, so the MS can be aware of its tour. Second, the MS moves with a constant speed of v. Third, comparing with the MS's traveling time, the communication time that the MS spends to receive data from the visiting nodes can be ignored. Finally, RPs will receive all

TABLE 1: Summary of common notations.

Notation	Definition
$\mathcal{S}$	the set of sensors, where $\|\mathcal{S}\| = n$
$\mathcal{R}$	the solution set of RPs
$\mathcal{F}$	the set of checking nodes
$\mathcal{C}$	the set to record the information of RP candidates
$\mathcal{V}_i$	the set of sensors which will send their data to a node s_i
$\mathcal{U}_j$	the set of deputies for an RP r_j
r_0	the BS where the MS can offload its data
m_i	the number of packets produced by sensor s_i in a round
a_i	the accumulative number of packets
B	the capacity of a buffer (in packets)
$\hat{H}(\cdot,\cdot)$	the hop count between two nodes
$\hat{D}(\cdot,\cdot)$	the Euclidean distance between two nodes
$\hat{P}(\cdot,\cdot,\cdot)$	the length of a path to visit three nodes in sequence

data from non-RP sensors before the MS comes to visit them, so the MS need not wait at RPs to get data.

3.2 Energy Model

Suppose that a sensor s_i transmits a packet with y bits to another sensor s_j, and their Euclidean distance is denoted by $\hat{D}(s_i, s_j)$. Then, the amount of energy consumed by s_i on data transmission can be calculated as follows [35]:

$$E_{\mathrm{Tx}}(s_i, s_j) = (\psi_1 + \psi_2[\hat{D}(s_i, s_j)]^{\varepsilon}) \times y, \tag{1}$$

where ψ_1 and ψ_2 indicate the amount of power required by the transmitter and amplifier circuits to send out one bit, respectively, and ε is an exponent for path loss, which ranges from 2 to 4 (usually set to 2). In Eq. 1, $\psi_2[\hat{D}(s_i, s_j)]^{\varepsilon}$ is the amount of energy spent by the amplifier for one bit. On the other hand, s_j also consumes some energy to receive the packet, which is measured by

$$E_{\mathrm{Rx}}(s_j, s_i) = \psi_3 \times y, \tag{2}$$

where ψ_3 gives the amount of power taken by the receiver circuit to get one bit. We will use Eqs. 1 and 2 to evaluate the amount of energy spent by sensors on communications.

3.3 Path Planning Problem

Given $\mathcal{S}$ and r_0, the path planning problem asks how to find a set of RPs $\mathcal{R} = \{r_0, r_1, r_2, \cdots, r_k\}$, where $\{r_1, r_2, \cdots, r_k\}$ are picked from $\mathcal{S}$, and decide a traveling path for the MS to start and finish its journey at r_0 and visit each of other nodes in $\mathcal{R}$ once, such that 1) the length of traveling path is minimized, 2) the amount of energy spent by sensors to send their data to the MS is minimized, and 3) the number of packets dropped by RPs due to buffer overflow is minimized. Note that each non-RP sensor has to send its data to the nearest RP to satisfy the second objective. Thus, all data forwarding paths of sensors will constitute a forest, where each tree is rooted at an RP. Fig. 1 shows an example.

Below, we formulate the path planning problem as a decision problem in Definition 1 and prove that it is NP-complete by Theorem 1. Table 1 summarizes our notations.

Definition 1. *Given $\mathcal{S}$ and r_0, the* path planning decision (PPD) *problem asks whether there exists a traveling path with length of L for the MS to collect data from all sensors in $\mathcal{S}$ such that the energy consumption of sensors is E and the number of packets discarded by RPs is X.*

Theorem 1. *The PPD problem is NP-complete.*

Proof: We first show that the PPD problem belongs to the NP class. Given a PDD problem instance and a traveling path of length L, we can verify whether the path meets the requirements of energy consumption E and packet loss X in polynomial time. Thus, the part is proved.

Then, we reduce an NP-complete problem, the minimum Hamiltonian cycle problem, to the PDD problem. Given a set of n points, the *minimum Hamiltonian cycle decision (MHCD)* problem asks whether there is a cycle with length L that passes each point once. We construct a PPD problem instance by considering a WSN with n sensors. Suppose that two adjacent sensors s_i and s_j both produce $B/2$ packets. Except for s_i and s_j, each sensor produces B packets. Then, the PDD problem instance asks whether we can find a traveling path whose starting and ending points are at r_0 such that the amount of energy consumed by sensors is

$$E = E'_{\mathrm{Rx}}B/2 + (n-1)E'_{\mathrm{Tx}}B, \tag{3}$$

and there is no packet loss by RPs (i.e., $X = 0$), where E'_{Tx} and E'_{Rx} denote the amount of energy consumption to transmit and receive a packet, respectively. Assume that s_j is closer to r_0 than s_i in terms of hop count. Obviously, to avoid the occurrence of packet loss, the only way is to let every sensor (except for s_i) to be an RP, and s_i forward its data to s_j. In this case, s_j spends energy of $E'_{\mathrm{Rx}}B/2$ to receive all packets from s_i, and totally $n-1$ RPs each spends energy of $E'_{\mathrm{Tx}}B$ to send its data to the MS. Thus, Eq. 3 will be the amount of energy consumed by sensors on communications. By setting the round time $T = L/v$, the PDD problem instance will ask to find a Hamiltonian cycle with length of $T \times v = L$ to visit n points (specifically, $n-1$ sensors and r_0), which is identical to the MHCD problem. The above reduction takes polynomial time, so the PPD problem is also NP-complete. □

3.4 Practical Issues

We address three issues arisen in the path planning problem:

- How to implement its solution in a real WSN?
- What are the extra overheads for communications?
- How to decide the round time T?

For the first issue, given the sensing rate r_i of each sensor and the round time T, the BS (i.e., r_0) can find the set of RPs by a path planning solution (e.g., our EARTH/eEARTH algorithms). Since all sensors in $\mathcal{S}$ and the BS form a connected network, the BS can announce the information of which nodes should serve as RPs to the WSN. Thus, non-RP sensors will forward their sensing data to the nearest RPs. Besides, as the MS initially locates at r_0, the BS can directly tell it the set of RPs along with the traveling path. Then, the MS will move along the path and come back to the BS to offload its collected data. The path planning problem is critical to the WSNs used in long-term monitoring applications, where sensors regularly produce sensing data for a long time [36]. To save their energy, the MS will move to visit RPs to collect the sensing data produced by sensors. Let us take the weather monitoring application in [36] as an example, where each sensor produces one 15-byte packet every 10 minutes, which includes the ZigBee header and weather-related data such as temperature and humidity. Each RP will cache these packets produced by itself and collected from nearby sensors, and wait for the MS to retrieve them.

For the second issue, there are two extra communication overheads for sensors. One is the *RP-announcement overhead*, where the BS has to notify sensors of the set of designate RPs. However, this overhead is 'constant' in the sense that any solution to the path planning problem will incur this overhead when a new set of RPs are found by the BS. The other is called the *rate-adjustment overhead*, where the BS will send messages to certain sensors to change their sensing rates based on the application requirement. Specifically, in many WSN applications [37]–[39], the BS can ask some sensors to increase their sensing rates in case of event occurrence, or decrease the rates to save sensors' energy. Therefore, the BS actually knows the sensing rates of sensors beforehand without querying them. Obviously, the rate-adjustment overhead is also viewed as a constant for any path planning solution, because the overhead is not caused by the solution itself. Since both RP-announcement and rate-adjustment overheads will be the same for any path planning solution, we will not evaluate their effects in Section 5. Instead, we aim to measure the amount of sensors' energy spent on sending/receiving sensing data due to the selection of RPs by different solutions.

For the last issue, most of the RP-finding methods discussed in Section 2.3 also dispatch the MS to collect data from sensors in a round-by-round manner. They assume that each sensor produces the same number of packets in a round, which may not be practical. That is why we consider different sensing rates in our path planning problem. To decide the round time T, these methods put an upper-bound limitation $L_{\max}$ on the traveling path that the MS can move in a round (e.g., due to its battery capacity). Suppose that the MS moves in a constant speed v. Then, we can set the default value of T to $L_{\max}/v$. Alternatively, we can also set the round time to the maximum tolerable delay that the BS should get sensing data from sensors. In this way, each sensor is expected to produce at most m_i packets in the worst case. To get a more accurate value of T, we can run a path planning solution (e.g., EARTH or eEARTH) and measure the actual time that the MS spends to visit all RPs and go back to r_0. In case that we get a smaller T value, we can use this new value for the following rounds (until the BS computes a new set of RPs due to the drastic change of sensing rates). Otherwise, it means that using one MS is not enough to collect data from all sensors. Thus, we can divide the WSN into non-disjointed parts and use one MS to conduct the job of data collection in each part.

4 The Proposed Algorithms

In this section, we present the design and discussion of our EARTH algorithm, followed by its enhanced version.

4.1 The EARTH Algorithm

The basic idea of our EARTH algorithm is to construct a tree in the WSN and traverse the tree to quickly identify *potential* RPs based on buffer limitation. Among these candidates, we select RPs by referring to their hop counts, distances, and subtrees. This procedure is repeated until we have visited all tree nodes. Then, the traveling path is created by connecting the selected RPs. In particular, Fig. 2 gives the flowchart of EARTH, which contains four phases below.

1. *Tree construction:* We first form a tree to connect all sensors and add leaf nodes to a set $\mathcal{F}$.

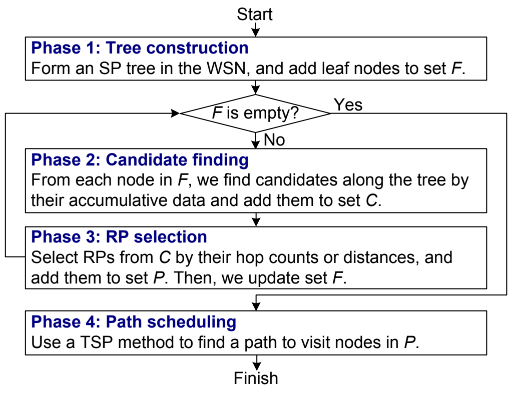

Start
Phase 1: Tree construction
Form an SP tree in the WSN, and add leaf nodes to set F.
F is empty?
Yes
No
Phase 2: Candidate finding
From each node in F, we find candidates along the tree by their accumulative data and add them to set C.
Phase 3: RP selection
Select RPs from C by their hop counts or distances, and add them to set P. Then, we update set F.
Phase 4: Path scheduling
Use a TSP method to find a path to visit nodes in P.
Finish

Fig. 2: The flowchart of our EARTH algorithm.

2. *Candidate Finding:* From the nodes in $\mathcal{F}$, we find RP candidates along the tree and add them to a set $\mathcal{C}$.
3. *RP Selection:* We select RPs from the nodes in $\mathcal{C}$, add them to $\mathcal{R}$, and update $\mathcal{F}$ accordingly. If $\mathcal{F} = \emptyset$ (which means that all RPs have been found), we go to phase 4. Otherwise, we return to phase 2 to find new candidates.
4. *Path Scheduling:* We finally use a TSP method to find a traveling path for the MS to visit all RPs in $\mathcal{R}$.

4.1.1 *Phase 1–Tree Construction*

Each sensor in $\mathcal{S}$ is associated with a *checked flag* (initially set to 'false'), which indicates whether this sensor has been selected as an RP or chosen one RP to forward its data. Afterwards, we construct an SP tree rooted at r_0 to connect all sensors, which can be done by the Dijkstra's algorithm. As mentioned in Section 3.3, the solution set $\mathcal{R}$ of RPs must contain r_0, so we have $\mathcal{R} = \{r_0\}$ in the beginning. Remark 1 discusses why we choose to construct an SP tree.

Then, we check the tree in a *bottom-up* manner. To do so, we create a set $\mathcal{F}$ of nodes to be checked. When a node $s_i \in \mathcal{S}$ satisfies either of the two conditions, it is added to $\mathcal{F}$: 1) s_i is a leaf node or 2) each of s_i's children has a true checked flag. Obviously, $\mathcal{F}$ contains merely leaf nodes in phase 1, as the checked flags of all nodes are false.

Remark 1 (The reasons of using an SP tree)**.** *We create an SP tree to select RPs due to two reasons. First, in a WSN with a static sink, it is natural to construct an SP tree for the routing purpose [7], because each sensor can send data to the sink with the minimum hop count and save its energy accordingly. Moreover, the SP tree helps us determine the ancestor-descendant relationship between any two sensors, so as to facilitate the process of selecting RPs. Second, a number of TSP heuristics or approximation algorithms rely on the structure of an SP tree to find the traveling path [13]. Thus, using an SP tree can also increase the possibility of reducing path length.*

4.1.2 *Phase 2–Candidate Finding*

Starting from a node in $\mathcal{F}$, we move towards the root r_0 and accumulate the number of packets produced by the visited nodes, which is denoted by a_j. When we visit a node s_j such that $a_j = B$ (i.e., reach the maximum capacity of a buffer), s_j will be an RP candidate. Let us denote by $\mathcal{V}_j$ the set of all nodes visited by the above procedure (excluding s_j). Specifically, we have

$$a_j = m_j + \sum\nolimits_{s_i \in \mathcal{V}_j} m_i. \tag{4}$$

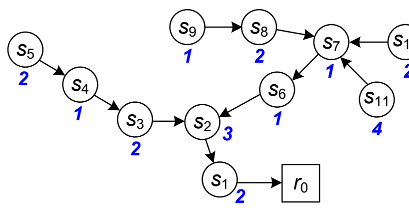


Fig. 3: An example of candidate finding, where the number close to each circle indicates the number of packets produced by the sensor.

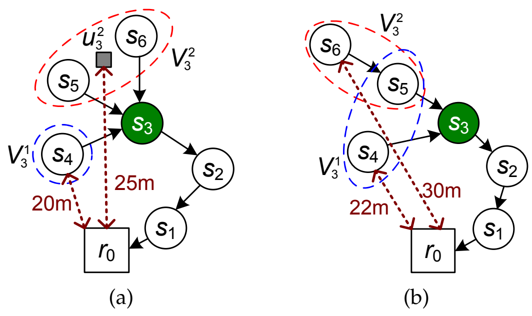


Fig. 4: Examples of RP selection: (a) $\mathcal{V}_3^1 \cap \mathcal{V}_3^2 = \emptyset$ and (b) $\mathcal{V}_3^1 \cap \mathcal{V}_3^2 \neq \emptyset$.

From Eq. 4, $\mathcal{V}_j$ indicates the subtree rooted at s_j where the nodes in the subtree can forward their data to s_j under the constraint of buffer capacity B. Moreover, we also create a set $\mathcal{C}$ to record the information of RP candidates, where each element of $\mathcal{C}$ is a two-tuple $(s_j, \mathcal{V}_j)$. Recall that the round time T is a given input to the path planning problem, so we can obtain the maximum number of packets that each sensor produces (i.e., m_j) in Eq. 4 before deciding the traveling path of the MS.

When visiting a node s_j, two cases may occur:

- *Case I: s_j has no or one child.* If $a_j < B$, we then check its parent. When either a_j reaches B or s_j's parent has a true checked flag, we let s_j be a candidate.
- *Case II: s_j has multiple children.* We use the depth-first search (DFS) to check other children of s_j. There are three subcases. 1) If the procedure ends at any descendant node[1] of s_j (due to $a_j \geq B$), we mark s_j as a candidate. 2) If every child of s_i leads to $a_j \geq B$, we then stop the procedure at s_j. 3) When we have visited all descendant nodes of s_j but a_j is still below B, we use the rule of case I to continue the procedure.

Fig. 3 gives an example, where $B = 6$. By phase 1, we have $\mathcal{F} = \{s_5, s_9, s_{10}, s_{11}\}$. Starting from s_5, we visit s_4 and s_3. By case I, if we visit s_2, it leads to $a_2 = 2+1+2+3 > B$, so s_3 is a candidate and $\mathcal{V}_3 = \{s_4, s_5\}$. Then, starting from s_9, we visit s_8 and s_7, and find that s_7 has multiple children. By case II, we visit s_{10} and get $a_7 = B$. Thus, we mark s_7 as a candidate and $\mathcal{V}_7 = \{s_8, s_9, s_{10}\}$. Similarly, starting from s_{10}, we visit s_7, s_8, and s_9, and derive the same result of $(s_7, \{s_8, s_9, s_{10}\})$. Finally, starting from s_{11}, we check its parent s_7. If we accumulate the data of either s_8 or s_{10}, it leads to $a_7 = 4+1+2 > B$. So, we stop at s_7. Therefore, we obtain that $\mathcal{C} = \{(s_3, \{s_4, s_5\}), (s_7, \{s_8, s_9, s_{10}\}), (s_7, \{s_{11}\})\}$. Note that a node (e.g., s_7) may be included in multiple elements of $\mathcal{C}$, each with different $\mathcal{V}_i$ sets. It means that the node can collect data from different sets of sensors, and these elements are viewed as different cases.

4.1.3 Phase 3–RP Selection

For each candidate node[2] s_i in $\mathcal{C}$, we give it a weight by

$$w_i = \min_{\forall r_j \in \mathcal{R}} \{\hat{H}(s_i, r_j)\}, \tag{5}$$

where $\hat{H}(s_i, r_j)$ is the hop count between s_i and r_j in the tree. From Eq. 5, w_i is actually s_i's hop count to the nearest RP in $\mathcal{R}$. Then, the node with the largest w_i value is selected as an RP. Fig. 3 gives an example. Since $\mathcal{R} = \{r_0\}$, we have $w_3 = 3$ and $w_7 = 4$. Thus, s_7 is selected as an RP.

1. We examine only the descendant nodes with false checked flags.
2. That is, the node in the first tuple of each element in $\mathcal{C}$.

However, if the selected node s_i has multiple elements in $\mathcal{C}$, we should find extra RP(s) from its $\mathcal{V}_i$ sets, which are denoted by $\mathcal{V}_i^1, \mathcal{V}_i^2, \cdots, \mathcal{V}_i^k$. There are two cases to be discussed. In case of $\mathcal{V}_i^1 \cap \mathcal{V}_i^2 \cap \cdots \cap \mathcal{V}_i^k = \emptyset$, it means that we make s_i collect data from disjointed sets of sensors in phase 2. Thus, s_i can pick one $\mathcal{V}_i$ set of sensors to collect data, and all one-hop neighbors of s_i in other $\mathcal{V}_i$ sets have to act as RPs to collect data from the nodes in their $\mathcal{V}_i$ sets. To do so, we define a weight ϖ_i^h for each $\mathcal{V}_i^h$ set by

$$\varpi_i^h = \min_{\forall r_j \in \mathcal{R}} \{\hat{D}(u_i^h, r_j)\} + (|\widetilde{\mathcal{V}}_i^h| - 1) \times \sigma, \tag{6}$$

where $\widetilde{\mathcal{V}}_i^h$ is the set of all s_i's one-hop neighbors in $\mathcal{V}_i^h$, u_i^h is the centroid of all nodes in $\widetilde{\mathcal{V}}_i^h$ (whose coordinates can be computed by taking the average of coordinates of these nodes), and σ is a small constant. Remark 2 gives the idea behind Eq. 6. To reduce the MS's traveling path, s_i selects the $\mathcal{V}_i^h$ set with the maximum ϖ_i^h value to collect data, and the one-hop neighbors of other $\mathcal{V}_i$ sets are marked as RPs. We take Fig. 4(a) as an example, where $\mathcal{C} = \{(s_3, \{s_4\}), (s_3, \{s_5, s_6\})\}$ and $\sigma = 5$. Thus, s_3 is an RP and we have $\mathcal{V}_3^1 = \{s_4\}$ and $\mathcal{V}_3^2 = \{s_5, s_6\}$. Since $\mathcal{V}_3^1$ has only one node, the centroid is the node itself. Then, we can obtain weights $\varpi_3^1 = \hat{D}(s_4, r_0) = 20$ and $\varpi_3^2 = \hat{D}(u_3^2, r_0) + 5 = 30$. Thus, s_3 picks the $\mathcal{V}_3^2$ set (i.e., s_5 and s_6) to collect data, and s_4 acts as an RP accordingly.

Otherwise, these $\mathcal{V}_i$ sets have some overlapped nodes, and we use $\mathcal{V}_i^{\mathbf{O}}$ to denote the set of such nodes. Obviously, no matter which $\mathcal{V}_i$ set that s_i chooses to collect data, s_i must be able to get data from the nodes in $\mathcal{V}_i^{\mathbf{O}}$. Thus, we have to select extra RPs from the nodes not in $\mathcal{V}_i^{\mathbf{O}}$. To do so, we define a weight ϖ_i^h for each $\mathcal{V}_i^h$ set by

$$\varpi_i^h = \min_{\forall r_j \in \mathcal{R}} \{\hat{D}(s_l, r_j)\}, \tag{7}$$

where $s_l \in \mathcal{V}_i^h - (\mathcal{V}_i^h \cap \mathcal{V}_i^{\mathbf{O}})$ such that $\hat{H}(s_l, s_i)$ is the minimum. Similarly, s_i selects the $\mathcal{V}_i^h$ set whose weight is the maximum. Then, for each of other $\mathcal{V}_i$ sets, we select the node $s_l \notin \mathcal{V}_i^{\mathbf{O}}$ that has the minimum hop count to s_i as an RP. Fig. 4(b) shows an example, where $\mathcal{C} = \{(s_3, \{s_4, s_5\}), (s_3, \{s_5, s_6\})\}$. So, we have $\mathcal{V}_3^1 = \{s_4, s_5\}$, $\mathcal{V}_3^2 = \{s_5, s_6\}$, and $\mathcal{V}_3^{\mathbf{O}} = \{s_5\}$. In this case, $\varpi_3^1 = \hat{D}(s_4, r_0) = 22$ and $\varpi_3^2 = \hat{D}(s_6, r_0) = 30$. Thus, s_3 chooses $\mathcal{V}_3^2$ to collect data, and s_4 will be an RP.

After selecting s_i and some nodes in its $\mathcal{V}_i$ sets as RPs, we conduct the following four actions: 1) Add these nodes to the solution set $\mathcal{R}$. However, if r_0 is s_i's parent, we need not add s_i to $\mathcal{R}$, as s_i can directly forward its data to r_0. 2) Mark the checked flags of s_i and each node in its $\mathcal{V}_i$ sets as 'true'. 3) Remove each element whose first tuple is s_i from $\mathcal{C}$. 4) Update $\mathcal{F}$ based on the rules defined in phase 1. In case of $\mathcal{F} = \emptyset$, which means that the checked flag of every node is true, we go to phase 4 to compute the traveling path. Otherwise, we return to phase 2 to find more RP candidates.

Remark 2 (The reasons of using a centroid). *In Eq. 6, we estimate the length of traveling path for the MS to visit all nodes in $\widetilde{\mathcal{V}}_i^h$ (if s_i does not collect data from $\mathcal{V}_i^h$). Let us consider the example in Fig. 4(a). Assume that s_3 selects $\mathcal{V}_3^1$ to collect data, so s_5 and s_6 (in $\widetilde{\mathcal{V}}_3^2$) have to act as RPs. Thus, the traveling path is $r_0 \to s_3 \to s_6 \to s_5 \to r_0$. On the other hand, when s_3 selects $\mathcal{V}_3^2$ to collect data, the path will be $r_0 \to s_3 \to s_4 \to r_0$. Comparing with these two paths, the differences are $s_6 \to s_5 \to r_0$ and $s_4 \to r_0$. Thus, the objective of Eq. 6 is to measure (and compare) the relative length of such a* differential *path in each $\mathcal{V}_i^h$ set. However, it incurs a high cost to compute an accurate length of the path, as we have to find the shortest path between every node in both $\mathcal{R}$ and $\widetilde{\mathcal{V}}_i^h$ and then use a TSP method to find a shortest path to visit all nodes in $\widetilde{\mathcal{V}}_i^h$. To solve the problem, we measure the distance between the centroid of $\widetilde{\mathcal{V}}_i^h$ and each node in $\mathcal{R}$. In fact, given a set $\widetilde{\mathcal{V}}_i^h$ of nodes, some clustering methods such as K-means have verified that if we want to compare the relative distances from two nodes s_i and s_j to $\widetilde{\mathcal{V}}_i^h$, it is efficient and accurate to directly compare the distances from $\widetilde{\mathcal{V}}_i^h$'s centroid respectively to s_i and s_j [40]. Therefore, the correctness of Eq. 6 in terms of distance comparison is justified. Besides, Eq. 6 also considers the number of nodes in $\widetilde{\mathcal{V}}_i^h$. In other words, when $\widetilde{\mathcal{V}}_i^h$ contains more nodes, the MS has to use a longer path to visit them. In this way, we can greatly save the computational cost of Eq. 6. Note that each node in $\widetilde{\mathcal{V}}_i^h$ is a one-hop neighbor of s_i, so the distance between any two nodes in $\widetilde{\mathcal{V}}_i^h$ is limited by $2l_c$, where l_c is the communication range of a sensor. Thus, we suggest setting $\sigma = l_c$ in Eq. 6.*

4.1.4 Phase 4–Path Scheduling

In the last phase, we adopt a local-search-based TSP heuristic to find a traveling path for the MS to visit RPs in $\mathcal{R}$. The heuristic takes computational time of $O(|\mathcal{R}|^3)$ and its detail can be found in [41].

We give a complete example in Fig. 3. Let us denote by $\mathcal{Q}$ the set of nodes whose checked flags are false. Then, the example has three iterations below.

Iteration 1: All tree nodes are included in $\mathcal{Q}$. In this case, we can derive that $\mathcal{F} = \{s_5, s_9, s_{10}, s_{11}\}$ and $\mathcal{C} = \{(s_3, \{s_4, s_5\}), (s_7, \{s_8, s_9, s_{10}\}), (s_7, \{s_{11}\})\}$. Thus, the solution set is $\mathcal{R} = \{r_0, s_7, s_{11}\}$.

Iteration 2: Since $\mathcal{Q} = \{s_1, s_2, s_3, s_4, s_5, s_6\}$, we have $\mathcal{F} = \{s_5, s_6\}$ and $\mathcal{C} = \{(s_3, \{s_4, s_5\}), (s_2, \{s_3, s_6\})\}$. In this iteration, we add s_3 to $\mathcal{R}$.

Iteration 3: $\mathcal{Q} = \{s_1, s_2, s_6\}$ and $\mathcal{F} = \{s_6\}$. So, we obtain that $\mathcal{C} = \{(s_1, \{s_2, s_6\})\}$. However, as r_0 is the parent of s_1, we do not add s_1 to $\mathcal{R}$. Thus, the final solution set $\mathcal{R}$ will be $\{r_0, s_3, s_7, s_{11}\}$ and the path is $r_0 \to s_{11} \to s_7 \to s_3 \to r_0$.

4.1.5 Discussion

We give the rationale of our EARTH algorithm. Phase 1 uses an SP tree to quickly find out RP candidates. In phase 2, two issues are addressed when selecting candidates. First, we want to better utilize the buffer of each RP without the occurrence of packet loss. Second, to reduce the traveling path, we prefer choosing the candidates close to r_0 (i.e., with fewer hop counts to the BS). Thus, we visit tree nodes in a bottom-up fashion and accumulate packets until a_i reaches buffer capacity B. Moreover, we also consider the combination of subtrees by DFS for each candidate to collect data. That is why the same candidate may have multiple elements in $\mathcal{C}$ (e.g., s_7 in Fig. 3). Then, phase 3 picks RP(s) from $\mathcal{C}$ based on the hop count or distance of each node to the closest RP in $\mathcal{R}$. In general, when a node s_i has a larger hop count to the closest RP, it implies that the sensors in s_i's subtrees (i.e., $\mathcal{V}_i$ sets) may have to relay their data to other RP(s) via more hops once s_i is not selected as an RP, thereby consuming more energy on multihop communications. Besides, Eqs. 6 and 7 select extra RPs closer to existing RPs in $\mathcal{R}$. Thus, there is a good possibility to reduce the traveling path, since the selected RPs have shorter distances between each other. Note that we pick just one candidate from $\mathcal{C}$ in each iteration of phase 3 (and go back to phase 2 to recompute candidates). The reason why we do not repeat phase 3 to select all RPs from $\mathcal{C}$ (until $\mathcal{C} = \emptyset$) is that when we identify RP(s), it actually conducts the action of tree pruning and changes the tree's topology. In this case, some candidates may have new subtrees to collect data. Thus, we return to phase 2 to find out these subtrees and change the set $\mathcal{C}$ to provide the updated information of RP candidates. Theorem 2 analyzes the time complexity of EARTH.

Theorem 2. *Given n sensors in $\mathcal{S}$, the EARTH algorithm requires computational time of $O(n^3)$ in the worst case.*

Proof: The EARTH algorithm has four phases. Phase 1 uses the Dijkstra's algorithm to find an SP tree to link all sensors in $\mathcal{S}$, which spends $O(n^2)$ time. In phase 2, we start from each node in $\mathcal{F}$ to traverse the tree, until $a_j \geq B$. The worst case occurs when each sensor produces one packet, so we need to visit either B or n nodes (the latter case occurs when $B > n$). Although phase 2 may be repeated multiple times, we observe that given the same starting node, we always get the same element(s) in $\mathcal{C}$. Thus, a smart design of phase 2 is to give a flag for each node in $\mathcal{F}$ to indicate whether we have used it to find candidates in $\mathcal{C}$. In this way, the worst case is that we use every node of $\mathcal{S}$ *once* as the starting node in the procedure of phase 2. Thus, the maximum time taken by phase 2 is $|\mathcal{S}| \times O(\min\{B, n\}) = O(n \times \min\{B, n\})$.

Phase 3 uses Eq. 5 to pick a node s_i from $\mathcal{C}$. Since we have to find the minimum hop count from s_i to the nearest RP, it takes $O(c|\mathcal{R}|)$ time, where c is the number of nodes in $\mathcal{C}$. Then, we find extra RPs from the branches of s_i's subtree by Eqs. 6 or 7. Since at most one RP will be found in each branch, it takes time of $O((\zeta - 1) \times |\mathcal{R}|)$, where ζ is the maximum node degree in the tree. A better design of phase 3 is to give a flag for each element in $\mathcal{C}$ to indicate whether we have checked it, and record its values in Eqs. 5, 6, and 7. Since $c \leq n$ and $|\mathcal{R}| \leq n$, phase 3 spends time of $O(c|\mathcal{R}|) + c \times O((\zeta - 1) \times |\mathcal{R}|) = O((\zeta - 1) \times n^2)$. Finally, phase 4 takes $O(|\mathcal{R}|^3)$ time as mentioned in Section 4.1.4. The worst case occurs when $\mathcal{R}$ has every node in $\mathcal{S}$ except for the only node linked to r_0. As $\mathcal{R}$ also contains $r_0 \notin \mathcal{S}$, phase 4 takes time of $O((n - 1 + 1)^3) = O(n^3)$. To sum up, the time complexity of EARTH is $O(n^2) + O(n \times \min\{B, n\}) + O((\zeta - 1) \times n^2) + O(n^3)$. Since $\zeta < n$, we can simplify the equation to $O(n^3)$, which proves the theorem. □

As discussed in Section 2.3, some methods like [14], [34] repeat the TSP method for $|\mathcal{R}|^2$ or n^2 times. When we apply the local-search-based TSP heuristic to these schemes, they inevitably incur very high time complexity of $O(n^5)$ in the worst case. By conducting such a time-consuming operation just once in phase 4, our EARTH algorithm can substantially reduce the computational time to only $O(n^3)$. Consequently, when sensors change their sensing rates, EARTH can flexibly compute a new traveling path for the MS to collect data, as

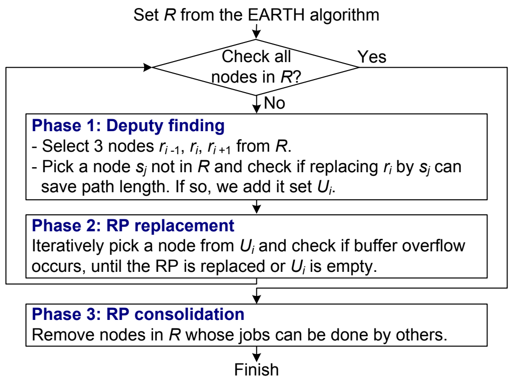

Set R from the EARTH algorithm
Check all nodes in R?
Yes
No
Phase 1: Deputy finding
- Select 3 nodes r_{i-1}, r_i, r_{i+1} from R.
- Pick a node s_j not in R and check if replacing r_i by s_j can save path length. If so, we add it set U_i.
Phase 2: RP replacement
Iteratively pick a node from U_i and check if buffer overflow occurs, until the RP is replaced or U_i is empty.
Phase 3: RP consolidation
Remove nodes in R whose jobs can be done by others.
Finish

Fig. 5: The flowchart of the eEARTH algorithm.

compared with the schemes in [14], [34].

4.2 The Enhanced EARTH (eEARTH) Algorithm

The EARTH algorithm finds a traveling path for the MS to collect data without packet loss. However, some RPs can be actually replaced or consolidated to further reduce the path. Suppose that all RPs in $\mathcal{R}$ have been sorted by their visiting sequence in EARTH. Fig. 5 gives the flowchart of the eEARTH algorithm, which contains three phases below.

1. *Deputy finding:* We select three RPs r_{i-1}, r_i, and r_{i+1} from $\mathcal{R}$. Then, for each node $s_j \in \mathcal{S} - \mathcal{R}$, it is a deputy of r_i if we can reduce the path by replacing r_i with s_j. We also create a set $\mathcal{U}_i$ to store such deputies.
2. *RP replacement:* We pick a node s_j from $\mathcal{U}_i$ and check if buffer overflow occurs at some RPs once we do the replacement. The above process is repeated until there is no buffer overflow or $\mathcal{U}_i = \emptyset$.
3. *RP consolidation:* Phases 1 and 2 are repeated until every (original) RP in $\mathcal{R}$ is checked. Then, for each node in the revised set $\mathcal{R}$, if its job can be done by another RP, we remove it from $\mathcal{R}$ to save path length.

4.2.1 Deputy Finding

We define the length of a path to visit three nodes s_a, s_b, and s_c in sequence by

$$\hat{P}(s_a, s_b, s_c) = \hat{D}(s_a, s_b) + \hat{D}(s_b, s_c). \tag{8}$$

Let $\mathcal{R} = \{r_0, r_1, r_2, \cdots, r_k\}$. Starting from $i = 1$, we find a set $\mathcal{U}_i$ of deputies that could replace r_i. For each node s_j in $\mathcal{S} - \mathcal{R}$, we check if $\hat{P}(r_{i-1}, r_i, r_{i+1}) > \hat{P}(r_{i-1}, s_j, r_{i+1})$. If so, it means that the traveling path can be reduced by replacing r_i with s_j. Thus, we add s_j to $\mathcal{U}_i$. The procedure is repeated until all nodes in $\mathcal{S} - \mathcal{R}$ are checked.

In case of $\mathcal{U}_i = \emptyset$, which implies that $r_{i-1} \to r_i \to r_{i+1}$ is an optimal segment in the path, we then check the next RP[3]. Otherwise, we sort all nodes in $\mathcal{U}_i$ by their values of $\hat{P}(r_{i-1}, s_j, r_{i+1})$ in an increasing order and go to phase 2.

4.2.2 RP Replacement

We then pick the first node in $\mathcal{U}_i$, say, s_j and check whether buffer overflow will occur at some RPs if we replace r_i by s_j. In particular, we use Eq. 4 to estimate the number of packets that an RP r_k should store, where $\mathcal{V}_k$ is the subtree rooted at r_k which will forward their data to r_k. Then, we consider two cases below.

- **Case A:** r_i is the original RP of s_j.
- **Case B:** s_j and r_i's parent have the same RP.

If there is no buffer overflow, we can replace r_i by s_j in $\mathcal{R}$. Otherwise, we check the next node in $\mathcal{U}_i$. The procedure is repeated until either $\mathcal{U}_i = \emptyset$ or we find a node in $\mathcal{U}_i$ without buffer overflow. Afterwards, we go back to phase 1.

Next, we discuss the two cases. For case A, if r_i is replaced by s_j, then the RP of r_i's parent, say, r_k has to collect *extra* data from the nodes in the set $(\mathcal{V}_i - \mathcal{V}_j) \cup \{r_i\}$, as r_i gives up the role of RP and the nodes in $\mathcal{V}_i$ but not in $\mathcal{V}_j$ need to send their data to r_k. Thus, we check whether

$$a_k + a_i - a_j \leq B. \tag{9}$$

When Eq. 9 holds, r_k has enough buffer space to store extra data, and it is safe to replace r_i by s_j. Note that r_i is the original RP of s_j, which means that $\mathcal{V}_j \subset \mathcal{V}_i$, so s_j's buffer will not overflow if it becomes an RP. Case A in Fig. 6 gives an example, where $B = 5$ and we check RP s_5. Since $\hat{P}(s_3, s_5, s_8) = 19$ and $\hat{P}(s_3, s_6, s_8) = 14$, s_6 is added to $\mathcal{U}_5$. Since s_5 is the original RP of s_6 and s_4 is the parent of s_5, we check whether s_4's RP (i.e., s_3) has enough buffer space by estimating $a_3 + a_5 - a_6 = (1 + 2) + (2 + 2 + 1) - (2 + 1) = 5 \leq B$. Thus, it is safe to replace s_5 by s_6 in $\mathcal{R}$.

For case B, there are two subcases to be addressed. In subcase 1, s_j is an ancestor node of r_i. If we replace r_i by s_j, s_j has to store the data not only collected by r_i (i.e., a_i) but also produced by the sensors in the set $\mathcal{V}_j - \mathcal{V}_i$. Thus, we check whether s_j's buffer will overflow by estimating

$$m_j + a_i + \sum\nolimits_{s_k \in \mathcal{V}_j - \mathcal{V}_i} m_k > B. \tag{10}$$

If not, it is safe to replace r_i by s_j. Case B in Fig. 6 shows an example, where we check RP s_7 and $\mathcal{U}_7 = \{s_3, s_4\}$. Here, s_3 is an ancestor node of s_7, so we compute $m_3 + a_7 + m_4 = 1 + 3 + 3 = 7 > B$. Thus, we cannot replace s_7 by s_3 due to buffer overflow. Subcase 2 occurs when s_j is not r_i's ancestor node. If s_j replaces r_i, then the RP, say, s_k of r_i's parent has to store r_i's collected data. However, s_k need not collect data from the sensors in $\mathcal{V}_j$, as s_j becomes an RP. Thus, we use Eq. 9 to check if s_k's buffer overflows. Let us continue the above example, where we check if s_4 can replace s_7. Since s_4 is not s_7's ancestor node, we check whether the RP of s_7's parent, which is s_2, will overflow its buffer. By estimating $a_2 + a_7 - a_4 = (1 + 1 + 3) + 3 - 3 = 5 \leq B$, s_7 can be safely replaced by s_4.

4.2.3 RP Consolidation

Finally, we check if it is possible to 'consolidate' some RPs to save path length. To do so, we pick each RP of $\mathcal{R}$, say, r_i in sequence. Let r_k be the RP of r_i's parent. If r_k's buffer will not overflow once it takes over the job of r_i, we can consolidate both r_i and r_k by removing r_i from $\mathcal{R}$. In particular, we estimate $a_k + a_i \leq B$ to determine whether buffer overflow occurs at s_k. Fig. 7 shows an example, where we check RP s_8. Since s_5 is its parent and also an RP, we compute $a_5 + a_8 = 2 + 2 = 4 \leq B$ and thus consolidate s_5 and s_8. The above procedure is repeated until every RP in $\mathcal{R}$ has been checked.

3. When $i = k$, we compute $\hat{P}(r_{k-1}, r_k, r_0)$ and $\hat{P}(r_{k-1}, s_j, r_0)$.

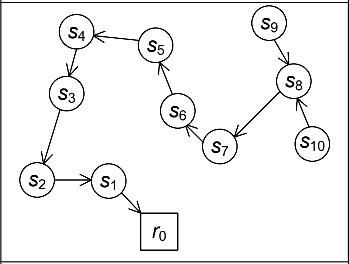
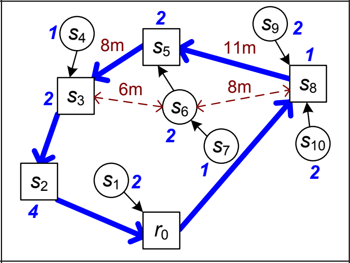
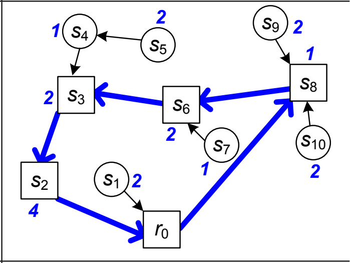
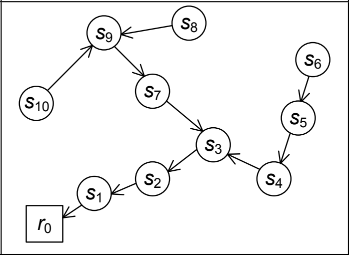
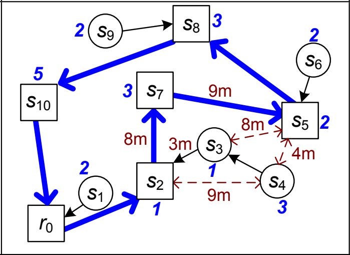
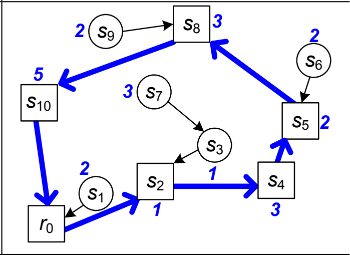
Case	SP tree	Before RP replacement	After RP replacement
Case A			
Case B			

Fig. 6: Examples of RP replacement, where squares indicate RPs in $\mathcal{R}$.

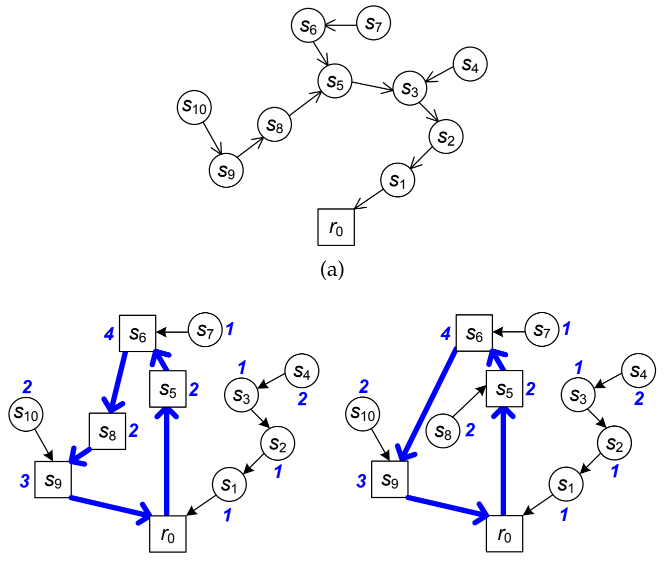


Fig. 7: An example of RP consolidation: (a) SP tree, (b) before RP consolidation, and (c) after RP consolidation.

4.2.4 Discussion

The eEARTH algorithm seeks to change the solution set $\mathcal{R}$ from EARTH to shorten the traveling path. To do so, eEARTH selects one RP r_i in $\mathcal{R}$, and checks if we can find a non-RP node s_j to replace r_i such that 1) path length is reduced and 2) it will not cause buffer overflow at other RPs. Thus, phase 1 uses Eq. 8 to find potential deputies of r_i to meet objective 1, while phase 2 selects the best deputy (to minimize path length) on the premise of objective 2. When r_i gives up the role of RP, it is possible that the RP of r_i's parent, say, r_k will encounter buffer overflow, as r_k is the only RP that should cache data from r_i and its subtree. Thus, we use two cases to examine r_k's buffer status when doing RP replacement. Both phases 1 and 2 are repeated until we check every RP in $\mathcal{R}$. Then, phase 3 removes some RPs from $\mathcal{R}$ to further reduce the traveling path, where an RP r_i can be removed once its adjacent RP has enough buffer space to collect extra data from r_i and its subtree. Theorem 3 proves that eEARTH can find a shorter traveling path than EARTH, while Theorem 4 analyzes its time complexity.

Theorem 3. *The eEARTH algorithm must reduce path length if it changes the solution set $\mathcal{R}$ from EARTH.*

Proof: Since phases 2 and 3 of eEARTH may change the solution set $\mathcal{R}$, we separate our proof into two parts. First, we show that phase 2 must reduce path length if it changes $\mathcal{R}$ by contradiction. Suppose that an RP $r_i \in \mathcal{R}$ is replaced by a non-RP node s_j in phase 2 such that the traveling path is increased. Let r_{i-1} and r_{i+1} be the previous and next RPs of r_i in $\mathcal{R}$, respectively. In eEARTH, we locally modify the path by replacing the segment $r_{i-1} \to r_i \to r_{i+1}$ with another segment $r_{i-1} \to s_j \to r_{r+1}$. As the path increases, we can derive that

$$\hat{D}(r_{i-1}, r_i) + \hat{D}(r_i, r_{i+1}) < \hat{D}(r_{i-1}, s_j) + \hat{D}(s_j, r_{i+1})$$
$$\Rightarrow \hat{P}(r_{i-1}, r_i, r_{i+1}) < \hat{P}(r_{i-1}, s_j, r_{i+1}),$$

which violates the rule in phase 1. Thus, a contradiction occurs since s_j will not be included in $\mathcal{U}_i$ by phase 1.

In phase 3, once the job of an RP r_i can be done by another RP, it will be removed from $\mathcal{R}$. In this case, the segment $r_{i-1} \to r_i \to r_{i+1}$ of the traveling path will be modified by $r_{i-1} \to r_{i+1}$. Obviously, we have $\hat{D}(r_{i-1}, r_{i+1}) < \hat{D}(r_{i-1}, r_i) + \hat{D}(r_i, r_{i+1})$ due to the property of triangle $\triangle_{r_{i-1}, r_i, r_{i+1}}$. Therefore, phase 3 must also reduce path length by removing r_i from $\mathcal{R}$. □

Theorem 4. *Given $\mathcal{R}$ from EARTH, the eEARTH algorithm takes computational time of $O(pq(B + \lg q))$ in the worst case, where $p = |\mathcal{R}|$ and $q = n - |\mathcal{R}|$.*

Proof: In eEARTH, we find $\hat{P}(r_{i-1}, r_i, r_{i+1})$ for each RP $r_i \in \mathcal{R}$, which takes $O(p)$ time as we have to estimate the lengths of p edges that form the traveling path by EARTH. In an iteration of phase 1, we find $\hat{P}(r_{i-1}, s_j, r_{i+1})$ for each non-RP node s_j. It spends $O(2q)$ time since we should compute $\hat{D}(r_{i-1}, s_j)$ and $\hat{D}(s_j, r_{i+1})$. For an iteration of phase 2, the worst case occurs when $\mathcal{U}_i$ contains all non-RP nodes (i.e., $\mathcal{U}_i = q$). It thus takes $O(q \lg q)$ time to sort $\mathcal{U}_i$. Since r_i and r_k belong to $\mathcal{R}$, we can know the amount of their accumulative data (i.e., a_i and a_k) from EARTH. Thus, what we need to find are a_j and $\mathcal{V}_j$ in Eqs. 9 and 10, respectively. Fortunately, we need to visit at most B nodes to find a_j and $\mathcal{V}_j$, since each node produces at least one packet of sensing data. Thus, each iteration of phase 2 spends time of $O(q \lg q + qB)$. Since we check every RP in $\mathcal{R}$ only once, phases 1 and 2 have p iterations. Therefore, these two phases totally spend time of $O(p) + O(p(2q + q \lg q + qB))$. Then, phase 3 checks if each RP's job can be taken over by another RP. As phases 1 and 2 keep the number of RPs in $\mathcal{R}$, phase 3 will check p RPs, which spends $O(p)$ time. Thus, eEARTH has time complexity of $O(p) + O(p(2q + q \lg q + qB)) + O(p) = O(pq(\lg q + B))$. □

Though a few methods discussed in Section 2.3 also seek to replace or combine some RPs to reduce the traveling path, there are two differences between our eEARTH algorithm and these methods in essence. First, none of them consider various sensing rates and buffer limitation, so these methods merely check whether path length can decrease when replacing RPs. On the contrary, eEARTH carefully checks the buffer statuses of upstream RPs by Eqs. 9 or 10 when selecting an RP to be replaced. Thus, it can prevent the upstream RPs from buffer overflowing. Second, as discussed in Section 4.1.5, several methods [14], [34] repeat the TSP method whenever doing RP replacement, which incur high complexity in computation. To address this issue, eEARTH locally amends the traveling path by iteratively looking over three adjacent RPs. Based on the triangle inequality, Theorem 3 proves that eEARTH is capable of saving path length. Since there is no need to invoke the TSP method, the time complexity of eEARTH can be greatly reduced, as presented in Theorem 4. These two differences distinguish eEARTH from existing methods, and justify its novelty.

5 SIMULATION STUDY

We develop a simulator in Java to evaluate the performance of EARTH and eEARTH algorithms. The sensing field is modeled by a $200 \times 200\,\mathrm{m}^2$ square, on which a number of sensors form a connected WSN and there is one MS to collect data. The communication range of a sensor is 20 m. Each sensor produces $[1, \alpha]$ packets of sensing data in a round, where packet length is 134 bytes. We use the energy model in Section 3.2 to estimate energy consumption of sensors on communications, where $\psi_1 = 50\,\mathrm{nJ/bit}$, $\psi_2 = 100\,\mathrm{pJ/bit}$ per m^2, $\psi_3 = 50\,\mathrm{nJ/bit}$, and $\varepsilon = 2$ in Eqs. 1 and 2. To evaluate buffer utilization of RPs, we define two metrics:

$$\beta_{\mathrm{avg}} = \frac{\sum_{r_i \in \mathcal{R}} \min\{a_i, B\}}{|\mathcal{R}| \times B} \tag{11}$$

$$\beta_{\mathrm{sd}} = \sqrt{\frac{1}{|\mathcal{R}|} \sum_{r_i \in \mathcal{R}} (\min\{a_i, B\} - \beta_{\mathrm{avg}})^2}, \tag{12}$$

where β_{avg} is the average ratio of buffer utilization, and β_{sd} is the standard deviation of the number of packets stored in each buffer. Note that when $a_i > B$, an RP r_i can store at most B packets in its buffer and other packets are dropped. That is why we use the term $\min\{a_i, B\}$ in Eqs. 11 and 12.

We compare our EARTH and eEARTH algorithms with the CB and WRP methods discussed in Section 2.3, which also select RPs to schedule the MS's traveling path. However, both CB and WRP require to indicate path length $L_{\max}$ in advance. Thus, we use EARTH and eEARTH to compute the traveling paths, and take their lengths as $L_{\max}$ for CB and WRP, which are respectively denoted by '_E' and '_e' in simulation figures. Below, we measure path lengths of EARTH and eEARTH, and then evaluate the effect of sensor number and buffer capacity on performance. Each experiment is repeated 1000 rounds, and we take their average.

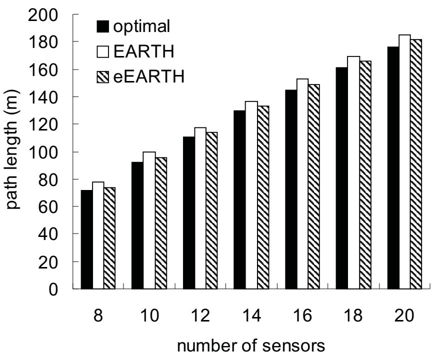


Fig. 8: Comparison on the average length of traveling path in a round.

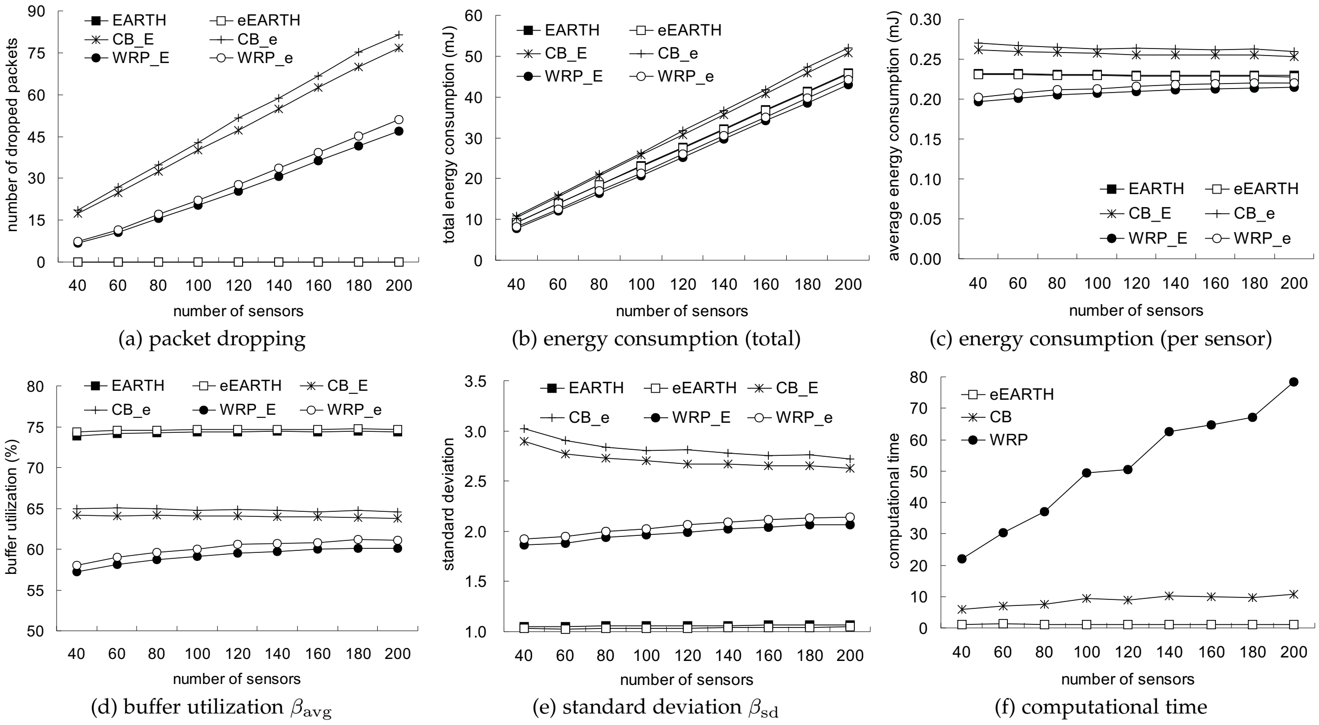
5.1 Path Length

We first compare the traveling paths computed by EARTH and eEARTH with the optimal solution. To do so, we use brute force to search all possible combinations of RPs and select the best one whose path length is the minimum and there is no buffer overflow. Since the brute force scheme takes very long time, we conduct the experiment in a small-scale WSN with 8 to 20 sensors. Fig. 8 gives the experimental result, where $\alpha = B = 5$. Obviously, as the number of sensors increases, we need to pick out more RPs to avoid buffer overflow. Thus, the traveling path increases accordingly. From Fig. 8, we observe that EARTH finds a traveling path whose length is slightly longer than the optimal one, while eEARTH can further reduce the path. On the average, EARTH and eEARTH increase around 6.3% and 3.0% of path length than the optimal solution, respectively, which verifies that they can find shorter traveling paths for the MS to efficiently collect data in the WSN without packet loss.

5.2 Effect of Sensor Number

By setting $\alpha = B = 5$, we investigate the performance of different methods with 40 to 200 sensors. Fig. 9(a) presents the result of packet loss, where both EARTH and eEARTH will not drop any packet. Since eEARTH finds a traveling path with smaller $L_{\max}$ value than EARTH, CB_e and WRP_e discard more packets than CB_E and WRP_E, respectively. As CB arbitrarily groups sensors into clusters and selects RPs accordingly, it loses more packets than WRP which selects RPs based on node degree and hop count. From Fig. 9(a), we observe that even though CB and WRP have the same path length with EARTH/eEARTH, they still encounter serious packet loss. This phenomenon becomes more significant when the number of sensors grows. The reason is that CB and WRP do not consider diverse sensing rates and buffer limitation, so these two methods inevitably force some RPs to collect a large amount of data and thereby overflow their buffers. On the contrary, our EARTH and eEARTH algorithms check whether each RP has enough buffer space to cache the collected sensing data, so they will incur no packet loss.

Then, we study energy consumption of sensors on communications, as shown in Fig. 9(b) and (c). When there are more sensors, some sensors have to send their data to RPs via more hops. Besides, most RPs may collect data from more



(a) packet dropping (b) energy consumption (total) (c) energy consumption (per sensor)

(d) buffer utilization β_{avg} (e) standard deviation β_{sd} (f) computational time

Fig. 9: Effect of the number of sensors on the performance of different methods in a round.

sensors. That is why the amount of total energy consumption increases when the number of sensors grows in Fig. 9(b). On the other hand, since the area of sensing field is fixed, the average distance between any two adjacent sensors will decrease when there are more sensors in the sensing field. By Eq. 1, a sensor can spend less energy to transmit data to its neighbor with a shorter distance. Thus, the amount of energy consumption per sensor slightly decreases as the number of sensors increases in Fig. 9(c). From the two figures, we observe that CB makes sensors spend the most amount of energy, as it may form some large groups of sensors. In this case, some sensors need to send their data to farther RPs and waste more energy. By considering both hop count and distance of each sensor to the nearest RP, EARTH and eEARTH result in energy consumption quite close to WRP. Moreover, EARTH/eEARTH further save around 10% of energy than CB, which shows their high energy-efficiency on sensors' communications.

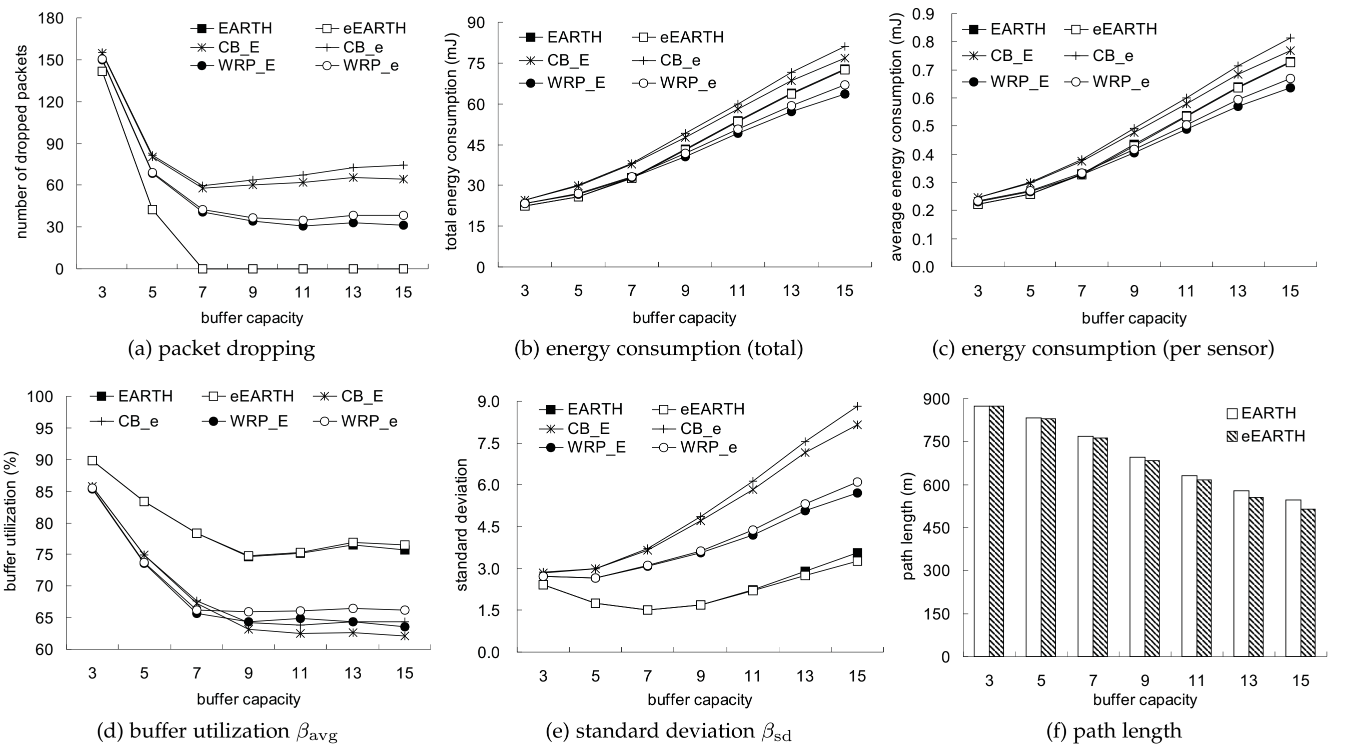
Both Fig. 9(d) and (e) evaluate buffer utilization of different methods. Obviously, a larger β_{avg} value indicates that each RP can better utilize its buffer to collect data. On the other hand, a lower β_{sd} value implies that the workloads of RPs are balanced, so the possibility of buffer overflow could decrease accordingly. Since CB may group sensors into different sizes of clusters, it has the largest β_{sd} value and thus would make some RPs easily overflow their buffers (referring to Fig. 9(a)). On the other hand, WRP selects RPs based on their weights, and it has smaller β_{avg} and β_{sd} values than CB. Our proposed algorithms always have the highest β_{avg} and the lowest β_{sd} values among all methods, which verifies that EARTH/eEARTH can allow RPs to well utilize buffers and also balance their workloads.

Afterwards, we measure the computational overhead required by different methods. In this experiment, we take the computational time of EARTH as one unit. The computational time of eEARTH also includes that of EARTH. Fig. 9(f) gives the experimental result. By Theorem 4, eEARTH takes extra time of $O(pq(B + \lg q))$ to do the improvement in traveling path, where $p + q = n$. Thus, it is expected that eEARTH increases very less computational overhead. On the other hand, since CB iteratively groups sensors into clusters and finds RPs accordingly, it averagely requires $6 \sim 10.7$ times of computational overhead than EARTH. For WRP, as it adopts the TSP method to recompute the traveling path whenever new RPs are found in each iteration, the computational time grows very fast when there are more sensors. In particular, WRP spends 22 to 78.4 times of computational overhead than EARTH when the number of sensors grows from 40 to 200. This experiment shows that both EARTH and eEARTH are computation-efficient and they can thus fast find a new traveling path when the MS's mission changes (e.g., the sensing rates of some sensors are changed).

5.3 Effect of Buffer Capacity

We evaluate how buffer capacity B affects the performance of EARTH, eEARTH, CB, and WRP methods. In the following experiments, the number of sensors is 100 and $\alpha = 7$. Fig. 10(a) shows the number of packets dropped by each method. Obviously, since each sensor produces $[1, 7]$ packets in a round, all methods encounter packet loss when $B < 7$. However, even though some sensors may produce more packets than buffer capacity, EARTH and eEARTH seek to find more RPs to store these packets. Thus, they always result in the lowest packet loss than other methods. When $B \geq 7$, our algorithms can adaptively select RPs without buffer overflow. Interestingly, both CB and WRP keep dropping around 60–70 and 30–40 packets when $B \geq 7$, respectively, which means that increasing buffer capacity cannot alleviate their packet loss. That is because these two methods do not take buffer capacity into consideration when selecting RPs.

Fig. 10(b) and (c) present total and average energy consumption of sensors, respectively. Since the number of sensors is fixed to 100, their results will be similar, except for the scale in the y-axis. When buffer capacity increases, each RP is



(a) packet dropping (b) energy consumption (total) (c) energy consumption (per sensor)

(d) buffer utilization β_{avg} (e) standard deviation β_{sd} (f) path length

Fig. 10: Effect of buffer capacity B on the performance of different methods in a round.

capable of storing data from more sensors. Thus, the number of RPs may decrease and sensors would have to relay their data to farther RPs, thereby consuming more energy. That is why all methods spend more energy when buffer capacity grows. Similar to Fig. 9(b) and (c), our EARTH and eEARTH algorithms spend slightly more energy than WRP but can save nearly 10% of energy than CB, which demonstrates their effectiveness on energy consumption.

Fig. 10(d) and (e) give buffer utilization of RPs. In general, β_{avg} decreases while β_{sd} increases when B grows, as each RP has larger buffer space to store data. Thus, some RPs may leave more buffer space, which also increases the standard deviation of buffer utilization. The experimental results show that both EARTH and eEARTH always result in the largest β_{avg} and the smallest β_{sd} under different values of B, which verifies that they can improve buffer utilization and also balance workloads of the selected RPs.

Finally, we observe how buffer capacity affects the traveling paths found by EARTH and eEARTH, as shown in Fig. 10(f). As mentioned earlier, the number of RPs reduces when buffer capacity increases, since each RP can cache data from more sensors. In this case, path length will decrease accordingly. In Fig. 10(f), path length decreases more significantly when $B \geq 7$, because no sensors will produce data more than buffer capacity. Moreover, eEARTH can find more deputies to replace RPs found by EARTH as B grows, thereby further shortening the traveling path. In particular, eEARTH reduces 0.7% to 5.5% of path length than EARTH, when B increases from 7 to 15.

6 Conclusion

WSNs with static sinks are vulnerable to the energy hole problem due to imbalanced energy consumption of sensors. Using an MS to visit a set of RPs selected from sensors to collect sensing data can efficiently conquer the problem. However, existing solutions usually assume that sensors have an equal sensing rate and sufficiently large buffer space. In practice, they will inevitably encounter packet loss caused by buffer overflow at some RPs when these assumptions are unavailable. Therefore, we propose the EARTH algorithm to pick out RPs and schedule the MS's traveling path with the consideration of diverse sensing rates and limited buffers. EARTH constructs an SP tree in the WSN and finds RPs based on the amount of accumulative data, hop counts, and distances of sensors. Moreover, an improved version, eEARTH, is also developed by replacing and consolidating RPs on the basis of triangular property. Through complexity analysis, we show that both EARTH and eEARTH require less computational overhead. In addition, extensive simulation results verify that our proposed algorithms not only incur less packet loss but also save sensors' energy on communications and better utilize their buffers, as compared with both CB and WRP methods.

For the future work, we will study how to adaptively adjust the round time T. In particular, a short round may result in the increase of overheads on communication and computation. On the other hand, a long round could cause a large delay in the data collection procedure. Therefore, it deserves further investigation on how to set some factors such as the MS's movement speed and the buffer size of each sensor to get a proper value of T, so as to balance between the latency of data collection and energy consumption of sensors on communications.

References

[1] I. Khan, F. Belqasmi, R. Glitho, N. Crespi, M. Morrow, and P. Polakos, "Wireless sensor network virtualization: a survey," *IEEE Commu. Surveys & Tutorials*, vol. 18, no. 1, pp. 553–576, 2016.

[2] S.C. Hu, Y.C. Wang, C.Y. Huang, and Y.C. Tseng, "Measuring air quality in city areas by vehicular wireless sensor networks," *J. Systems and Software*, vol. 84, no. 11, pp. 2005–2012, 2011.

[3] M. Janidarmian, A.R. Fekr, K. Radecka, and Z. Zilic, "Multi-objective hierarchical classification using wearable sensors in a health application," *IEEE Sensors J.*, vol. 17, no. 5, pp. 1421–1433, 2017.

[4] Y.C. Wang, "Mobile sensor networks: system hardware and dispatch software," *ACM Computing Surveys*, vol. 47, no. 1, pp. 12:1–12:36, 2014.

[5] P. Tokekar, J.V. Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture," *IEEE Trans. Robotics*, vol. 32, no. 6, pp. 1498–1511, 2016.

[6] Y.C. Wang and C.C. Yang, "3S-cart: a lightweight, interactive sensor-based cart for smart shopping in supermarkets," *IEEE Sensors J.*, vol. 16, no. 17, pp. 6774–6781, 2016.

[7] N.A. Pantazis, S.A. Nikolidakis, and D.D. Vergados, "Energy-efficient routing protocols in wireless sensor networks: a survey," *IEEE Comm. Surveys & Tutorials*, vol. 15, no. 2, pp. 551–591, 2013.

[8] J. Ren, Y. Zhang, K. Zhang, A. Liu, J. Chen, and X.S. Shen, "Lifetime and energy hole evolution analysis in data-gathering wireless sensor networks," *IEEE Trans. Industrial Informatics*, vol. 12, no. 2, pp. 788–800, 2016.

[9] Y.C. Wang, F.J. Wu, and Y.C. Tseng, "Mobility management algorithms and applications for mobile sensor networks," *Wireless Comm. and Mobile Computing*, vol. 12, no. 1, pp. 7–21, 2012.

[10] Y. Gu, F. Ren, Y. Ji, and J. Li, "The evolution of sink mobility management in wireless sensor networks: a survey," *IEEE Comm. Surveys & Tutorials*, vol. 18, no. 1, pp. 507–524, 2016.

[11] M. Mathew, N. Weng, and L.J. Vespa, "Quality-of-information modeling and adapting for delay-sensitive sensor network applications," *Proc. IEEE Int'l Performance Computing and Comm. Conf.*, 2012, pp. 471–477.

[12] Y.C. Wang and C.T. Wei, "Lightweight, latency-aware routing for data compression in wireless sensor networks with heterogeneous traffics," *Wireless Comm. and Mobile Computing*, vol. 16, no. 9, pp. 1035–1049, 2016.

[13] D.L. Applegate, R.E. Bixby, V. Chvatal, and W.J. Cook, *The traveling salesman problem: a computational study*, Princeton, NJ, USA: Princeton University Press, 2007.

[14] G. Xing, T. Wang, Z. Xie, and W. Jia, "Rendezvous planning in wireless sensor networks with mobile elements," *IEEE Trans. Mobile Computing*, vol. 7, no. 12, pp. 1430–1443, 2008.

[15] P. Castiglione and G. Matz, "Energy-neutral source-channel coding with battery and memory size constraints," *IEEE Trans. Comm.*, vol. 62, no. 4, pp. 1373–1381, 2014.

[16] S. Padakandla, K.J. Prabuchandran, and S. Bhatnagar, "Energy sharing for multiple sensor nodes with finite buffers," *IEEE Trans. Comm.*, vol. 63, no. 5, pp. 1811–1823, 2015.

[17] W. Zhao, M. Ammar, and E. Zegura, "Controlling the mobility of multiple data transport ferries in a delay-tolerant network," *Proc. IEEE INFOCOM*, 2005, pp. 1407–1418.

[18] R. Moazzez-Estanjini, J. Wang, and I.C. Paschalidis, "Scheduling mobile nodes for cooperative data transport in sensor networks," *IEEE/ACM Trans. Networking*, vol. 21, no. 3, pp. 974–989, 2013.

[19] Y.C. Wang, W.C. Peng, and Y.C. Tseng, "Energy-balanced dispatch of mobile sensors in a hybrid wireless sensor network," *IEEE Trans. Parallel and Distributed Systems*, vol. 21, no. 12, pp. 1836–1850, 2010.

[20] Y.C. Wang, "A two-phase dispatch heuristic to schedule the movement of multi-attribute mobile sensors in a hybrid wireless sensor network," *IEEE Trans. Mobile Computing*, vol. 13, no. 4, pp. 709–722, 2014.

[21] D.J. Abraham, K. Cechlarova, D.F. Manlove, and K. Mehlhorn, "Pareto optimality in house allocation problems," *Proc. Int'l Conf. Algorithms and Computation*, 2005, pp. 1163–1175.

[22] B. Yuan, M. Orlowska, and S. Sadiq, "On the optimal robot routing problem in wireless sensor networks," *IEEE Trans. Knowledge and Data Engineering*, vol. 19, no. 9, pp. 1252–1261, 2007.

[23] J. Tang, H. Huang, S. Guo, and Y. Yang, "Dellat: delivery latency minimization in wireless sensor networks with mobile sink," *J. Parallel and Distributed Computing*, vol. 83, pp. 133–142, 2015.

[24] R. Sugihara and R.K. Gupta, "Optimal speed control of mobile node for data collection in sensor networks," *IEEE Trans. Mobile Computing*, vol. 9, no. 1, pp. 127–139, 2010.

[25] L. He, J. Pan, and J. Xu, "A progressive approach to reducing data collection latency in wireless sensor networks with mobile elements," *IEEE Trans. Mobile Computing*, vol. 12, no. 7, pp. 1308–1320, 2013.

[26] E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," *New Results and New Trends in Computer Science*, vol. 555, pp. 359–370, 1991.

[27] A. Chakrabarti, A. Sabharwal, and B. Aazhang, "Communication power optimization in a sensor network with a path-constrained mobile observer," *ACM Trans. Sensor Networks*, vol. 2, no. 3, pp. 297–324, 2006.

[28] S. Gao, H. Zhang, and S.K. Das, "Efficient data collection in wireless sensor networks with path-constrained mobile sinks," *IEEE Trans. Mobile Computing*, vol. 10, no. 4, pp. 592–608, 2011.

[29] M. Ma and Y. Yang, "SenCar: an energy-efficient data gathering mechanism for large-scale multihop sensor networks," *IEEE Trans. Parallel and Distributed Systems*, vol. 18, no. 10, pp. 1476–148, 2007.

[30] G. Xing, T. Wang, W. Jia, and M. Li, "Rendezvous design algorithms for wireless sensor networks with a mobile base station," *Proc. ACM Int'l Symp. Mobile Ad Hoc Networking and Computing*, 2008, pp. 231–240.

[31] M. Zhao and Y. Yang, "Bounded relay hop mobile data gathering in wireless sensor networks," *IEEE Trans. Computers*, vol. 61, no. 2, pp. 265–277, 2012.

[32] M. Ma, Y. Yang, and M. Zhao, "Tour planning for mobile data-gathering mechanisms in wireless sensor networks," *IEEE Trans. Vehicular Technology*, vol. 62, no. 4, pp. 1472–1483, 2013.

[33] K. Almiani, A. Viglas, and L. Libman, "Energy-efficient data gathering with tour length-constrained mobile elements in wireless sensor networks," *Proc. IEEE Local Computer Network Conf.*, 2010, pp. 582–589.

[34] H. Salarian, K.W. Chin, and F. Naghdy, "An energy-efficient mobile-sink path selection strategy for wireless sensor networks," *IEEE Trans. Vehicular Technology*, vol. 63, no. 5, pp. 2407–2419, 2014.

[35] Z.M. Wang, S. Basagni, E. Melachrinoudis, and C. Petrioli, "Exploiting sink mobility for maximizing sensor networks lifetime," *Proc. IEEE Annual Hawaii Int'l Conf. System Sciences*, 2005, pp. 1–9.

[36] Y.C. Wang, Y.Y. Hsieh, and Y.C. Tseng, "Multiresolution spatial and temporal coding in a wireless sensor network for long-term monitoring applications," *IEEE Trans. Computers*, vol. 58, no. 6, pp. 827–838, 2009.

[37] D. Wu, B. Yang, H. Wang, D. Wu, and R. Wang, "An energy-efficient data forwarding strategy for heterogeneous WBANs," *IEEE Access*, vol. 4, pp. 7251–7261, 2016.

[38] F. Wang, S. Wu, K. Wang, and X. Hu, "Energy-efficient clustering using correlation and random update based on data change rate for wireless sensor networks," *IEEE Sensors J.*, vol. 16, no. 13, pp. 5471–5480, 2016.

[39] Y.C. Wang and G.W. Chen, "Efficient data gathering and estimation for metropolitan air quality monitoring by using vehicular sensor networks," *IEEE Trans. Vehicular Technology*, vol. 66, no. 8, pp. 7234–7248, 2017.

[40] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Cambridge: Academic Press, 2001.

[41] W. Zhang, "Depth-first branch-and-bound versus local search: a case study," *Proc. National Conf. Artificial Intelligence*, 2000, pp. 930–935.