Multicast Scheduling Algorithms for Battery-Free Wireless Sensor Networks

Bingkun Yao
Harbin Institude of Technology
Harbin, China
bingkunyao@stu.hit.edu.cn

Hong Gao
Harbin Institude of Technology
Harbin, China
honggao@hit.edu.cn

Jianzhong Li
Harbin Institude of Technology
Harbin, China
lijzh@hit.edu.cn

***Abstract*—Currently, a new type of wireless sensor network (WSN) named as battery-free network (BF-WSN), has been proposed and widely studied. Compared with traditional battery-powered WSN (BP-WSN), nodes in BF-WSN can harvest energy from ambient environment, prolonging the lifetime of the network greatly. Multicast is an important way for data dissemination in WSNs. The problem of minimum latency multicast scheduling (MLMS) that seeks a fast schedule without collision for data multicast has been studied extensively in BP-WSNs. However, existing algorithms are not suitable in BF-WSNs. In this paper we study the MLMS problem in BF-WSNs (BF-MLMS). To reduce latency, we investigate how to compute the end-to-end transmission delay. By considering both energy supply and collision, we propose centralized and distributed algorithms for constructing collision-free multicast trees in BF-WSNs. To the best of our knowledge, this is the first work to consider the BF-MLMS problem. Simulation results verify our protocols have high performance in terms of multicast latency and message volume.**

***Index Terms*—Multicast, scheduling algorithms, minimum-latency, battery-free wireless sensor networks (BF-WSNs).**

I. Introduction

Wireless Sensor Networks (WSNs) are significant components of the Internet of Things (IoTs). Lifetime of WSNs powered by batteries (BP-WSNs) is limited because it is impossible to replace batteries in many cases (e.g. In forests, volcanos). To solve this problem, Battery-Free Wireless Sensor Networks (BF-WSNs) have been proposed in recent years [1]. Nodes in BF-WSNs can harvest ambient energy to support its work (e.g. Solar power [2], RF energy [3] and vibration). For example, [4] presents a prototype of BF-node that is able to capture both solar and RF energy. Although the lifetime of BF-WSN can be infinite in thoery [5], recharge rate is usually very low and varies dramatically among nodes. Hence, there are still many challenging problems for designing BF-WSNs.

Multicast is a fundamental data delivery pattern whose task is transmiting packets from the sink node to a subset of all the nodes [11]. For some real-time applications, a fast and collision-free scheduling for multicast is highly required. Algorithms for multicast has been well studied in BP-WSNs [6]- [20]. They mainly focus on constructing an efficient multicast tree spanning all distination nodes, or reducing the number of transmitted packets under a delay threshold.

For BF-WSNs, nodes need to gather enough energy to support their operations (i.e., sending or receiving packets, data processing). Unlike traditional networks, when there is not adequate ambient energy (e.g. Cloudy days for solar-energy platforms), it may take a long time to get nodes recharged, which is the main cause for transmission latency. Protocols for BP-WSNs have not considered energy harvesting and relay nodes with low recharge rate may leads to a huge latency. Therefore, existing algorithms for BP-WSNs are not applicable in BF-WSNs.

To address these issues, we investigate MLMS problem in multi-hop BF-WSNs (BF-MLMS). To the best of our knowledge, this problem has not been studied yet. Our main contributions are as follows:

(1) We formalize the problem of BF-MLMS and prove it to be **NP-Hard**.
(2) We discuss how to compute end-to-end latency in BF-WSNs and propose a dynamic-programming centralized algorithm. Then an edge-based distributed algorithm is presented.
(3) Extensive simulations are carried out and the results verify both of our algorithms have high performance on multicast latency.

The rest of paper is organized as follows. Section II surveys related works. Section III presents network structure and problem difinition. In Section IV and V, we describe both algorithms in detail. Section VI shows simulation results. Finally, Section VII concludes the paper.

II. Related works

Multicast in BP-WSNs has been studied extensively. [11], [16], [21] and [23] focused on energy-efficient multicast problems. Besides, [13], [18], [19] and [22] investigated the problem of jointly optimization of energy consumption and delay, seeking a balance between them.

As a special case for the problem of MLMS, the problem of MLBS (i.e. Minimum Latency Broadcast Scheduling) has received significant attention from researchers. In always-awake WSNs it was proved to be NP-Hard in [6] and studied in [6]- [9]. Besides, [10], [12], [14] and [17] investigated the MLBS problem in duty-cycled networks.

In BF-WSNs, energy saving is not an essential requirement when designing protocols since nodes can harvest ambient energy. [24]- [26] investigated energy-adaptive routing in BF-WSNs. [27] and [28] focused on joint mobile data gathering and energy replenishment. Online broadcast strategies were explored by [29] with assuming harvestable energy process as an independent and identically distributed (i.i.d) process.

Unfortunately, the BF-MLMS problem has not been studied yet. In terms of existing works on transmission for BF-WSNs, [24]- [28] did not consider multicast latency, [29] focused on one-hop, rather than multi-hop networks, which are not appropriate for solving the problem of the BF-MLMS.

Also, the above mentioned works for BP-WSNs can not be used to solve the problem of BF-MLMS for the following reasons:

(1) Works on flooding in BP-WSNs do not take transmission collision into consideration and they are not suitable for BF-WSNs.
(2) Works for solving the problem of MLMS in BP-WSNs do not consider energy harvesting so they can not be applied to BF-WSNs.

III. Problem Definition

A. Network and Energy Models

We consider a multi-hop BF-WSN G = (V,E). V = V' $\cup$ $\{s\}$, where s denotes the sink and V' denotes the set of all BF-nodes. Node u and v are neighbors and can communicate with each other directly if there exists an edge *(u,v)* $\in$ E between them. Each node's working period is divided into timeslots with length of τ, which is enough to complete a transmission successfully. For every node $v \in V$', let $B_v[k]$ and $EH(v)$ denote the residual energy at the beginning of timeslot k and recharge rate (*i.e.*, energy harvested in k-th timeslot), respectively. Sink s is assumed to be energy-abundant and able to transmit message at any timeslot. Nodes are recharged through wireless energy transmitters, which is much stabler than other sources so $EH[v]$ can be predicted precisely within a period of time (*e.g.* 20 minutes) [30]. We also assume the network is synchronized.

Most energy consumed in WSNs is for data transmission. Thus, we omit other types of expenditure (*e.g.* data processing) but they are easy to be integrated into our model. A node is able to work only if it has enough energy to support at least one transmission. Let e_s and e_r denote energy needed for sending and receiving one packet, respectively. Then at the start of k-th timeslot the remaining energy of node v is updated by:

$$B_v[k] = \begin{cases} B_v[k-1] + EH(v) - e_r \\ \qquad \textit{If } B_v\textit{[k-1]} \geq e_r \textit{ and v receives a packet} \\ B_v[k-1] + EH(v) - e_s \\ \qquad \textit{If } B_v\textit{[k-1]} \geq e_s \textit{ and v sends a packet} \\ B_v[k-1] + EH(v) \\ \qquad \textit{otherwise} \end{cases}$$

B. Problem Definition

Given a multi-hop BF-WSN G = (V,E), the sink node s needs to deliver messages to all target nodes in the distination set $\mathcal{D}$ $\subseteq$ V' with the minimum latency. Thus, we need to construct a tree spanning all destination nodes in $\mathcal{D}$ firstly, and then generate corresponding transmission schedule. Let $p(u)$ and $ch(u)$ denote the parent and children set of u. $NB(u)$ denotes u's neighbouring nodes. The schedule of node u is represented by $Sch(u)$ = (u, $p(u)$, t), which means node u is due to receive the multicast packet from its parent $p(u)$ at timeslot t.

Executable schedule for multicast in BF-networks must be free of **Time Collision** and **Energy Collision** [31], which are explained in detail below.

(1) Time Collision: In protocol interference model, if a node hears more than one messages simultaneously, it will fail to receive any of them due to interference. Thus, the schedule is free of **Time Collision** if and only if for any pair of nodes (u, v) $\in$ V and their schedules $Sch(u)$ = (u, $p(u)$, t_1) and $Sch(v)$ = (v, $p(v)$, t_2), one of the following conditions must be satisfied: (a) $t_1 \neq t_2$; (b) $t_1 = t_2$, $u \notin NB(p(v))$ and $v \notin NB(p(u))$.
(2) Energy Collision: For example, in Fig.1(b), there are three transmission schedules: $Sch(n_2)$ = (n_2, n$_1$, 8), $Sch(n_3)$ = (n_3, n$_1$, 10) and $Sch(n_4)$ = (n_4, n$_1$, 10) but they can not execute together for the following reasons. Assume $Sch(n_3)$ and $Sch(n_4)$ execute at timeslot 10, e_s = 100 and e_r = 80. Without $Sch(n_2)$, $B_{n_1}[10]$ = 145 $>e_s$, which is enough for n_1 to complete the transmission at timeslot 10. Otherwise, since n_1 is due to transmit at timeslot 8 before timeslot 10, $B_{n_1}[10]$ changes to 45 = 145-e_s $<e_s$, making both $Sch(n_3)$ and $Sch(n_4)$ non-executable. Therefore, we have following definition for **energy collision**.

Definition 1. ***(Energy Collision)*** *Schedules Sch(u)=(u, p(u) ,t_1) and Sch(v)=(v, p(v) ,t_2) have energy collision if the following two conditions satisfy: (a) p(u) = p(v) or p(u) = v or p(v) = u; (b)Sch(v) can not execute for energy constraints if Sch(u) execute.*

Energy collision can be classified into two types in the problem of BF-MLMS, which are illustrated in Fig.1. In Fig.1(a), $Sch(n_2)$, $Sch(n_3)$, $Sch(n_4)$ and $Sch(n_5)$ are not compatible. Without $Sch(n_2)$, $B_{n_2}[12]$ = 120 $>e_s$. On the contrary, $B_{n_2}[12]$ will change to 40 $<e_s$ since n_2 is due to receive message at timeslot 10 before timeslot 12. In this case, node n_3, n_4 and n_5 can not be scheduled to receive at timeslot 12. Since n_2's sending schedule conflict with its receiving, we call this kind of conflict **send-after-receive**. Similarly, In Fig.1(b), since some sending schedules of n_1 can not execute together, we call this type of collision **send-after-send**. Obviously, we must deal with energy collision carefully when designing algorithms.

We expect a collision-free schedule with the minimum **multicast latency**, which is defined as below:

Definition 2. ***(multicast latency)*** *Given a multicast tree T = (V_t,E_t) spanning all the nodes in $\mathcal{D}$ and the corresponding transmission schedule $\mathcal{S} = \{Sch(v) | for\ each\ v \in V_t \setminus \{s\}\}$, the multicast latency is defined as the number of timeslots needed for all the distination nodes to receive the packet from the sink node s. (i.e. Max$\{sch(v)[t] | v \in \mathcal{D}\}$)*

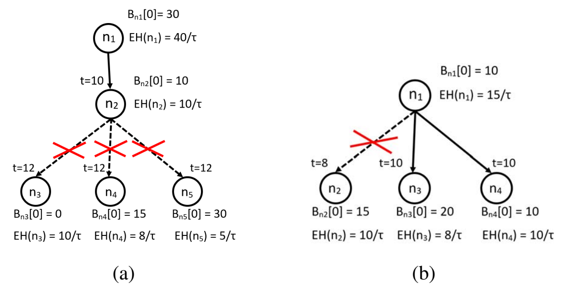


Fig. 1. Examples of energy collision in the problem of BF-MLMS.

The MLMS problem for BF-WSNs (BF-MLMS) is defined as below:

Input: A BF-WSN $G = (V,E)$, where $V = V' \cup \{s\}$; The set of target nodes $\mathcal{D} \subseteq V'$; The initial energy $B_v[0]$ and recharge rate $EH[v]$ for every node $v \in V'$.

Output: A multicast tree $T = (V_t,E_t)$ rooted at s spanning all the nodes in $\mathcal{D}$, which is a subgraph of G; Schedule $\mathcal{S} = \{sch(v) | for\ each\ v \in V_t \setminus \{s\}\}$ satisfying the following three conditions:

(1) For every $v \in V_t \backslash \{s\}$, $B_v[Sch(v)[t]] \geq e_r$ and $B_{p(u)}[Sch(v)[t]] \geq e_s$.
(2) The schedule $\mathcal{S}$ is both time-collision-free and energy-collision-free.
(3) The multicast latency is minimized.

Theorem 1. *The BF-MLMS problem is NP-Hard.*

Proof. When $\mathcal{D} = V \backslash \{s\}$ and for each node $v \in V$, $B_v[0] > e_s$ and $EH[v] > e_s$, BF-MLMS is equivalent to the MLBS problem, which has been proved to be NP-Hard in [6]. Therefore, the BF-MLMS problem is also NP-Hard. □

IV. Centralized Dynamic programming Algorithm

In this section, we first focus on the scenario when $|\mathcal{D}| = 1$. Based on this, we propose a dynamic-programming algorithm to generate multicast tree and node's schedule simultaneously for multiple destination nodes.

A. Compute End-to-end Delay

when $|\mathcal{D}| = 1$, the BF-MLMS problem is equivalent to finding a minimum-time path from source s to the only target node d. We use $\boldsymbol{L(u, v, t)}$ to denote the forwarding delay from node u to node v if u receives packet at timeslot t. In BF-WSNs, both u and v have to wait for recharging if either of them does not have enough energy at timeslot t. Thus, we have:

$$L(u,\ v,\ t) = \begin{cases} 1 \quad \textit{If } B_u[0] + EH(u) * t - e_s - e_r \geq 0 \\ \quad \textit{and } B_v[0] + EH(v) * t - e_r \geq 0 \\ Max\,\{ \\ ceil((e_s - (B_u[0] + t * EH(u) - e_r))/EH(u)) \\ \quad + 1, \\ ceil((e_r - (B_v[0] + t * EH(v)))/EH(v)) + 1\} \\ \quad \textit{Otherwise} \end{cases}$$

Then, the minimum latency from source s to node v, denoted by $\boldsymbol{T(s,\ v)}$, could be computed by following theorem:

Theorem 2. $T(s,\ v) = Min\{T(s,\ u) + L(u,v,T(s,\ u)) | u \in NB(v)\}$.

Proof. We prove this by contradiction. Assume the last hop of node v is u and $T(s,\ u)$ is not the minimum latency from source s to u so there exists $T'(s,\ u) < T(s,\ u)$. If $B_u[0] + EH[u] * T'(s,\ u) - e_s - e_r \geq 0$ and $B_v[0] + EH[v] * T'(s,\ u) - e_r \geq 0$, $T'(s,\ v) = T'(s,\ u) + 1 < T(s,\ v)$; Otherwise, at least one of them (i.e. u or v) has to sleep for recharging before it has enough residual energy. Since node u and v still need to transmit and receive once respectively, their energy requirements remain unchanged and both of they still need to wait until timeslot $T(s,\ v)$. So $T'(s,\ v) \not> T(s,\ v)$, which is contridicted with assumption that $T(s,\ v)$ is the minimum latency. □

According to Theorem 2, like Bellman-Ford algorithm, we can generate the minimum-latency path from s to all distination nodes $v \in \mathcal{D}$. We call $T(s,\ v)$ the **benchmark delay** of node v, which is **the minimum possible timeslot** for v to receive the multicast packet.

However, if we apply this method directly to the BF-MLMS problem when $|\mathcal{D}| > 1$, energy and time collision is very likely to occur among transmissions on paths to different targets.

B. Generate Route and Schedule Simultaneously

To avoid collision when $|\mathcal{D}| > 1$, we extend the algorithm in the last subsection. First we derive the scheduling priority of each destination node, then the minimum-latency route and collision-free schedule for nodes along the route are generated simultaneously for every node $v \in \mathcal{D}$ in an one-by-one manner according to their priorities.

Before we describe the algorithm, some parameters and their maintenance methods are given below.

(1) $\boldsymbol{La(s,\ u)}$**:** Node u's actual timeslot for receiving packets.
(2) $\boldsymbol{stime(u)}$**:** Node u's timeslot(s) used for sending packets.
(3) $\boldsymbol{F2T(u)}$**:** Timeslots can not be used for transmitting by node u.
(4) $\boldsymbol{F2R(u)}$**:** Node u is forbidden to receive in these timeslots. Specially, each $t \in F2R(u)$ might be assigned a **unique parent** p, which means u can hear packets from p at timeslot t.

The *stime*, *F2T* and *F2R* vector are used for avoiding collision when calculating routes and corresponding schedule and they are updated in a **backtrace** process start from target node v after the route to v is generated. When backtracing, for each relay node u on the path from s to v, we have:

(1) The new sending timeslot of u is added into $stime(u)$.
(2) Nodes in $NB(u)$ add $La(s,\ u)$ into their *F2T* vector.
(3) For each node in $NB(p(u))$, if $Sch(u)[t]$ already exists in its *F2R* vector, just set its unique parent to *null*. Otherwise, $(Sch(u)[t],\ p(u))$ is added into the *F2R* vector.

Now we explain the algorithm in detail.

First of all, we discuss how to determine the scheduling priority for each node $v \in \mathcal{D}$. As $T(s,\ v)$ is derived with the assumption that v is the only target node, when scheduling in multi-target scenarios, **extra delay** not included in v's benchmark delay would be introduced for some timeslots can not be used to compatible with the completed schedules. If v is scheduled after many other distination nodes, transmissions on its optimal path have to avoid lots of other already-determined schedules, which may cause extremely high extra delay. Therefore, routes to target nodes with higher $T(s, v)$ are calculated first.

Then we uses a modified Dijkstra algorithm to search for the path to target node v that does not conflict with existing schedules with the minimum latency. Like typical shortest path algorithm, for edge $(u,v) \in E$, **relaxation** checks whether switching $p(v)$ to u can reduce the current minimum latency of node v (i.e. $La(s,v)$). To avoid collision, we propose the collision-aware relaxation which takes both energy-collision and time-collision into consideration.

First we consider time collision. Given a candidate schedule $Sch(v) = (u, v, t)$, we check whether it has time collision with other existing schedules. Obviously, if $t \notin F2T(u)$ and $F2R(v)$, this transmission is time-collision-free. Otherwise, on

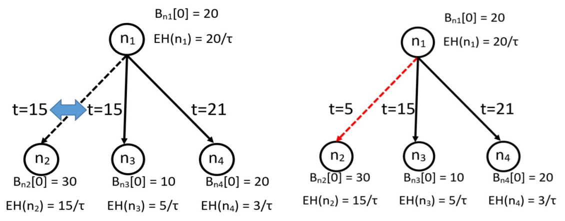


Fig. 2. An example of relaxation on edge (n_1, n_2)

.

condition that $t \in F2R(v)$, we must have $F2R(v)[p] = u$.

Now we discuss how to avoid energy collision by considering relaxation on an arbitrary edge $(u, v) \in E$. Apparently, there is no need to do relaxation about this edge if $v \in V_t$ so we only focus on the circumstance when $v \notin V_t$. If we have $stime(u) = \emptyset$, energy collision with completed schedules will not occur and we can simply apply $L(u, v, La(s,u))$ as the one-hop latency, then choose the minimum timeslot without time collision bigger than the smallest energy-collision-free timeslot. On the contrary, if $|stime(u)| > 0$, transmitting from u to v is likely to have send-after-send energy collision. In this case, we have the following two strategies:

(1) Exploiting the broadcast nature of wireless channel. Node v can pick the minimum timeslot from $stime(u)$ that is free of time collision and we denote this slot b_{can}, which means "broadcast candidate slot". Therefore, u do not need to spare energy for extra sending and no energy collision would occur.
(2) If v's scheduled brother nodes have too large latency (i.e. La value), assigning new timeslots for u's sending may be beneficial and free of energy-collision when u has high recharge rate or initial energy. As the example in Fig.2, edge (n_1, n_2) is under relaxation with transmissions from n_1 to n_3 and n_4 completed ($La(s, n_3) = 15$ and $La(s, n_4) = 20$). Assume n_1 is the source node and we can easily verify the benchmark delay for n_2 is 5 and n_2 can receive the multicast packet from n_1 without any collision. However, if we switch n_2's receiving time to $Sch(n_3)[t]$ or $Sch(n_4)[t]$, the smallest timeslot for n_2's receiving is 15.

For the second strategy above, we have the following theorem to compute v's candidate receiving timeslot s_{can}:

Theorem 3. *Let $stime(u) \neq \emptyset$ and assume v have enough energy to receive at any time and there is no time collision. If $Sch(v)[t] \notin stime(u)$, Let $c = Max\{t | t \in stime(u) \& (t-1) * EH(u) + B_u[0] - e_r - i * e_s < e_s\}$, where $i = |\{t' \mid t' \in stime(u) \;\&\; t' < c\}|$, then the minimum collision-free $Sch(v)[t] = c + ceil((e_s-((c-1)*EH[u]+B_u[0]-e_r-i*e_s))/EH[u]) + 1$.*

Proof. For any $t \in stime(u)$, assume u has sent i-1 packets before timeslot t. If inserting a sending slot in time [1, t-1] causes $B_u[t] < e_s$, we can only select new timeslots bigger than t. Since v is energy-abundant and is ready to receive packets at any slots and there is no time collision, we only need to wait for node u getting recharged. It's not hard to see, the minimum timeslot for $B_u > e_s$ after t-th timeslot is $t + ceil((e_s - (t * EH[u] + B_u[0] - e_r - i * e_s)) / EH[u]) + 1$. Let c denote the largest existing sending slot satisfying the above condition. Finally, at (c+1)-*th* timeslot, u may need extra $(ceil((e_s-((c-1)*EH[u]+B_u[0]-e_r-i*e_s))/EH[u]))$ timeslot to get recharged. This ends the proof. □

s_{can} is the smallest timeslot larger than $Sch(v)[t]$ derived by Theorem 3, which satisfies $B_v[s_{can}] > e_r$ and free of time collision.

Based on above methods, when v chooses u as its parent, its minimum collision-free latency becomes $Min\{b_{can}, s_{can}\}$. Then v determines whether to change its parent to u. The detailed relaxation algorithm is shown in Algorithm 1. Let δ denotes the maximum degree of nodes in G and its expected time complexity is $\theta(\delta)$, which can be seen as a constant.

In short, the centralized algorithm computes collision-free routes and corresponding schedules for each destination node through relaxations and its details are shown in Algorithm 2. For each target node there is at most $O(|E| + |V|^2)$ relaxations for edges in E and determining scheduling priority costs $O(|V||E|)$-time. Hence, the overall time complexity is $O(|V||E| + |\mathcal{D}|(|E| + |V|^2))$.

V. Edge-Based Distributed Algorithm

In this part we introduce the distributed scheduling algorithm. Unlike broadcasting, multicasting targets at only a portion of all nodes. Therefore, we construct a tree spanning all distination nodes firstly, then all edges belong to the multicast tree are scheduled in a top-bottom manner.

A. Construct the Multicast Tree

First we discuss how to construct the multicast tree, in which minimum-latency paths from the sink towards different

Algorithm 1: Collision-aware edge relaxation

```
Input: $G$ = ($V$,$E$); $V_t$ and an edge ($u$,$v$)$\in E$ where $v \notin V_t$;
if $u \notin V_t$ or $u$ is leaf node then
    $mintime \leftarrow$
      $Min\{t \mid timecollision(u,v,t) = false$ &
            $t \geq La(s,u) + L(u,v,La(s,u))\}$;
else
    $b_{can} \leftarrow$
      $Min\{t \mid t \in stime(u)$ & $timecollision(u,v,t) =$
            $false$ & $B_v[0] + t * EH(v) \geq e_r\}$;
    $d \leftarrow$
      $Max\{ceil((e_r - B_v[0])/EH(V)) + 1, La(s,u)\}$;
    $c \leftarrow$
      $Max\{t \mid t \in stime(u)$ & $(t-1) * EH(u) + B_u[0] -$
            $e_r - i * e_s < e_s$ & $t > d,\ i = |\{t' \mid t' \in$
            $stime(u)$ & $t' \leq t\}|\}$;
    $s_{can} \leftarrow$
      $Min\{t \mid t \geq c + ceil((e_s - ((c-1) * EH(u) +$
            $B_u[0] - e_r - i * e_s))/EH(u)) + 1$ &
            $timecollision(u,v,t) = false$ & $t \notin stime(u)\}$;
    $mintime \leftarrow Min\{b_{can}, s_{can}\}$;
if $mintime < La(s,v)$ then
    $La(s,v) \leftarrow mintime$;
    $p(v) \leftarrow u$;
else if $mintime = La(s,v)$ then
    if $b_{can} < s_{can}$ then
        //Some scheduled nodes whose parent are $u$
          receive packet at $mintime$;
        $p(v) \leftarrow u$;
    else if $EH(u)/(ch(u)+1) > EH(p(v))/ch(p(v))$
     then
        $p(v) \leftarrow u$;
```

distinations are generated without considering collision. We use a classic distributed relaxation method where the control message containing a node's shortest-path information is first transmitted by the source node. When a node receives a control message, it checks whether switching parent to a new node can reduce its benchmark delay (i.e. $T(s,v)$). Nodes will broadcast its control message while it has its own information updated. To reduce the number of message transmitted, we apply a technique: a node would wait for a period of time that is proportional to its updated $T(s,v)$ before sending its control message. In this way, messages from nodes with large latency is tend to be omitted and broadcast storm will not occur. Further, we notice an interesting fact that there is usually multiple minimum-latency paths for a distination in BF-WSNs, in which a node can select its last hop arbitrarily without increasing its benchmark delay. On this condition, neighbors with abundant energy and fewer child nodes is preferred by a node.

We note that, the control message can be piggybacked in ordinary data packets so this process can be integrated into ordinary data transmission easily.

Algorithm 2: Centralized algorithm

```
Input: $G$=($V$, $E$); source node $s$; target node set $\mathcal{D}$;
        $EH(v)$ and $B_v[0]$ for all nodes v $\in V \setminus \{s\}$.
Output: Multicast tree node set $V_t$ and all $Sch(v)$ for
        each node in $V_t$.
$Compute\ T(s,v)\ for\ each\ v \in \mathcal{D}$;
$Sort\ \mathcal{D}\ by\ T(s,v)\ in\ descending\ order$;
for $node\ v \in \mathcal{D}$ & $v \notin V_t$ do
    $P \leftarrow V_t$;
    for $node\ v' \notin V_t$ do
        $La(s,v') \leftarrow \infty, p(v') \leftarrow null$;
    for $edge\ (u',v') \in E, u' \in P \& v' \in V \setminus P$ do
        $Relaxation(u',v')$;
    $m \leftarrow \arg\min_{v'} \{La(s,v') | v' \in V \setminus P\}$;
    $P \leftarrow P \cup \{m\}$;
    while $V \setminus P \neq \emptyset$ do
        for $node\ v' \in V \setminus P$ do
            $Relaxation(m,v')$;
        $m \leftarrow \arg\min_{v'} \{La(s,v') | v' \in V \setminus P\}$;
        $P \leftarrow P \cup \{m\}$;
    $Backtrace\ from\ v\ to\ s\ to\ update\ stime,\ F2R,\ F2T$
    $for\ node\ along\ the\ path$;
```

B. Edge-Based Scheduling

By now, the multicast tree T = (V_t,E_t) spanning all distination nodes has been determined. In this part we compute the actual receiving time $La(s,v)$ for each node $v \in V_t \setminus \{s\}$ by putting off the receiving time on the basis of the benchmark latency in a top-to-bottom manner. Each node $v \in V_t \setminus \{s\}$ receives multicast packet once, which corresponds to edge (u,v) $\in E_t$ so we schedule **edges** in our algorithm. Node u and v are called **sending node** and **receiving node** of edge (u,v), respectively.

First, we discuss how to determine the scheduling priority of an edge. The **scheduling section** of an edge targetting at an distination node as below:

Definition 3. ***(Scheduling Section)*** *For any relay node r on the path from source s to v* $\in \mathcal{D}$ *(r*$\neq$*s), interval [T(s, r), T(s, v)-h(r, v)] is called Scheduling Section of edge (p(r),r) targeting at v, where h(r, v) is the hop count from r to v in the multicast tree.*

We have following theorem for an edge's Scheduling Section:

Theorem 4. *Considering relay node r* $\in V_t$ *and v is a leaf node in the subtree rooted at r. If La(s,r) has been determined, the possible minimum latency for v is* $Max\{T(s,v), T(s,v) + La(s,r) - T(s,v) - h(r,v)\}$.

Proof. First we have the following sub-theorem. When $La(s,r)$ $\in$ [$T(s,r)$, $T(s,v)$-$h(r,v)$]. The possible minimum value for $La(s,v)$ is $T(s,v)$. We prove this by induction.

When $h(r,v) = 1$, r is the parent node of v. We have the following two cases:

(a) If $B_r[0]+EH(r)*T(s,r)-e_r-e_s \geq 0$ and $B_v[0]+EH(v)*T(s,r)-e_r \geq 0$, we have $T(s,v) = T(s,r)+1$ so that $La(s,r) = T(s,r)$. Obviously the minimum possible value for $La(s,v)$ is $T(s,v)$.

(b) Otherwise, without loss of generality, we assume $L(r,v,t) = ceil((e_s-(B_r[0]+EH(r)*T(s,r)-e_r))/EH(r))+1$. Let $La(s,r) = T(s,r)+x$ and the minimum timeslot for v to receive packet is $T'(s,v)$. We have the following two cases:

(1) $e_s+e_r-B_r[0]-La(s,r)*EH(r) \leq 0$, apparently we also have $e_r-B_v[0]-La(s,r)*EH(v) \leq 0$, thus, $T'(s,v) = T(s,r)+x+1$ and $x \geq (e_s+e_r-B_r[0]-T(s,r)*EH(r))/EH(r)$. As $La(s,r) < T(s,v)-h(r,v)$ also holds, we have $x = ceil((e_s+e_r-B_r[0]-T(s,r)*EH(r))/EH(r))$. Thus, $T'(s,v)=T(s,v)$.

(2) Otherwise, we have $T'(s,v)=La(s,r)+ceil((e_s+e_r-B_r[0]-La(s,r)*EH(r))/EH(r))+1 = x+T(s,r)+ceil((e_s+e_r-B_r[0]-T(s,r)*EH(r))/EH(r))-x+1=T(s,v)$.

Then we assume when $h(r,v) = k>1$, the sub-theorem also holds. If $h(r,v) = k+1$, let u denote the child node of r on the path from s to v. Then if $La(s,r) \in [T(s,r), T(s,u)-1]$, the minimum possible value for $La(s,u)$ is $T(s,u)$. Otherwise, if $La(s,r) \in [T(s,u), T(s,v)-k-1]$, $L(r,u,La(s,r))=1$. Therefore, $La(s,u)$'s minimum value is $La(s,r)+1$, which belongs to $[T(s,u), T(s,v)-k]$. As u is k hops from v, the minimum possible value for $La(s,v)$ is still $T(s,v)$. This proves the sub-theorem.

Next we have another sub-theorem: When $La(s,r)>T(s,v)-h(r,v)$, the minimum possible value of $La(s,v)$ is $T(s, v)+La(s, r)-T(s, v)-h(r, v)$. It can be verified easily similar to the first sub-theorem. Based on the above-mentioned two sub-theorems, Theorem 4 is proved. □

According to the analysis on two sub-theorems related to Theorem 4, we can schedule an edge $(p(r),r)$ without increasing the minimum delay of its successive distination nodes (i.e. leaf nodes of the subtree rooted at r) if r is scheduled to receive in its scheduling section. Moreover, once $La(s,r)$ is beyond its scheduling section, this schedule must cause extra latency on its successive distinations. Obviously, scheduling section can be exploited to determine the scheduling priority. As the number of scheduling section of an edge $(p(r),r)$ is equal to the amount of leaf nodes in the subtree rooted at r and all the section has the same left end, only the average length of all the edge $(p(r),r)$'s scheduling sections is kept by node $p(r)$, which is denoted by $avgl(r)$. $avgl(r)$ is computed as follows:

(1) Each leaf node $v \in \mathcal{D}$ transmits a control packet denoted by ($nodenum$, $avgl(v)$, $T(s,v)$, $leafnum(v)$) = (v, 0, $T(s,v)$, 0) to its parent node, in which $leafnum(v)$ means the number of leaf nodes in the subtree rooted at v.

(2) Once a relay node r has received control packets from all its child nodes, it will records $T(s,v')$ and $avgl(v')$ for each $v' \in ch(r)$, then we have $avgl(r) = \sum_{v' \in ch(r)} leafnum(v') * (T(s,v') - T(s,r) + avgl(v') - 1) \Big/ \sum_{v' \in ch(r)} leafnum(v')$. A packet including the same information as in (1) as also transmitted to $p(r)$.

As our algorithm runs in top-to-bottom manner, the scheduling priority is determined by following four factors, which is listed in descending order of their importance: (a) Edge's level in the multicast tree (i.e. $h(s,v)$); (b) $avgl(v)$; (c) $EH(p(v))/|ch(p(v))|$; (d) The node number of v. The smaller they are, the higher priority an edge will have.

To avoid collision while scheduling, edge $(p(v),v)$'s collision set [32] containing other edges in multicast tree conflicting with it is defined as follows:

Definition 4. ***(Collision Set)*** *$(p(v),v)$'s collision set $C(p(v),v) = \{(p(u),u) \mid u \in V_t \;\&\; h(s,u) \leq h(s,v) \;\&\; (p(u) \in NB(v)$ or $u \in NB(p(v)))\}$*

For each edge $(p(v),v)$, $C(p(v),v)$ is maintained at node $p(v)$. For any edge $(p(u),u) \in C(p(v),v)$, its information (i.e. $h(s,u)$, $avgl(u)$, $La(s,u)$) is maintained as a triple (h, $avgl$, La). Since scheduling starts from s and follows top-to-bottom manner, there is no send-after-receive collision. Thus, only send-after-send collision and time collision is taken into account when computing the collision set.

Now we discuss how to compute collision set. It's not hard to see, messages from both $p(v)$ and v can be heard by either sending or receiving nodes of any edge in $C(p(v),v)$. Further, to reduce message transmitted we also record the set of all edges which may cause negative collision [32] at node v, which is denoted by $Own\_c(v)$. Therefore, $C(p(v),v)$ and $Own\_c(v)$ can be calculated by following steps:

(1) $C(p(v),v) = C(p(v),v) \cup \{(p(v),u) \rightarrow (h(s,u),\ avgl(u),\ 0) \mid u \neq v\}$;

(2) Both $p(v)$ and v broadcast a packet containing $p(v)$, v, $T(s,v)$, $avgl(v)$ and $h(s,v)$.

(3) When v receives a packet with information of edge ($p(u)$, u), if $p(u) \in NB(v)$, $h(s,u) \leq h(s,v)$ and edge ($p(u)$, u) is not included in $Own\_c(v)$, v forwards this packet to $p(v)$ and $Own\_c(v) = Own\_c(v) \cup \{(p(u),u) \rightarrow (h(s,u),\ 0,\ 0)\}$;

(4) When $p(v)$ receives a packet from v containing information of edge ($p(u)$, u), $p(v)$ just adds ($p(u)$, u) and its information into $C(p(v),v)$. Otherwise, if $u \in NB(p(v))$ and $h(s,u) \leq h(s,v)$, we have $C(p(v),v) = C(p(v),v) \cup \{(p(u),u) \rightarrow (h(s,u),\ 0,\ 0)\}$.

Next we describe how to schedule edges by exploiting collision set. We take an edge $(p(v),v)$ for example. In $C(p(v),v)$, each edge has a scheduling priority and we can calculate $La(s,v)$ only when all other edges in $C(p(v),v)$ with a higher priority than $(p(v),v)$ is scheduled. Besides, a scheduling state is assigned for each edge $(p(v),v)$, which is maintained at both $p(v)$ and v and denoted by $state(v)$. There are three states for each edge, including *NOT_READY*, *READY* and *FINISHED*. As any two edges are different in priority and they are scheduled at distinct time with the lower-priorty edge avoiding conflict with existing schedule of higher-priorty edge, the overall scheduling result for all the edges in the multicast tree is collision-free.

The scheduling algorithm is described below. We consider the scenario when a node v receives control packet *con-*

trol($p(u)$, u)=($p(u)$,u,$EH(p(u))$,$avgl(u)$,$La(s,u)$) and there are two cases.

In the first case, we have $avgl(u) \neq 0$, $La(s,u) = 0$ and the packet indicates edge ($p(u)$, u) is ready to be scheduled.

(1) If $u = v$ and $state(v)$ = *NOT_READY*, v changes $state(v)$ to *READY* and rebroadcast this packet;

(2) If $u \neq v$ and $Own\_c(v)[p(u), u].avgl = 0$, $Own\_c(v)[p(u), u].avgl$ is updated by $avgl(u)$ and v forwards this packet to $p(v)$. Then, if there exists some nodes $n \in ch(v)$ that satisfies $C(v, n)[p(u), u].avgl = 0$ and $p(u) \neq v$, $C(v, n)[p(u), u].avgl$ is updated by $avgl(u)$ and v will try to schedule its sending edges.

In the second case, $avgl(u) = 0$ and $La(s,u) \neq 0$. This means edge ($p(u)$, u) has been scheduled.

(1) If $u = v$ and $state(v) \neq$ *FINISHED*, $state(v)$ is updated to *FINISHED* and this packet is rebroadcasted by v. Then for each node $n \in ch(v)$, set $state(n)$ to *READY*. If $La(s, u) + 1 > T(s, n)$, $avgl(n)$ is updated by $avgl(n)+T(s, n)$-$La(s, u)$-1. Then, the control packet (v, n, $EH(v)$, $avgl(n)$, 0) is broadcasted. After processing all child nodes, v will try to schedule its sending edges.

(2) If $u \neq v$ and $Own\_c(v)[p(u), u].La = 0$, $Own\_c(v)[p(u), u].La$ is set to $La(s,u)$ and this packet is forwarded to v's parent. Then, if there exists v's child node n satisfying $C(v, n)[p(u), u].La = 0$ and $p(u) \neq v$, v will update $C(v, n)[p(u), u].La$ and try to schedule its sending edges.

while node v try to schedule its sending edges, it checks whether there exists some node n whose state is *READY* and has the highest priority compared with all the edges (p,q) that $C(v, n)[p, q].La = 0$. If this condition satisfies, $La(s,n)$ is updated by the *mintime* derived by the first half of Algorithm 1, then v broadcast the control packet (v, n, $EH(v)$, 0, $La(s,n)$) to inform other nodes that edge (v, n) has been scheduled.

The theorem below verifies the correctness of above two algorithms for multicast tree constructing and scheduling.

Theorem 5. *Both centralized and distributed algorithms can stop within limited time and generate a multicast tree and the corresponding schedule without time-collision and energy-collision.*

VI. Simulation Results

In this section we evaluate our algorithms by simulations. First, we study the performance of multicast latency for both algorithms under different energy conditions and application scenarios. Since there is not a state-of-art method for solving the problem of BF-MLMS, we regard $Max\{T(s,v) \mid v \in \mathcal{D}\}$ as the benchmark latency, which is one **lower bound** of multicast delay. Next, we focus on communication overheads for different methods.

A. Simulation Setups

The simulation parameters are set as follows. Firstly, we deploy 200 battery-free nodes in a 200m $\times$ 200m field randomly and the average degree of nodes is 10. Secondly, we set e_s = 100 and e_r = 80, with a node 's energy harvesting rate varies from 1 to 70. When there is adequate ambient energy, nodes can be always-awake and work continuously, which is usually not true for BF-WSNs. Therefore, to simulate harsh energy conditions, the average recharge rate and the initial energy is set to 10 and 0 in general cases, respectively. Thirdly, 5%˜50% of all nodes are selected as distination nodes randomly and we pick 20% under general circumstances. In all simulations, each plotted point is the average of 100 executions.

B. Multicast Latency

Fig.3 shows the multicast latency as a function of average initial energy and recharge rate. Obviously, the latency decreases as initial energy and energy harvesting rate increase. Fig.3(a) shows the centralized algorithm only increases the benchmark delay by about 2%. Distributed method has the same performance as the centralized when there is few initial energy, which is also true when the initial energy is enough, with the latency only 12% bigger than the lower bound. As in Fig.3(b), both algorithms can derive the schedule with multicast latency very close to the benchmark, which bring 1% and 4% of extra latency compared with the lower bound for centralized and distributed methods, respectively. Thus, the schedules generated by both algorithms are time-effective in different ambient energy conditions, especially when there is not abundant energy. This is because both algorithms use local optimal strategies. Centralized method jointly considers route calculation and collision avoidance, generates the minimum-latency path and receiving schedule at the same time at each iteration with the route towards destination nodes with higher benchmark latency (i.e. $T(s,v)$) derived firstly. Distributed algorithm uses greedy method that assigning an edge with the minimum collision-free timeslot. Besides, the edge with short scheduling section is considered with higher priority. These techniques greatly reduce the possibility of introducing extra delay for each target node. However, compared with the centralized method considering all nodes in the network when computing routes, the distributed can only schedule on the pre-determined multicast tree, which brings more locality, making the derived multicast latency higher.

Fig.4 studies the impact of application scenarios with network size and coverage ratio (i.e. the percentage of distination node) considered. Both algorithms achieve a very low multicast latency, with only 1% more than the minimum possible delay approximately. Yet the multicast latency sees little change with the number of nodes increases. Similarly, although there is a rapid growth as coverage ratio increases at the beginning, the latency remains stable after the coverage ratio reaches about 20%. The cause for this issue is that, the time consumed for energy replenishment is the main cause for transmission latency in BF-WSNs. The number of nodes to be scheduled increases with both coverage quality and network size. as it reaches some certain quantities, nodes with the minimum energy harvesting rate must be included in the multicast tree, which act as **bottlenecks** dominating the overall latency.

To investigate how energy bottleneck nodes influence scheduling, the same simulations as Fig.3,4 are carried out without bottleneck, where all nodes share the same recharge rate. As in Fig.5, the latency also shows a decreasing trend as in Fig.3, yet the latency by both algorithms changes dramatically, with 5% and 15% to 40% higher than benchmark latency for the centralized and distributed, respectively. The reasons for this are as follows.

(1) Compared with results in Fig.3, the extra delay introduced remains unchanged for it has little relationship with bottleneck nodes. Meanwhile, the benchmark latency decreases by 50%.
(2) The scheduling priority of nodes (edges) for both algorithms has strong connection with the benchmark delay but it shows no significant difference among different nodes in energy-bottleneck-free BF-WSNs.

Similarly, as in Fig.6, the lower bound is only 40% of that in general networks and the proportion of extra delay is 3% for the centralized and 20% for the distributed. In Fig.6(a), with the scale of networks, the contribution of hop count to the multicast latency is more obvious than that in Fig.4(a). In terms of coverage ratio in In Fig.6(b), as each node has the same energy harvesting rate, the benchmark latency sees almost no change as the number of target nodes increases. Nevertheless, the extra latency increases as there are more nodes that needs scheduling. It is worth mentioning that, the advantage for centralized method over the distributed is especially significant in the existence of bottleneck nodes.

Generally speaking, in spite of the performance degradation when the network is free of bottlenecks, both algorithms still have high effeciency in terms of multicast latency since the ambient energy is usually distributed inhomogeneously among nodes.

C. Communication Overhead

In this part we compare the communication overhead (i.e. the number of control message transmitted) of both algorithms under different network topologies. We assume the centralized algorithm operates as follows: First, each node in the network transmits a packet containing its information. After gathering packets from all the nodes, the sink run the algorithm and send the schedule message for each target node through the corresponding minimum-hop-count route respectively.

In Fig.7(a), the communication overhead is a function of network scale. As the number of nodes increases, both algorithms have more message transmitted, with the centralized increases more faster than the distributed. We notice that the average hop count to sink is directly proportional to the number of nodes in the network as long as the average degree remains unchanged. For centralized method, both average hop count and nodes to transmit packets increase by the same factor as network scale expands. Thus, if we enlarges the network n times, the message overhead will increase by $O(n^2)$ times. Nevertheless, a node in distributed algorithm only gathers one or two-hop information and the amount of transmitted message is independent of its hop to the sink. Hence, the message overhead will only increases by $O(n)$ times in the same scenario. Actually, in large-scale networks, sink may need to add path information to the packet to transmit it directly to the corresponding target, which makes the size of control packet for centralized algorithm much larger than that in the distributed.

Fig.7(b) discusses relation between node's degree and communication overhead. As the deployment of nodes becomes denser, the overhead by centralized method decreases while the distributed increases. The reason why they show distinct trends is as follows. When the network scale is fixed, for one thing, the minimum hop for a node to the sink decreases as it has more neighbors. For another, in distributed algorithm, the size of collision set increases with the number of neighboring node, resulting in more information to be exchanged. Particularly, the message overhead of distributed algorithm is even higher when the average degree reaches a certain quality.

In short, the message volume for scheduling has great relationship with network topology. The distributed algorithm can reduce the communication overhead when the network is sparse or in large scale, yet centralized algorithm has lower message complexity if nodes are deployed in a very dense manner.

VII. Conclusion

In this paper we focus on the problem of BF-MLMS. Based on the calculation of end-to-end delay, a centralized dynamic-programming algorithm is first proposed to generate transmission path and schedule simultaneously. Then by postpone receiving time appropriately, an edge-based distributed algorithm is introduced to reduce multicast delay. Simulation results show that both our algorithm have high performance in terms of multicast latency and their suitable scenarios concerning communication overhead is analysed.

References

[1] S. Ulukus et al., "Energy Harvesting Wireless Communications: A Review of Recent Advances," in *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 3, pp. 360-381, March 2015.
[2] Y. Yang, L. Su, Y. Gao and T. F. Abdelzaher, "SolarCode: Utilizing Erasure Codes for Reliable Data Delivery in Solar-powered Wireless Sensor Networks," in *INFOCOM* 2010, pp. 1-5.
[3] X. Lu, P. Wang, D. Niyato, D. I. Kim and Z. Han, "Wireless Networks With RF Energy Harvesting: A Contemporary Survey," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 757-789, 2015.
[4] Z. Yang, H. Gao, S. Cheng, Z. Cai and J. Li, "IEA: An Intermittent Energy Aware Platform for Ultra-Low Powered Energy Harvesting WSN," in *WASA* 2017, vol 10251.
[5] T. Shi, J. Li, H. Gao and Z. Cai, "Coverage in Battery-Free Wireless Sensor Networks," in *INFOCOM* 2018, pp. 108-116.
[6] R. Gandhi, A. Mishra and S. Parthasarathy, "Minimizing broadcast latency and redundancy in ad hoc networks," In *MobiHoc* 2003, pp. 222-232.
[7] R. Mahjourian, F. Chen, R. Tiwari, M. Thai, H. Zhai, and Y. Fang, "An approximation algorithm for conflict-aware broadcast scheduling in wireless ad hoc networks," In *MobiHoc* 2008, pp, 331-340.
[8] R. Gandhi, Y. Kim, S. Lee, J. Ryu and P. Wan, "Approximation Algorithms for Data Broadcast in Wireless Networks," in *IEEE Transactions on Mobile Computing*, vol. 11, no. 7, pp. 1237-1248, July 2012.
[9] Z. Chen, C. Qiao, J. Xu and T. Lee, "A Constant Approximation Algorithm for Interference Aware Broadcast in Wireless Networks," *INFOCOM* 2007, pp. 740-748.

[10] R. Tiwari, T. N. Dinh and M. T. Thai, “On Centralized and Localized Approximation Algorithms for Interference-Aware Broadcast Scheduling,” in *IEEE Transactions on Mobile Computing*, vol. 12, no. 2, pp. 233-247, Feb. 2013.

[11] H. Gong, L. Zhao, K. Wang, W. Wu, and X. Wang. “A Distributed Algorithm to Construct Multicast Trees in WSNs: An Approximate Steiner Tree Approach,” In *MobiHoc* 2015. pp. 347-356.

[12] J. Hong, J. Cao, W. Li, S. Lu and D. Chen, “Sleeping Schedule-Aware Minimum Latency Broadcast in Wireless Ad Hoc Networks,” in *ICC* 2009, pp. 1-5.

[13] F. Wang and J. Liu, “Duty-Cycle-Aware Broadcast in Wireless Sensor Networks,” in *INFOCOM* 2009, pp. 468-476.

[14] X. Jiao, W. Lou, J. Ma, J. Cao, X. Wang and X. Zhou, “Duty-Cycle-Aware Minimum Latency Broadcast Scheduling in Multi-hop Wireless Networks,” in *ICDCS* 2010, pp. 754-763.

[15] Wang. X, Zhou. X, Cao. J, et al. “Minimum Latency Broadcast Scheduling in Duty-Cycled Multihop Wireless Networks,” *IEEE Transactions on Parallel & Distributed Systems*, 2012, 23(1):110-117.

[16] L. Su, B. Ding, Y. Yang, T. F. Abdelzaher, G. Cao and J. C. Hou, “oCast: Optimal multicast routing protocol for wireless sensor networks,” in *ICNP* 2009, pp. 151-160.

[17] D. T. Le, T. L. Duc, V. V. Zalyubovskiy, D. S. Kim, H. Choo,“ LABS: Latency aware broadcast scheduling in uncoordinated Duty-Cycled Wireless Sensor Networks”, *Journal of Parallel and Distributed Computing*,Volume 74, Issue 11,2014, 3141-3152.

[18] L. Xu, G. Chen, J. Cao, S. Lin, H. Dai, X. Wu, and F. Wu, “Optimizing Energy Efficiency for Minimum Latency Broadcast in Low-Duty-Cycle Sensor Networks,” in *ACM Trans. Sen. Netw.* 11, 4, Article 57 (July 2015), 31 pages.

[19] Q. Chen, H. Gao, S. Cheng, X. Fang, Z. Cai and J. Li, “Centralized and Distributed Delay-Bounded Scheduling Algorithms for Multicast in Duty-Cycled Wireless Sensor Networks,” in *IEEE/ACM Transactions on Networking*, vol. 25, no. 6, pp. 3573-3586, Dec. 2017.

[20] L. Cheng, J. Niu, C. Luo, L. Shu, L. Kong, Z. Zhao, Y. Gu, “Towards minimum-delay and energy-efficient flooding in low-duty-cycle wireless sensor networks,” in *Computer Networks*, Volume 134, 2018, Pages 66-77.

[21] T. Zhu, Z. Zhong, T. He and Z. Zhang, “Achieving Efficient Flooding by Utilizing Link Correlation in Wireless Sensor Networks,” in *IEEE/ACM Transactions on Networking*, vol. 21, no. 1, pp. 121-134, Feb. 2013.

[22] L. Cheng, Y. Gu, T. He and J. Niu, “Dynamic switching-based reliable flooding in low-duty-cycle wireless sensor networks,” in *INFOCOM* 2013 pp. 1393-1401.

[23] Q. Chen, T. Wang, L. Cheng, Y. Tao and H. Gao, “Energy-Efficient Broadcast Scheduling Algorithm in Duty-Cycled Multihop Wireless Networks,” *Wireless Communications and Mobile Computing*. 2019. 1-14.

[24] E. Z. Ang , H. P. Tan , and W. K. G. Seah. “Opportunistic routing in wireless sensor networks powered by ambient energy harvesting,” in *Computer Networks* 54.17(2010):2943-2966.

[25] S. Yang, Y. Tahir, P. Chen, A. Marshall and J. McCann, “Distributed optimization in energy harvesting sensor networks with dynamic in-network data processing,” in *INFOCOM* 2016, pp. 1-9.

[26] Gu Y , He L , Zhu T , et al. “Achieving energy-synchronized communication in energy-harvesting wireless sensor networks”. in *ACM Transactions on Embedded Computing Systems*, 2014, 13(2s):1-26.

[27] S. Guo, C. Wang and Y. Yang, “Mobile data gathering with Wireless Energy Replenishment in rechargeable sensor networks,” in *INFOCOM* 2013, pp. 1932-1940.

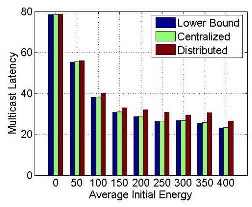
[28] C. Wang, J. Li, Y. Yang and F. Ye, “A hybrid framework combining solar energy harvesting and wireless charging for wireless sensor networks,” in *INFOCOM* 2016, pp. 1-9.

[29] A. Baknina and S. Ulukus, “Optimal and Near-Optimal Online Strategies for Energy Harvesting Broadcast Channels,” in *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3696-3708, Dec. 2016.

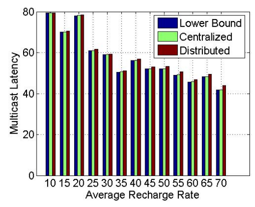
[30] M. Y. Naderi, K. R. Chowdhury, S. Basagni, W. Heinzelman, S. De and S. Jana, “Experimental study of concurrent data and wireless energy transfer for sensor networks,” in *GLOBECOM* 2014, pp. 2543-2549.

[31] Q. Chen, H. Gao, Z. Cai, L. Cheng and J. Li, “Energy-Collision Aware Data Aggregation Scheduling for Energy Harvesting Sensor Networks,” in *INFOCOM* 2018, pp. 117-125.

[32] B. Yu, J. Li and Y. Li, “Distributed Data Aggregation Scheduling in Wireless Sensor Networks,” in *INFOCOM* 2009, pp. 2159-2167.

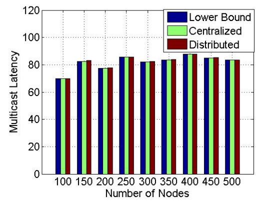


(a) Initial energy.

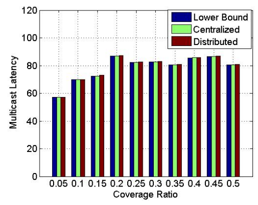


(b) Recharge rate.

Fig. 3. Latency for different energy conditions.

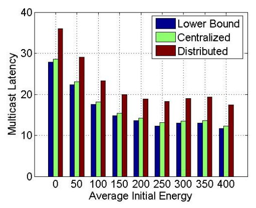


(a) Network scale.

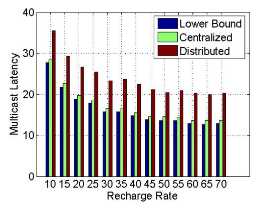


(b) Coverage ratio.

Fig. 4. Latency for different application scenarios.

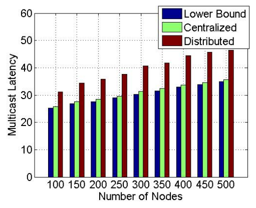


(a) Initial energy.

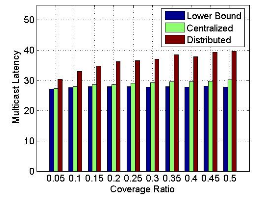


(b) Recharge rate.

Fig. 5. Latency for different energy conditions without bottleneck nodes.

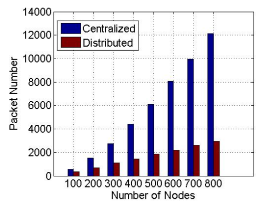


(a) Network scale.

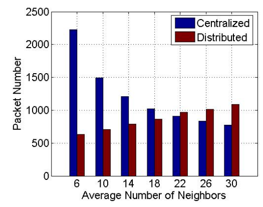


(b) Coverage ratio.

Fig. 6. Latency for different application scenarios without bottleneck nodes.



(a) Network scale.



(b) Average degree.

Fig. 7. Communication overhead for both algorithms.