oCast: Optimal Multicast Routing Protocol for Wireless Sensor Networks

Lu Su*, Bolin Ding*, Yong Yang*, Tarek F. Abdelzaher*, Guohong Cao† and Jennifer C. Hou*
*Department of Computer Science
University of Illinois at Urbana-Champaign
{lusu2, bding3, yang25, zaher, jhou}@illinois.edu
†Department of Computer Science & Engineering
The Pennsylvania State University
gcao@cse.psu.edu

***Abstract*—In this paper, we describe oCast, an energy-optimal multicast routing protocol for wireless sensor networks. The general minimum-energy multicast problem is NP-hard. Intermittent connectivity that results from duty-cycling further complicates the problem. Nevertheless, we present both a centralized and distributed algorithm that are provably optimal when the number of destinations is small. This model is motivated by scenarios where sensors report to a small number of base stations or where data needs to be replicated on a small number of other nodes. We further propose an extended version of oCast, called Delay Bounded oCast (DB-oCast), which can discover optimal multicast trees under a predefined delay bound. Finally, we demonstrate the advantages of our schemes through both theoretical analysis and simulations.**

I. Introduction

Multicast is an essential operation in wireless sensor networks, as it can provide an efficient way for group communication, which is widely used in sensor networks. Generally, a sensor network is composed of a large number of sensing nodes, whose task is to sense physical surroundings, and a small number of base stations (or sink nodes), whose task is to store and process the sensory readings. Based on the communication patterns between these two kinds of nodes, the multicast applications of sensor networks can be divided into two categories, namely, **large group multicast** and **small group multicast**.

Large group multicast normally takes place when a sink node attempts to disseminate a query (or command) to parts or all of the sensor network. A notable feature of this style of multicast is the large size of the destination set. This scenario is similar to the multicast/broadcast applications in wireless ad hoc networks, which have been widely studied in recent years. Unfortunately, the problem of finding minimum-energy multicast/broadcast trees in ad hoc networks is proven to be NP-hard [1], [2], [3], making it generally impossible to find an optimal solution in polynomial time.

In contrast, the small group multicast is used when a sensor reports its readings to multiple sink nodes. In this case, the number of the destinations is small, which is normally less than 10. The well known Directed Diffusion paper [4] gives a typical description of this scenario. In its description, a sensor node needs to periodically report its sensing data to one or multiple sink nodes that are interested in the data. Naturally, if more than one sinks are interested in the same data or events, a small group multicast tree should be constructed. Another common example of small group multicast involves data-centric storage (DCS) [5], [6], where the sensing data are saved on some nodes inside the sensor network. In data storage protocols, to increase the data reliability and availability, and to balance the load, the data are usually replicated to multiple nodes. This strategy can make good use of small group multicast trees to improve the energy efficiency of data replication.

This work was supported in part by National Science Foundation under grant CNS-0916171.

As depicted in [4], after the multicast tree is constructed, a sensor may be required to report its data in a very frequent pattern (once every 20ms for example). Therefore, great energy savings can be achieved if we can optimize the small group multicast. To our pleasure, the small-destination-number property of the small group multicast provides us a chance to exploit optimal multicast trees.

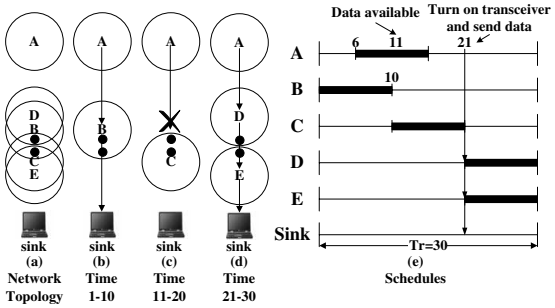


Fig. 1. An example of intermittently connected topology, where nodes B,C,D,E monitor the targets in turn, and A wants to send packets to the sink

Providing multicast services in sensor networks is further complicated by another unique challenge: **intermittently connected topology**, which is recognized in the recent work [7]. It arises because of the energy conservation scheduling scheme [8], [9], [10], [11], [12], [13], [14], a frequently used mechanism in sensor networks.

The basic idea of the scheduling scheme is to schedule the activity of each sensor node such that sensors perform the given mission in turn and at any time, only a small number of sensors are active to meet the coverage requirement of the mission. With this scheme, the number of active nodes is significantly reduced and thus the network lifetime is prolonged. However, some scheduling strategies, such as partial coverage scheduling [10], [11] and target coverage scheduling [12],

[13], may result in very sparse distribution of active nodes. Consequently, the network may not be connected at some instant due to the low node density.

Example 1.1: Consider a target coverage scheduling example shown in Fig. 1(a). The mission of the sensor network is to continuously monitor (cover) two targets (black points) in the deployed region. Initially, the targets are sensed by nodes B, C, D, E (circles denote the sensing range of the nodes). To save power, the scheduling algorithm proposed in [12] is used to only keep part of the nodes active. As illustrated in Fig. 1, node B is active from time 1 to 10 (Fig. 1(b)), node C is active from time 11 to 20 (Fig. 1(c)), and both D and E are active from time 21 to 30 (Fig. 1(d)). Suppose at time 11, node A collects some data and wants to send the data to the sink. However, since only node C is active at this time, and it is outside the communication range of A (Fig. 1(c)), A's data cannot be forwarded to the sink.

For such intermittently connected sensor networks, traditional multicast algorithms designed for wireless ad hoc networks cannot be applied. To address this problem, we propose **oCast**, the **O**ptimal multi**Cast** routing protocol for sensor networks. oCast can effectively identify minimum-energy multicast trees for small group multicast in intermittently connected sensor networks. In case the network is always connected (i.e., no energy saving scheduling), oCast can also construct optimal multicast trees. Furthermore, oCast does not rely on any specific unicast routing protocol, and it can provide the shortest end-to-end path from any sensor node to any sink by itself. In this paper, we describe both centralized and distributed implementations of oCast.

Many applications in sensor networks, such as military surveillance and forest fire alarms, have delay constraints, but the multicast trees identified by oCast may not satisfy such delay requirements. To address this problem, we propose an extended version of oCast called **D**elay **B**ounded **oCast** (**DB-oCast**), which can discover optimal multicast trees under a predefined delay bound.

The rest of the paper is organized as follows. We introduce the system model in Section II and formulate the problem in Section III. Section IV and Section V present our oCast and DB-oCast algorithms respectively. In Section VI, we explain how oCast and DB-oCast are applied in a decentralized manner. Then, we discuss some related issues in Section VII, and evaluate the performance of the proposed schemes in Section VIII. Section IX concludes the paper.

II. System Model

Similar to many application-driven scheduling schemes [8], [9], [10], [11], [12], [13], we assume the lifetime of a sensor network is divided into *rounds* with equal duration (T_r). For each node, a round consists of a *working period* (T_w) and a *sleep period*. Each round is divided into time slots, and a packet is only transmitted at the start of a time slot. Without loss of generality, we assume T_w varies among the nodes. In our discussion, we use $s_i.start$ and $s_i.end$ to denote the start and end times of node s_i's working period ($s_i.end \equiv s_i.start + T_w - 1 \pmod{T_r}$).

To simplify the discussion, in all the examples, the sensor nodes are synchronized. Nevertheless, our algorithms do not pose any restriction on the synchronization between neighboring nodes. Each node only needs to know the relative position of its neighbor's working period (i.e., the offset between the start times of their working period), and this can be easily achieved through the periodic exchange of hello messages.

Under an intermittently connected topology, if a node only delivers packets during the working period, the packets may never be sent out, because its working period may not overlap with that of its neighbors. To address this issue, each node still receives packets within the working period but the sending rule is modified as follows: *a node can wake up its transceiver and send packets at any time, if needed*. For instance, as shown in Fig. 1(e), node A can buffer the data until D wakes up at time 21. Then, it turns on its transceiver and passes the data to D .

In intermittently connected networks, a new kind of packet delivery delay called *buffer delay* is introduced. Buffer delay is the duration from the moment when a packet is available for sending to the time when the packet is successfully sent out. For instance, in Example 1.1, the buffer delay of the packet at node A is $21 - 11 = 10$. Since buffer delay (in the order of seconds) is much longer than other kinds of delays (e.g., processing delay, transmission delay and propagation delay, which are in the order of milliseconds), we only consider buffer delay in this paper.

In our model, each sensor node does not need to know its own location. For simplicity of discussion, we assume that each node has the same transmission power level. We discuss the varying power level scenario in Section VII.

III. Multicast in Sensor Networks

In this section, we define the problem of finding optimal multicast trees in sensor networks. We start with single-hop multicast, and then extend it to the general multi-hop scenario. By default, we assume energy-conservation scheduling schemes are employed, since the scenario where there is no scheduling scheme can be considered as a special case when $T_w = T_r$.

A. Single-hop Multicast

In single-hop multicast, a packet is delivered from the source to part (or all) of its one-hop neighbors. If all nodes are always connected (i.e., no energy conservation scheduling), this problem is trivial because the broadcast nature of the wireless link allows the source to disseminate the packet to all the neighbors through a single transmission. However, if the network is intermittently connected, the problem becomes nontrivial.

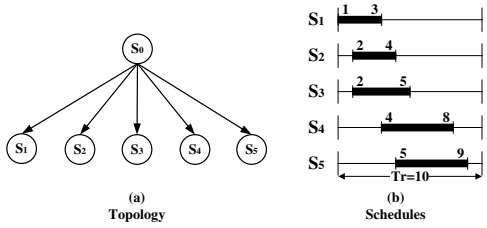


Fig. 2. An example of single-hop multicast

Example 3.1: For example, suppose the network shown in Fig. 2(a) is used to monitor the target region for $H = 10$ hours. Unfortunately, none of the sensor nodes has such a long lifetime. Specifically, their lifetimes are: $L_1 = 3$ hours, $L_2 = 3$ hours, $L_3 = 4$ hours, $L_4 = 5$ hours, $L_5 = 5$ hours. A feasible solution is to schedule the activity of each node and let the length of its working period be $T_r \times \frac{L_i}{H}$. For each sensor node, the start time of its working period can be random or deterministic, depending on the scheduling scheme. In this case, we assume each round has $T_r = 10$ slots, and show the schedule of each node in Fig. 2(b).

As can be seen, the time slot at which the source transmits the packet decides which neighbors can receive it. For example, if the source sends a packet at the first time slot, only node s_1 can receive it because all others are in sleep. We formally define the set of the nodes which can receive the transmitted packet at a specific time slot as follows:

Definition 1: Forwarding Set. If node s_i transmits a packet at time slot t, s_i's *forwarding set* at t, denoted as $\mathcal{F}_{i,t}$, includes all s_i's neighbors that can receive the packet.

In the above example, $\mathcal{F}_{0,1} = \{s_1\}$ by our definition. The upper bound of the number of different forwarding sets a sensor can have is given below:

Theorem 1: For any sensor node s_i, it has at most $2|N(s_i)|$ different forwarding sets, where $N(s_i)$ denotes the neighbor set of s_i.

Proof: We call both the start and the end slot of a sensor node as switch slots. Apparently, the forwarding sets at the slots between two consecutive switch slots of s_i's neighbors are identical. For instance, as shown in Fig. 2(b), $s_2.start = 2$ and $s_1.end = 3$ are two consecutive switch slots, and thus $\mathcal{F}_{0,2} = \mathcal{F}_{0,3} = \{s_1, s_2, s_3\}$. Also, since different neighbors may have the same switch slots, duplicate forwarding sets may be recorded. Therefore, the number of s_i's different forwarding sets cannot be larger than $2|N(s_i)|$. ■

Therefore, minimizing the number of transmissions to deliver a packet from the source to part (or all) of its one-hop neighbors is equivalent to *minimizing the number of forwarding sets which can collectively cover these destination neighbors*.

Algorithm 1 Optimal Forwarding Sets Selection Algorithm

Input: (s_0's Forwarding Sets $\mathcal{F}_{0,t_i}, 1 \le i \le m_0$, Destination Set $\mathcal{D}$);
Output: (Optimal Multicast Set $\mathcal{OS}$);

```
1: i ← 1
2: while D ≠ ∅ do
3:     if (F_{0,t_i} ⋂ D) ⊂ (F_{0,t_{i+1}} ⋂ D) then
4:         i++
5:     else
6:         OS ← OS ⋃ {F_{0,t_i}}
7:         D ← D − F_{0,t_i}
8:         i++
9: return OS
```

Example 3.2: According to the schedules illustrated in Fig. 2(b), we record the forwarding sets at the switch slots ($s_i.start$ and $s_i.end + 1$) of each neighbor node, and show them in Table I. After eliminating the duplicate and empty sets, there are six candidate forwarding sets. Subsequently, we run a simple iterative algorithm (Algorithm 1[1]) on these sets. The algorithm scans the forwarding sets in chronological order (i.e.,$t_i < t_{i+1}$), and in each iteration, it checks whether the uncovered destinations contained in the current forwarding set are included in the next forwarding set (line 3). Finally, $\mathcal{F}_{0,2}$ and $\mathcal{F}_{0,5}$ are selected as the optimal multicast set.

TABLE I
FORWARDING SETS

Recording Time	Time Slot	Forwarding Set	Candidate FS	Destination Set
$s_1.start$	$slot1$	$\{s_1\}$	$\mathcal{F}_{0,1}$	$\{s_1, s_2, s_3, s_4, s_5\}$
$s_2.start$	$slot2$	$\{s_1, s_2, s_3\}$	$\mathcal{F}_{0,2}$✓	$\{s_1, s_2, s_3, s_4, s_5\}$
$s_3.start$	$slot2$	$\{s_1, s_2, s_3\}$		
$s_1.end + 1$	$slot4$	$\{s_2, s_3, s_4\}$	$\mathcal{F}_{0,4}$	$\{s_4, s_5\}$
$s_4.start$	$slot4$	$\{s_2, s_3, s_4\}$		
$s_2.end + 1$	$slot5$	$\{s_3, s_4, s_5\}$	$\mathcal{F}_{0,5}$✓	$\{s_4, s_5\}$
$s_5.start$	$slot5$	$\{s_3, s_4, s_5\}$		
$s_3.end + 1$	$slot6$	$\{s_4, s_5\}$	$\mathcal{F}_{0,6}$	
$s_4.end + 1$	$slot9$	$\{s_5\}$	$\mathcal{F}_{0,9}$	
$s_5.end + 1$	$slot10$	$\{\}$		

B. Multi-hop Multicast

Example 3.3: Fig. 3(a) shows a simple multi-hop multicast scenario where the source node s_0 tries to deliver a packet to nodes s_3, s_4 and s_5. Suppose only nodes s_4 and s_5 have overlapped working schedules. Then, there are only two possible ways to forward the packet, i.e., two different multicast trees, shown in Fig. 3(b) and (c), respectively. Apparently, the latter is better in terms of the number of transmissions, because s_2 can deliver the packet to s_4 and s_5 through a single transmission due to their schedule overlap.

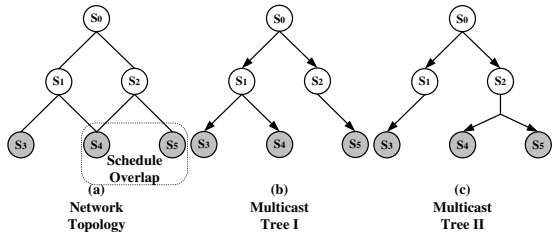


Fig. 3. An example of multi-hop multicast

When the network scales, the problem becomes intractable. To address this problem, we propose oCast which is explained in the next section.

IV. THE OPTIMAL MULTICAST ALGORITHM (OCAST)

A. Graph Transformation

It is well known that in wired networks, determining a minimum-energy multicast tree is equivalent to finding a minimum-cost Steiner tree (the cost of a tree is the sum of its edges' weights) in the network graph. However, as disclosed in [15], this conclusion does not hold in wireless applications because of the broadcast nature of the wireless links, which is referred to as the "wireless multicast advantage". Furthermore, the network graph is only based on the geographical distances of the node pairs, and cannot reflect the topology dynamics

[1]In this algorithm, we assume the synchronization among the sensor nodes, which is trivial for single-hop networks to achieve. Under this assumption, in s_0's point of view, each of its neighbors has a single duration of working period within a round, i.e., $s_i.start < s_i.end$. However, our oCast algorithm, which is designed for multi-hop scenario, does not have such an assumption.

when scheduling schemes are employed. Thus, it is impossible to solve the multicast problem directly based on the network graph. To address this problem, we transform the original network graph to a directed graph which can characterize both the wireless multicast advantage and the intermittently connected topology.

Given the undirected network graph $G = (V, E)$, we construct a directed graph $G' = (V', E')$ by substituting each node $s_i \in V$ with a ***widget***, denoted by $W_i = (V_i, E_i)$. W_i is illustrated in Fig. 4, and defined as follows:

- $V_i = \{s_i, f_{i,t_1}, f_{i,t_2}, ..., f_{i,t_{m_i}}\}$, where s_i is the original sensor node in V, and f_{i,t_j} $(1 \le j \le m_i)$ corresponds to the forwarding set of s_i (suppose s_i has m_i different forwarding sets) at time t_j. In the rest of this paper, s_i is referred to as the *sensor vertex*, and f_{i,t_j} is referred to as the *forwarding vertex*.
- $E_i = \{(s_i, f_{i,t_j}) | 1 \le j \le m_i\}$, i.e., the sensor vertex s_i has a directed edge to each of its forwarding vertices. Each edge $(s_i, f_{i,t_j}) \in E_i$ has a weight of 1 since s_i can deliver a packet to all the nodes in the forwarding set $\mathcal{F}_{i,t_j}$ with one transmission.
- For each forwarding vertex of s_i, say f_{i,t_j}, it has a directed edge to each sensor vertex in its corresponding forwarding set. We set the weights of these edges as 0 and denote the set of them by E_{i,t_j}.

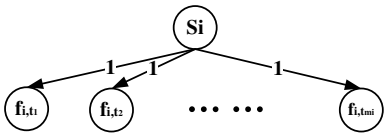


Fig. 4. The widget for node s_i

With a widget for each node in G, we can construct G' by letting $V' = \bigcup_{i=1}^{n} V_i$ and $E' = \bigcup_{i=1}^{n} (E_i \bigcup_{j=1}^{m_i} E_{i,t_j})$. Furthermore, for each $v \in V'$, we define $N_{in}(v)$ as the set of vertices which have outgoing edges to v, and $N_{out}(v)$ as the set of vertices which have incoming edges from v.

By this transformation, we translate the multicast problem into the well known *directed Steiner tree problem*. We apply this transformation to the examples discussed previously. Fig. 5 illustrates the derived graph for single-hop topology shown in Fig. 2(a). In this graph, there are six forwarding vertices, corresponding to six different forwarding sets listed in Table I. There is a 1-weighted edge from the source sensor vertex to each forwarding vertex, and a 0-weighted edge from a forwarding vertex to each of its member sensor vertices. A minimum directed Steiner tree originated from vertex s_0 is depicted in the figure (the red dashed nodes and edges). As can be seen, the Steiner tree spans the forwarding vertices $f_{0,2}$ and $f_{0,5}$, implying that the forwarding sets at slot 2 and 5 should be selected. This matches the result obtained by Algorithm 1.

The derived graph of multi-hop topology (Fig. 3(a)) is illustrated in Fig. 6. Since only nodes s_4 and s_5 have overlapped working schedules, there is only one forwarding vertex, i.e., $f_{2,2}$, which is incident on more than one sensor vertex. Therefore, the minimum directed Steiner tree (red dashed nodes and edges) originated from node s_0 spans this vertex

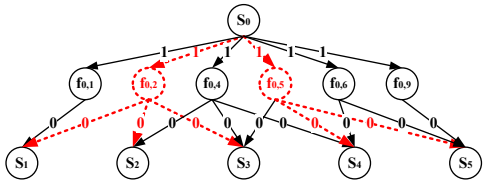


Fig. 5. Derived graph of single-hop topology

because it can cover two destinations at the cost of only 1 transmission.

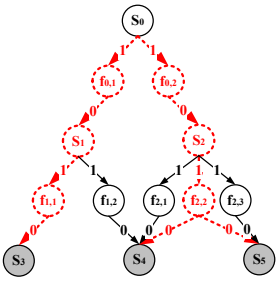


Fig. 6. Derived graph of multi-hop topology

Although constructing a minimum directed Steiner tree is NP-hard, the small multicast group nature of sensor networks makes it possible to find an optimal directed Steiner tree in G' with polynomial time.

B. oCast Algorithm

oCast borrows the idea of dynamic programming. It starts with recording each destination vertex as a simplest tree, and subsequently expands the trees by either increasing the heights of trees or merging the different trees rooted at the same vertex. This expansion process is repeated until for any vertex $v \in V'$ and any destination subset $D \subseteq \mathcal{D}$, the optimal Steiner tree rooted at v which covers D is found.

Before describing oCast in detail, we first introduce some notations: (1) Let $T(v, D)$ denote the minimum directed Steiner tree originated at vertex v and spanning all the destination vertices in the subset D. (2) Let $C(T)$ be a function denoting the cost of tree T.

In oCast, we construct $T(v, D)$ recursively as described by Eqn. (1). Initially, each vertex $v \in V'$ is a simplest rooted tree with zero height. If v is one of the destinations, we set $C(T(v, \{v\}))$ to be 0. Otherwise, we set D to be $\emptyset$ and $C(T(v, D))$ to be ∞. After that, the trees are constructed through two different tree expansion operations: *tree grow* and *tree merge*, which are precisely explained as follows:

- **Tree grow:** As shown in Eqn. (2), $\oplus$ denotes an operation of attaching a directed edge (v, u) to $T(u, D)$, a tree rooted at u and containing D (as shown in Fig. 7(a)). Eqn. (2) picks the tree with the smallest cost and marks it as $T_g(v, D)$. As a special case, if v is an element of D, the last term on the right-hand side is changed to $T(u, D - \{v\})$.
- **Tree merge:** Tree merge operation is described by Eqn. (3). In this case, $\oplus$ denotes an operation of merging two trees rooted at the same vertex into a new tree (as shown in Fig. 7(b)). Eqn. (3) exhaustively checks all the

possible D_1 and D_2 pairs, and picks the one which can minimize the cost of $T_m(v, D)$.

$$C(T(v,D)) = \begin{cases} 0 & \text{if } D = \{v\}, \\ \infty & \text{if } D = \emptyset, \\ \min(C(T_g(v,D)), C(T_m(v,D))) & \text{otherwise.} \end{cases} \quad (1)$$

Where

$$C(T_g(v,D)) = \begin{cases} \min_{u \in N_{out}(v)} \{C((v,u) \oplus T(u,D))\} & \text{if } v \notin D, \\ \min_{u \in N_{out}(v)} \{C((v,u) \oplus T(u, D - \{v\}))\} & \text{if } v \in D. \end{cases} \quad (2)$$

$$C(T_m(v,D)) = \min_{D_1 \bigcup D_2 = D \text{ and } D_1 \bigcap D_2 = \emptyset} \{C(T(v,D_1) \oplus T(v,D_2))\} \quad (3)$$

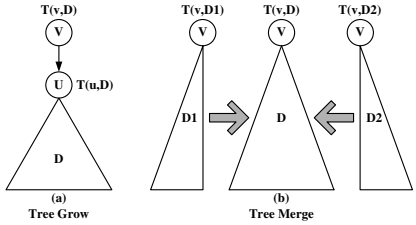


Fig. 7. Tree Expansion

Finally, we assign the one with smaller cost among $T_g(v, D)$ and $T_m(v, D)$ to $T(v, D)$, which is illustrated in Eqn. (1). In the following, we first prove the correctness of tree merge operation by Theorem 2, then prove the correctness of oCast by Theorem 3.

Theorem 2: Suppose tree T is obtained through merging T_1 and T_2 based on Eqn. (3), then there is no common vertex shared by T_1 and T_2.

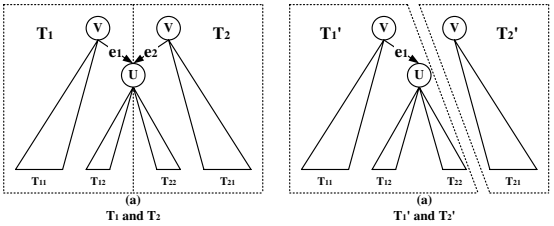


Fig. 8. Correctness of Tree Merge

Proof: By contradiction, suppose T_1 and T_2 share a common vertex u, and $T_1 = T_{11} \oplus T_{12}$ and $T_2 = T_{21} \oplus T_{22}$ as illustrated in Fig. 8(a). Obviously, $T_1' = T_{11} \oplus T_{12} \oplus T_{22}$ and $T_2' = T_{21}$ (as shown in Fig. 8(b)) is another possible tree pair which can construct T. It is easy to see that if the weight of edge e_2 is nonzero (If its weight equals 0, e_2 is just a virtual edge, and thus it does not matter whether it is included), then $C(T_1 \oplus T_2) > C(T_1' \oplus T_2')$. Therefore, the assumption does not hold, and the theorem is proved. ■

Theorem 3: Given the derived graph G', and a set of d destination vertices, which is $\mathcal{D} = \{d_1, d_2, \cdots, d_d\}$. The optimal directed Steiner tree which is rooted at any vertex $v \in V'$ and spans any destination subset $D \subseteq \mathcal{D}$, i.e., $T(v, D)$, can be identified through Eqn. (1).

Theorem 3 can be proved by induction on $|D|$. Due to space limit, we omit the detailed proof.

oCast: The pseudo code of the oCast algorithm is shown in Algorithm 2. In oCast, trees are maintained in a min-priority queue Q_T in the increasing order of costs. Initially, oCast sets

Algorithm 2 oCast

Input: (Derived Directed Graph $G' = (V', E')$, Destination Set $\mathcal{D}$);
Output: (Optimal Steiner Tree Set $\mathcal{TS}$);

1: Let Q_T be a priority queue sorted in the increasing order of the costs of trees;
2: $Q_T \leftarrow \emptyset$;
3: **for** each $v \in V'$ **do**
4: **if** $v \in \mathcal{D}$ **then**
5: Enqueue $T(v, \{v\})$ into Q_T;
6: **while** $Q_T \neq \emptyset$ **do**
7: Dequeue Q_T to $T(v, D)$;
8: $\mathcal{TS} \leftarrow \mathcal{TS} \bigcup T(v, D)$
9: **for** each $u \in N_{in}(v)$ **do**
10: **if** $C((u,v) \oplus T(v,D)) < C(T(u,D))$ **then**
11: $T(u, D) \leftarrow (u, v) \oplus T(v, D)$;
12: Update Q_T with the new $T(u, D)$;
13: $D_1 \leftarrow D$;
14: **for** each D_2 satisfying $D_1 \bigcap D_2 = \emptyset$ **do**
15: **if** $C(T(v,D_1) \oplus T(v,D_2)) < C(T(v, D_1 \bigcup D_2))$ **then**
16: $T(v, D_1 \bigcup D_2) \leftarrow T(v, D_1) \oplus T(v, D_2)$;
17: Update Q_T with the new $T(v, D_1 \bigcup D_2)$;
18: **return** $\mathcal{TS}$

Q_T to be empty (line 2). Subsequently, it enqueues the single-vertex tree $T(v, \{v\})$ into Q_T if vertex v is a destination (line 3-5). In each iteration of the while loop (line 6-17), if the queue Q_T is non-empty, the algorithm dequeues the top tree $T(v, D)$ (i.e., the tree with the smallest cost) from Q_T, and inserts it into $\mathcal{TS}$ (line 7-8).

Afterwards, oCast carries out the tree grow operation in line 9-12. The algorithm examines each of v's incoming neighbor vertices, say u, to see whether it can improve the tree rooted at u and containing the destination subset D, by attaching edge (u, v) directly to $T(v, D)$. If so, $T(u, D)$ will be updated by $(u, v) \oplus T(v, D)$ (line 11-12). If $T(u, D)$ does not originally exist in Q_T, it will be enqueued as a new element into Q_T.

The tree merge operation is performed in line 13-17. The algorithm considers every possible pair of disjoint subsets, denoted by D_1 and D_2, and tries to reduce the cost of $T(v, D_1 \bigcup D_2)$. To achieve this, the algorithm fixes D_1 as D (line 13) and exhaustively checks all the D_2 which satisfies $D_1 \bigcap D_2 = \emptyset$ to see whether the tree $T(v, D_1) \oplus T(v, D_2)$ has a smaller cost than $T(v, D_1 \bigcup D_2)$ (line 14-15). If so, $T(v, D_1 \bigcup D_2)$ will be updated by $T(v, D_1) \oplus T(v, D_2)$ (line 16-17). If $T(v, D_1 \bigcup D_2)$ does not originally exist in Q_T, it will be enqueued as a new element into Q_T.

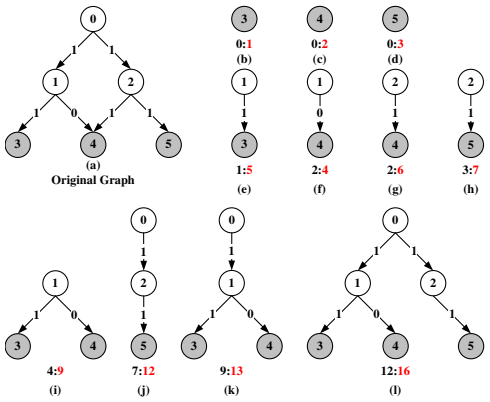


Fig. 9. An example of oCast

Example 4.1: Fig. 9 shows how the algorithm works with an example. As shown in the figure, vertex 3, 4 and 5 are designated as destinations, and each small subgraph represents a Steiner tree identified by the algorithm. Two positive integers, separated by a colon, are associated with each tree. The former one denotes the iteration of the algorithm in which this tree is enqueued into Q_T, and the latter one shows the iteration in which this tree is dequeued.

During the initialization phase, i.e., the 0th iteration, each destination vertex is enqueued into Q_T as a single-vertex tree (Fig. 9(b) to (d)). After three iterations, all the single-vertex trees are dequeued and four 1-height trees are constructed through tree grow and they are enqueued into Q_T (Fig. 9(e) to (h)). Then, the optimal tree rooted at vertex 1, containing destinations 3 and 4, is constructed by merging trees (edges) $(1,3)$ and $(1,4)$ in the 4th iteration (Fig. 9(i)), and it is further expanded to vertex 0 in the 9th iteration (Fig. 9(k)). The single-path tree from vertex 0 to destination 5 is found in the 7th iteration (Fig. 9(j)). Finally, the tree which is originated at vertex 0 and covers all the destinations is constructed in the 12th iteration (Fig. 9(l)). After this tree is dequeued in the 16th iteration, Q_T becomes empty and the algorithm terminates.

Actually, during the whole 16 iterations, 16 different optimal Steiner trees are identified and inserted into $\mathcal{TS}$. Due to space limitation, we only show part of them.

C. Computational Complexity of oCast

In this subsection, we analyze the computational complexity of oCast, and explain why oCast can identify the optimal multicast trees in polynomial time with respect to n.

1) Computational Complexity: Let $n' = |V'|$ and $m' = |E'|$, the overall cost of oCast consists of three parts: (1) dequeue. (2) tree grow. (3) tree merge.

- **Dequeue:** With Fibonacci Heap, the cost of each dequeue operation is $O(\log 2^d n')$, and thus the total cost needed for the dequeue operations is $O(2^d n'(d + \log n'))$.
- **Tree Grow:** The total number of comparisons in the tree grow operations is $O(2^d \Sigma_{v \in V'} |N_{in}(v)|) = O(2^d m')$. Since the enqueue/update of Q_T takes $O(1)$ time, the total time is $O(2^d m')$.
- **Tree Merge:** The number of comparisons is $O(n' \sum_{i=0}^{d} \binom{d}{i} 2^{d-i}) = O(3^d n')$, and thus the total time needed for all the tree merge operations is $O(3^d n')$.

In summary, the overall computational complexity is $O(3^d n' + 2^d((d + \log n')n' + m'))$.

2) Analysis of Graph Transformation: Suppose the average size of neighbor sets of the nodes in the whole network is $\bar{n} = \frac{\sum_{i=1}^{n} |N(s_i)|}{n}$. By Theorem 1, in each widget, there are on average $O(\bar{n})$ different forwarding vertices, we can get $n' = \sum_{i=1}^{n} |V_i| = O(n(1 + \bar{n})) = O(n\bar{n})$.

Apparently, the size of each forwarding set is $O(\bar{n})$, and thus each forwarding vertex has on average $O(\bar{n})$ outgoing edges. From this hint, it can be inferred that $m' = \sum_{i=1}^{n} |E_i \bigcup_{j=1}^{|E_i|} E_{i,t_j}| = \sum_{i=1}^{n} |E_i| + \sum_{i=1}^{n} |\bigcup_{j=1}^{|E_i|} E_{i,t_j}| = O(n\bar{n}) + O(n\bar{n}^2) = O(n\bar{n}^2)$.

In sensor networks, $\bar{n}$ is normally small due to the low density of sensor nodes, and thus can be regarded as a constant. Therefore, $n' = O(n)$ and $m' = O(n)$. Furthermore, since d can be considered as a constant in small group multicast, the computational complexity of oCast is actually a polynomial function of n.

V. Delay Bounded oCast (DB-oCast)

Although oCast can minimize the energy consumption, it may not be able to achieve satisfactory *end-to-end packet delivery delay*, which is defined as the period from the moment when the source node has a packet to send to the time when every destination in the multicast group successfully receives the packet[2].

Example 5.1: As shown in Fig. 10, suppose at time 1, sensor s_0 receives a packet and wants to forward it to its five neighbors. It has three different forwarding sets, which are shown in the derived graph. Obviously, the most energy efficient way to disseminate the packet is to transmit it to $\mathcal{F}_{0,4}$ at slot 4 because only a single transmission is needed. However, if there is a delay constraint, e.g., the communication must be done before time 3, this solution is not valid. To satisfy the delay constraint, $\mathcal{F}_{0,2}$ and $\mathcal{F}_{0,3}$ are used to construct the optimal multicast set, although they have one more transmission.

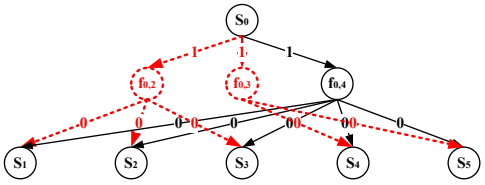


Fig. 10. An example of delay bounded multicast

Energy and delay are two conflicting parameters, and thus it is very difficult to satisfy one goal without compromising the other. Our objective is to minimize the number of transmissions, while guaranteeing its delay is bounded by a predefined threshold. This problem is equivalent to finding *the minimum delay bounded directed Steiner tree (DB-Steiner tree)* in G'. In this section, we will introduce Delay Bounded oCast (DB-oCast), which can effectively identify DB-Steiner trees.

A. DB-oCast Algorithm

In DB-oCast, the DB-Steiner tree is constructed recursively by Eqn. (4). By adding a *delay bound parameter* t, $T(v, D, t)$ denotes the optimal Steiner tree whose delay is bounded by t. Initially, $C(T(v, \{v\}, 0))$ is set to be 0, and $C(T(v, \emptyset, 0))$ is set to be ∞. Similar to oCast, the trees are expanded through either tree grow or tree merge (in DB-oCast, we call them *delay bounded tree grow* and *delay bounded tree merge*, respectively).

- **DB tree grow:** As described in Eqn. (5), it exhaustively examines each $u \in N_{out}(v)$ to see whether $T_g(v, D, t)$

[2]Sometimes, packet congestion may cause unpredictable delay as well. Firstly, in sensor networks, packet traffic is normally light. Secondly, in this paper, we focus on the delay that depends on the path (i.e., the delay caused by the disconnected network topology). The delay depending on the load is not our concern and can be addressed by specific load control element like an admission or rate controller.

can be improved by attaching (v,u) to $T(u,D,t-t(v,u))$ (or $T(u,D-\{v\},t-t(v,u))$), a tree rooted at u which holds the optimal cost under the delay constraint of $t-t(v,u)$. Here $t(v,u)$ denotes the delay for a packet to go through edge (v,u), and will be further explained later.

- **DB tree merge:** As described in Eqn. (6), all possible $T(v,D_1,t)$ and $T(v,D_2,t)$ pairs are checked to minimize the cost of $T_m(v,D,t)$.

$$C(T(v,D,t)) = \begin{cases} 0 & \text{if } D=\{v\} \text{ and } t=0, \\ \infty & \text{if } D=\emptyset \text{ and } t=0, \\ \min(C(T_g(v,D,t)), & \\ C(T_m(v,D,t)), & \\ C(T(v,D,t-1))) & \text{otherwise.} \end{cases} \tag{4}$$

Where

$$C(T_g(v,D,t)) = \begin{cases} \min_{u\in N_{out}(v)}\{ & \\ C((v,u)\oplus T(u,D,t-t(v,u)))\} & \text{if } v\notin D, \\ \min_{u\in N_{out}(v)}\{ & \\ C((v,u)\oplus T(u,D-\{v\},t-t(v,u)))\} & \text{if } v\in D. \end{cases} \tag{5}$$

$$C(T_m(v,D,t)) = \min_{D_1\bigcup D_2=D \text{ and } D_1\bigcap D_2=\emptyset}\{C(T(v,D_1,t)\oplus T(v,D_2,t))\} \tag{6}$$

After the tree grow and tree merge operations are done, Eqn. (4) compares the generated trees to the tree with 1-slot lower delay bound, i.e., $T(v,D,t-1)$, and update $T(v,D,t)$ with the best one.

Now, we clarify how the delay function of directed edges in G' is calculated: Each $v\in V'$ has a receiving time, which denotes the moment when the packet is received by v. We denote the receiving time of v by $v.rtime$, and thus the delay of directed edge (v,u) can be calculated by $t(v,u)=u.rtime-v.rtime$ (if $u.rtime\geq v.rtime$) or $t(v,u)=u.rtime+T_r-v.rtime$ (if $u.rtime<v.rtime$).

For a forwarding vertex, its receiving time is actually the time when its corresponding forwarding set is recorded. On the other hand, the determination of a sensor vertex's receiving time is much more complicated. According to the construction of G', the time when a sensor vertex receives packets is determined by the forwarding vertices connected to it. However, a sensor vertex may have multiple incoming edges from different forwarding vertices, resulting in multiple possible receiving times.

To solve this problem, we split each sensor vertex into a group of new vertices, and each new vertex corresponds to a possible receiving time of the original sensor vertex. Formally, suppose s_i is a sensor vertex, and it has l_i different receiving times. After transformation, it will be split into l_i new vertices, which form a small vertex group. We denote this group by S_i, and the l_i new vertices in S_i by $s_{i,t_j}(1\leq j\leq l_i)$ (where t_j represents a receiving time). These new sensor vertices, together with the forwarding vertices in V', construct a new graph, $G''=(V'',E'')$. In G'', the edges are defined as follows:

- For each sensor vertex $s_i\in V'$, suppose $f_{i,t}$ is one of s_i's forwarding vertices, $(s_{i,t_j},f_{i,t})\in E''(1\leq j\leq l_i)$. Moreover, the delay of the edge is $t(s_{i,t_j},f_{i,t})=t-t_j$ (or $t+T_r-t_j$).
- For each forwarding vertex $f_{j,t}\in V'$ (suppose t denotes its receiving (recording) time), if edge $(f_{j,t},s_i)\in E'$, then $(f_{j,t},s_{i,t})\in E''$. Obviously, $t(f_{j,t},s_{i,t})=0$.

Example 5.2: Fig. 11 shows how to construct G''. Suppose each of sensor vertices X, Y, Z has a forwarding vertex connected to s_0, as illustrated in Fig. 11(a). Since the forwarding vertex from X is recorded at time 1, whereas the forwarding vertices from Y and Z are both recorded at time 5, vertex s_0 has two possible receiving times. Therefore, in G'' (Fig. 11(b)), s_0 is split into two different vertices. Among them, $s_{0,1}$ corresponds to receiving time 1, and thus is connected by the forwarding vertex from X. $s_{0,5}$ corresponds to receiving time 5, and thus is connected by the forwarding vertices from Y and Z. In addition, both $s_{0,1}$ and $s_{0,5}$ retain the connections to the forwarding vertices of s_0. With G'', we can easily figure out the delays of the edges, for example $t(f_{X,1},s_{0,1})=0$ and $t(s_{0,5},f_{0,2})=2+T_r-5=7$ (Suppose $T_r=10$).

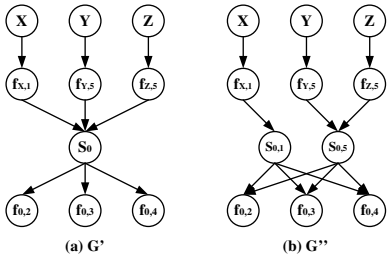


Fig. 11. Split of sensor vertex

In G'', each vertex has a fixed receiving time, thus we can further transform the problem of finding the minimum delay bounded directed Steiner tree in G' into the problem of finding *the minimum delay bounded directed group Steiner tree* in G''. Group Steiner tree problem is another well known NP-hard problem [16], [17]. Its major difference from the standard Steiner tree problem includes: (1) Each destination of the standard Steiner tree is a single vertex, whereas each destination of the group Steiner tree is a group (set) of vertices. (2) The group Steiner tree only needs to cover at least one vertex of each destination group.

In our transformation, each destination vertex s_i in G' has a corresponding destination group S_i in G''. In DB-oCast, the minimum group Steiner tree T must satisfy $V(T)\bigcap S_i\neq\emptyset$ for any $S_i\in\mathcal{D}$, where $V(T)$ denotes the vertices spanned by T.

We roughly describe DB-oCast as follows: the algorithm starts to iterate on the delay bound t, from 0 to τ, *the predefined maximum bound*. Under each bound, the algorithm expands the group Steiner trees through either DB tree grow operations (Eqn. 5) or DB tree merge operations (Eqn. 6). Finally, all the optimal group Steiner trees will be identified.

B. Computational Complexity and Delay Granularity

The overall computational complexity of DB-oCast is $O(\tau(2^d m''+3^d n''))$, where $n''=|V''|=O(n\bar{n})$ and $m''=|E''|=O(n\bar{n}^2)$. Since $\bar{n}$ and d are considered as constants in small group multicast, the computational complexity of DB-oCast is actually a polynomial function of n.

From the computational complexity analysis of DB-oCast, we can see that it's approximately τ times larger than that of oCast. This is because in the recursive formula of DB-oCast, a new dimension of delay bound is added. If the maximum delay bound of the network is relatively large, for instance 1000 time slots, the computational complexity of DB-oCast would be unacceptably large. However, in the simulations, we find that the minimum DB Steiner tree does not change frequently with the increase of the delay bound. For example, a minimum DB Steiner tree whose delay is 150 may still remain optimal when the delay bound increases to 170. Therefore, it is not necessary to calculate the DB Steiner trees for each bound, and we can greatly reduce the computational complexity by increasing the granularity of the delay bound, without losing too much accuracy.

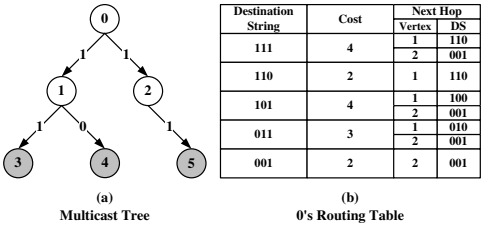
VI. Distributed Implementation

In this section, we demonstrate that oCast and DB-oCast can be easily applied in a decentralized fashion. Distributed oCast borrows the idea of Distance Vector routing algorithm. Each vertex in G' maintains an *oCast routing table*, and periodically broadcasts a distance vector, which contains a "digest" of the information in its oCast routing table.

An example of oCast routing table is illustrated in Fig. 12. In this figure, a multicast tree is shown on the left and part of vertex 0's oCast routing table is shown on the right. In the routing table, the first column contains the *Destination String*[3](*DS*) of each destination subset which is used to uniquely identify an entry, and the second column displays the cost of the corresponding Steiner tree. The third column of the table stores the information of next hops. It includes two sub-columns. The first sub-column represents the ID of each next-hop vertex, and the second sub-column indicates the corresponding destination set which is expected to be covered by the subtree rooted at the next-hop vertex. In each entry, there may be multiple next-hop vertices, which implies that the packet be forwarded to multiple subtrees.

In this scenario, the sorted super destination set comprises vertex 3, 4 and 5, and thus they correspond to the 1st, 2nd and 3rd bit of the destination string, respectively. Suppose vertex 0 wants to multicast a packet to all the desired destinations. Based on the first entry of the routing table (next hop part), it should send the packet to both vertex 1 and 2. Also, it will encode the destination strings (110) and (001) in the headers of the packets to vertex 1 and 2, respectively. Since the length of a DS equals the size of the super destination set, which is a small number, this kind of destination encoding won't cause too much communication overhead. Subsequently, after receiving the packet, vertex 1 and 2 will decode the DS, and forward the packet along their own subtrees according to the decoded DS.

In our distributed oCast algorithm, neighboring vertices periodically exchange distance vectors. The distance vector consists of the costs of the Steiner trees (i.e., first two columns)



Destination String	Cost	Next Hop	
		Vertex	DS
111	4	1	110
		2	001
110	2	1	110
101	4	1	100
		2	001
011	3	1	010
		2	001
001	2	2	001

Fig. 12. An example of oCast routing table

in the routing table. To address the routing loops and count-to-infinity problem caused by link-cost changes or link failures, we can borrow the idea of DSDV [18], and add sequence numbers to the distance vectors.

We imitate the study of the communication complexity of DSDV in [19] to analyze the message overhead of oCast. In conclusion, the overall communication overhead of oCast is from $O(2^d n')$ (best case) to $O(2^d n'^2)$ (worst case). Therefore, the average communication overhead to identify a Steiner tree is from $O(1)$ (best case) to $O(n')$ (worst case), given the number of Steiner trees to be $2^d n'$.

However, the above description and analysis are based on G', a virtual graph. In reality, there exist only n sensor nodes which form the network graph G. Our solution is to let each sensor node, say s_i, delegate its corresponding widget, i.e., W_i in G'. In other works, s_i locally maintains not only its own routing table and distance vector, but also each of its forwarding vertices' routing table and distance vector. In this way, the total communication overhead drops to $O(2^d n)$ (best case) to $O(2^d n^2)$ (worst case).

DB-oCast can be decentralized in a similar way. One problem is the communication overhead caused by DB-oCast would be τ times larger than that of oCast. As explained in the last section, by carefully selecting the delay granularity, we can reduce this overhead.

VII. Discussions

A. Imperfect Link

In our discussions and examples, we assume perfect link quality. If the link quality is not perfect, retransmissions may occur[4]. A simple solution is to increase the length of time slots to accommodate retransmissions. In addition, each node should estimate the expected number of transmissions for each link, and use these numbers as the weights of the edges in the network graph. As aforementioned, we focus on the buffer delay, which can be in the order of seconds, and thus the delay caused by these retransmissions (in the order of milliseconds) can be ignored.

B. Varying Power Level

In this paper, sensor nodes have a fixed power level, which is different from some multicast schemes developed for ad hoc networks [1], [2], [20]. This assumption is used to simplify the discussion. Also, varying the power level is not widely employed in sensor networks. Since a node may be involved in

[3]A DS is used to describe a subset of the destination set. For instance, a DS of (100) means that only the first destination appears in the subset.

[4]Retransmission can be easily achieved by allowing receivers to explicitly send ACKs back.

multiple multicast sessions simultaneously, frequently changing its power level may incur some implementation problems on the existing sensor platforms.

However, our oCast can be easily adapted to the varying power level applications. The only change occurs in the transformation from G to G'. In each widget W_i of G', we connect some *virtual power vertices* corresponding to the alternative power levels to the sensor vertex, and each power vertex is incident on s_i's forwarding vertices. Subsequently, we apply the oCast algorithm to the new G', and the optimal solution can be obtained.

VIII. Performance Evaluations

A. Simulation Model

To evaluate the performance of different multicast protocols, we use a flexible parameterized destination placement model which includes two parameters to control the destination locations. One parameter is the *angle of dispersion (AOD)*, and the other is the *radius (or range)*. These two parameters define a pie-shaped region that contains the destinations and the source, as shown in Fig. 13.

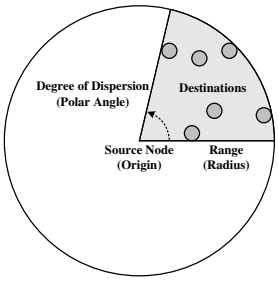


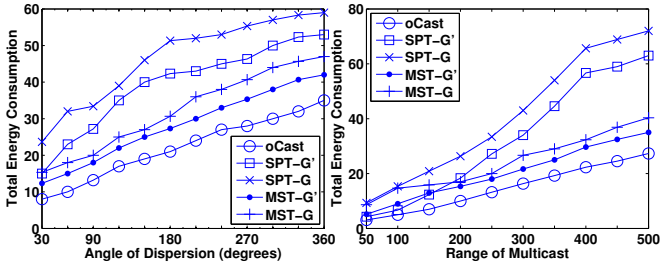
Fig. 13. Destination placement model

We compare oCast with two classic heuristics used to approximate the optimal Steiner tree: the *Shortest Path Tree (SPT)*, and the *Minimum Spanning Tree (MST)*. SPT is a naive approximation of the Steiner tree because it is simply the aggregation of the shortest paths from the source to the destinations. MST is the most widely used heuristic for Steiner tree. Its basic idea is to use a partial spanning tree to approximate the Steiner tree. As mentioned in [21], under the fixed power level model, MST is a bestcase baseline and no decentralized scheme can outperform it so far. Therefore, to demonstrate the performance advantage of oCast, we only need to outperform MST.

Unless otherwise stated, the default parameters are as follows. The communication range is $50m$, The area is $500m \times 500m$, and the density is 20 nodes per communication range. AOD is 90 degrees, and the radius of the pie shaped area is $250m$. A total of 636 nodes are deployed and the number of destinations is 10.

B. Performance Evaluation of oCast without Scheduling

Without scheduling protocols, the network topology does not change and the network is always connected. In this case, each sensor node has a single forwarding set, which includes all its neighbors. We apply SPT and MST on the original network graph G and the derived graph G', denoted as SPT-G, SPT-G', MST-G, MST-G', and compare their performances with that of oCast.



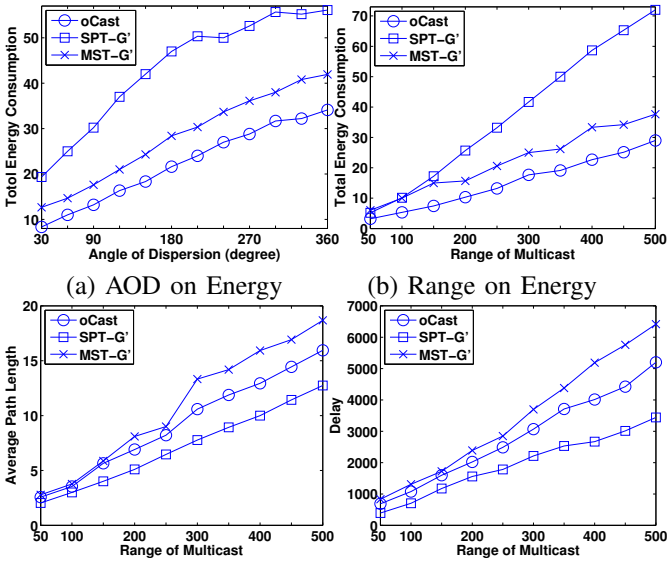
(a) AOD on Energy (b) Range on Energy

Fig. 14. Performance evaluation of oCast without scheduling protocols

The results are illustrated in Fig. 14. Fig. 14(a) and (b) show the impact of two destination placement parameters on the energy consumptions (i.e., the costs of the multicast trees) of 5 different algorithms. As expected, oCast has the best performance. We can also see that MST outperforms SPT, and both protocols perform better on G' than on G. As previously discussed, this is because G' can reflect the wireless multicast advantage.

C. Performance Evaluation of oCast with Scheduling Protocols

In this subsection, the network is intermittently connected. Therefore, we only compare SPT and MST on G' with oCast. In the simulation, the length of each round is T_r=500 slots, and the working period (T_w) of each node varies between 100 slots and 300 slots with a mean of 200 slots. Without loss of generality, each node randomly selects the start time of its working period.



(a) AOD on Energy (b) Range on Energy

(c) Range on Path Length (d) Range on Delay

Fig. 15. Performance Evaluation of oCast with Scheduling Protocols

Fig. 15(a) and (b) show the energy consumption of the three algorithms, and oCast has the best performance under either parameter setting. Fig. 15(c) compares the average path lengths of the multicast trees identified by the three algorithms. Intuitively, SPT-G' is the best one since it is actually composed of the shortest paths to each destination. On the contrary, MST-G' is the worst one since it tends to find the trees with relatively large heights. The packet delivery delays with

different multicast ranges are shown in Fig. 15(d). As can be seen, the algorithm which has shorter path length performs better.

D. Performance Evaluation of DB-oCast

In this subsection, we evaluate the performance of DB-oCast. We first set the maximum delay bound τ to be 500 slots and the delay granularity to be 1 slot, and then test DB-oCast under 40% working percentage (T_r=100 slots). As illustrated in Table II, 8 different optimal DB Steiner trees are identified. The algorithm starts with zero delay bound, and finds the first tree (shown in the leftmost column) when the delay bound reaches 83 slots. This tree has a delay of 83 and a cost of 24. It remains optimal until the delay bound is further increased to 97. This implies that the calculations between bound 84 and 96 are redundant because no change is recognized.

TABLE II
MULTICAST TREES WITH 1-SLOT-GRANULARITY

Delay Bound	83	97	175	256	302	378	423	478
Tree Cost	24	22	21	20	18	17	15	13

Fig. 16(a) shows the cost of the DB-oCast as a function of the delay bound. As can be seen, as the delay bound increases, the cost of DB-oCast drops. Eventually, at delay bound of 478, the cost of the multicast tree reaches 13. We compare DB-oCast with SPT in terms of cost and delay. As mentioned in the last subsection, SPT has smaller delay than oCast and MST. However, DB-oCast can provide trees with better cost and delay than SPT. In Fig. 16(a), the average delay and cost of SPT under the same simulation setting are shown. As we can see, when the delay bound is set to be smaller than the delay of SPT, the trees provided by DB-oCast have smaller costs than SPT.

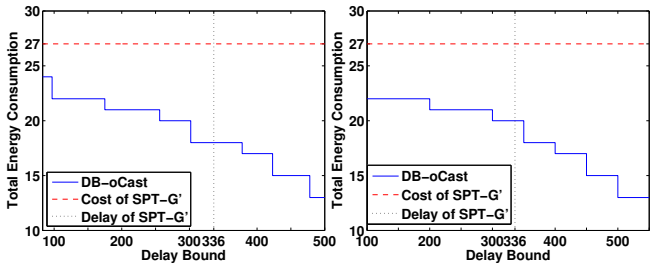
TABLE III
MULTICAST TREES WITH 50-SLOT-GRANULARITY

Delay Bound	100	200	300	350	400	450	500
Tree Cost	22	21	20	18	17	15	13

In the last experiment, the tree operations are iterated for $\tau = 500$ times, and only 8 different trees are found. To reduce the computational complexity, we increase the delay granularity to 50, and run DB-oCast again on the same topology. The final multicast trees are illustrated in Table III. This time, 7 trees are identified. Compared with the experiment with 1-slot-granularity, one multicast tree (the tree shown in the leftmost column of Table II) is missed. This is because there are two trees (whose delays are 83 and 97 respectively) from slot 51 to 100, and DB-oCast with 50-slot-granularity only picks the one with lower cost. We believe this level of accuracy loss can be tolerated in most applications. The cost curve of the trees found in this experiment is shown in Fig. 16(b). It can be seen that DB-oCast with 50-slot-granularity can still outperform STP in terms of both delay and cost.

IX. CONCLUSIONS

In this paper, we propose an effective multicast algorithm called oCast, which can construct optimal multicast trees in sensor networks. To achieve bounded delay, we propose DB-oCast, which can discover optimal multicast trees under a predefined delay bound.



(a) 1-granularity (b) 50-granularity

Fig. 16. Performance evaluation of DB-oCast

REFERENCES

[1] M. Čagalj, J.-P. Hubaux, and C. Enz, "Minimum-energy broadcast in all-wireless networks: Np-completeness and distribution issues," in *MobiCom*, 2002.

[2] W. Liang, "Constructing minimum-energy broadcast trees in wireless ad hoc networks," in *MobiHoc*, 2002.

[3] D. Li, X. Jia, and H. Liu, "Energy efficient broadcast routing in static ad hoc wireless networks," *IEEE Transactions on Mobile Computing*, vol. 3, no. 2, pp. 144–151, April-June 2004.

[4] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: a scalable and robust communication paradigm for sensor networks," in *MobiCom*, 2000.

[5] S. Shenker, S. Ratnasamy, B. Karp, R. Govindan, and D. Estrin, "Data-centric storage in sensornets," *SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 1, pp. 137–142, 2003.

[6] W. Zhang, G. Cao, and T. L. Porta, "Data dissemination with ring-based index for wireless sensor networks," *IEEE Transactions on Mobile Computing*, vol. 6, no. 7, pp. 832–847, 2007.

[7] L. Su, C. Liu, H. Song, and G. Cao, "Routing in intermittently connected sensor networks," in *ICNP*, 2008.

[8] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated coverage and connectivity configuration in wireless sensor networks," in *SenSys*, 2003.

[9] T. Yan, T. He, and J. A. Stankovic, "Differentiated surveillance for sensor networks," in *SenSys*, 2003.

[10] C. Gui and P. Mohapatra, "Power conservation and quality of surveillance in target tracking sensor networks," in *MobiCom*, 2004.

[11] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic, "Towards optimal sleep scheduling in sensor networks for rare-event detection," in *IPSN*, 2005.

[12] M. Cardei, M. Thai, Y. Li, and W. Wu, "Energy-efficient target coverage in wireless sensor networks," in *INFOCOM*, 2005.

[13] H. Liu, P. Wan, C.-W. Yi, X. Jia, S. Makki, and P. Niki, "Maximal lifetime scheduling in sensor surveillance networks," in *INFOCOM*, 2005.

[14] C. Liu and G. Cao, "Minimizing the cost of mine selection via sensor networks," in *INFOCOM*, 2009.

[15] J. E. Wieselthier, G. D. Nguyen, and A. Ephremides, "On the construction of energy-efficient broadcast and multicast trees in wireless networks," in *INFOCOM*, 2000.

[16] G.Reich and P.Widmayer, "Beyond steiners problem: A vlsi oriented generalization," in *WG*, 1989.

[17] B. Ding, J. Xu Yu, S. Wang, L. Qin, X. Zhang, and X. Lin, "Finding top-k min-cost connected trees in databases," in *ICDE*, 2007.

[18] C. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *SIGCOMM*, 1994.

[19] E. M. Royer and C. K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE Personal Communications*, vol. 6, no. 2, pp. 46–55, Apr. 1999.

[20] P.-J. Wan, G. Călinescu, and C.-W. Yi, "Minimum-power multicast routing in static ad hoc wireless networks," *IEEE/ACM Trans. Netw.*, vol. 12, no. 3, pp. 507–514, 2004.

[21] Q. Cao, T. He, and T. Abdelzaher, "ucast: Unified connectionless multicast for energy efficient content distribution in sensor networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 2, pp. 240–250, Feb. 2007.