Trading Routing Diversity for Better Network Performance

Wei Dong, *Member, IEEE*, Gonglong Chen, *Student Member, IEEE*, Xiaoyu Zhang, and Yi Gao, *Member, IEEE*

Abstract—Most sensor networks employ *distributed* and *dynamic* routing protocols. The *flexibility* that each node can choose the best forwarder from a diverse candidate set could offer excellent routing performance when the network is highly dynamic. However, it sacrifices routing *predictability* since it is possible that routing loops are frequently formed. Can we increase the network predictability by controlling the network? As a step towards solving this problem, we introduce FlexCut, a flexible approach for cutting off wireless links, which essentially limits the candidate forwarder set of each node. Unlike existing SDN solutions, FlexCut introduces flexible control over existing distributed and dynamic routing protocols. FlexCut can trade arbitrary amounts of routing diversity for better network performance by exposing to network operators a parameter which quantifies the aggressiveness. We propose novel algorithms, both centralized and distributed, to cut off *user-defined* number of links so that loops can be alleviated while routing flexibility can be preserved to the largest extent. We evaluate FlexCut extensively by both testbed experiments and simulations. Results show that FlexCut improves the performance by $40\% \sim 90\%$ compared with a baseline algorithm in terms of our optimization goal. Results also show that FlexCut can improve the network performance of a sensor network by $20\% \sim 35\%$, $30\% \sim 50\%$, 25% respectively, in terms of packet delivery ratio, transmission delay, and radio duty cycle.

Index Terms—Wireless sensor network, routing control, link cutting

1 Introduction

Most sensor networks employ *distributed* and *dynamic* routing protocols. In these protocols, each node makes its own decisions on choosing its next-hop forwarder (i.e., parent) and the overall routing topology can be dynamically optimized with environmental changes. The Collection Tree Protocol (CTP) [1] in TinyOS is an instance of distributed and dynamic routing protocol with which each node regularly estimates the expected number of transmissions (ETX) [2] to the sink and dynamically selects the next-hop forwarder with the minimum ETX along the path.

The *flexibility* that each node can choose the best forwarder from a diverse candidate set could offer excellent routing performance when the network is highly dynamic since the node can easily switch its forwarders. However, it sacrifices routing *predictability* since it is possible that routing loops can be frequently formed. This is because the routing information maintained by each node cannot always be up-to-date and each node does not have a consistent view of the entire network. Once there are routing loops in the network, the performance rapidly degrades.

GreenOrbs is a real-world wireless sensor network consisting of more than 400 nodes deployed in a forest, covering an area of about 60,000m^2 [3]. Each sensor node delivers packets to the sink node with a period of 10 min. The application uses TinyOS and its network protocols, including the TinyOS LPL MAC protocol (with a sleep interval of 500 ms) for achieving low duty cycling and the CTP routing protocol [1] for multihop forwarding.

The CTP protocol uses the number of expected transmissions (ETX) [2] as a routing metric. Each CTP node maintains an estimate of its ETX of its path to the sink node. A given node's (path) ETX is the sum of its next hop plus the link ETX from this node to the next hop. When a node receives a packet to forward, it compares the ETX of the previous hop (carried in the packet) with its own. The sink node advertises an ETX of zero and ETX must always decrease when a packet traverses over a path. Hence, if the transmitter's ETX is not larger than the receiver's, the topology information is stale and there may be a routing loop. In this case, the receiver increments its loop counter. Fig. 1 shows the radio duty cycle ratio and the loop counter of a particular node in the network. Since the radio consumes a large fraction of energy on a node [4], a low radio duty cycle is usually preferred so that a node can work for a long lifetime. From Fig. 1, we can clearly see that the node's radio duty cycle becomes high when its measured loop counter is also high. This is because when routing loop happens, the energy is wasted on forwarding packets involved in the loops.

Can we increase the network predictability by controlling the network? The idea of Software-Defined Networking (SDN) can increase the network predictability as it enables centralized and direct control of the forwarding behavior. There are, however, significant challenges in directly applying SDN to sensor networks since existing solutions for

• *The authors are with the College of Computer Science, Zhejiang University, Hangzhou Shi, Zhejiang Sheng 310027, China. E-mail: {dongw, desword, gaoyi}@zju.edu.cn, zhangxy@emnets.org.*

Manuscript received 16 Feb. 2017; revised 9 July 2018; accepted 13 July 2018. Date of publication 24 July 2018; date of current version 2 May 2019. (Corresponding author: Yi Gao.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMC.2018.2859270

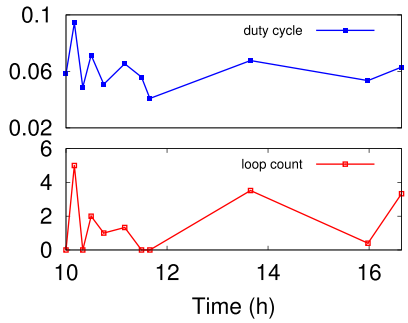


Fig. 1. The loop counter and radio duty cycle of a particular node in GreenOrbs [3].

networks with IP-based forwarding cannot well adapt to high dynamics in wireless ad hoc networks.

As a step towards solving this problem, we introduce FlexCut, a flexible approach for cutting off wireless links, which essentially limits the candidate forwarder set of each node. Unlike existing SDN solutions [5], FlexCut introduces flexible control over existing distributed and dynamic routing protocols. FlexCut can trade arbitrary amounts of routing diversity for better network performance by exposing to network operators a parameter α which quantifies the aggressiveness. In its most conservative form ($\alpha = 0$), no link is cut off so that the largest routing flexibility can be preserved. In its most aggressive form ($\alpha = 1$), FlexCut cuts off the minimum number of links so that the largest predictability can be achieved by guaranteeing that routing loops can never occur. By specifying and tuning this parameter, network operators can conveniently control the routing behaviors of the network.

We model the network as a directed graph with link weights. Each node in the graph points to its candidate forwarders. In the original graph, packets can follow directed links that form loops. The goal of FlexCut is to cut off *user-defined* number of links (determined by the aggressive parameter α) so that loops can be alleviated while routing flexibility can be preserved to the largest extent. We find that our problem is similar to a well-studied NP-complete problem called minimum feedback arc set (FAS) whose goal is to find a minimum edge set C from a directed graph $\mathcal{G}$ such that $\mathcal{G} - C$ is a DAG. Existing heuristic solutions for FAS [6], [7], [8] cannot directly be applied because (1) they do not guarantee the connectivity from every node to the sink node; (2) they do not consider link weights. We propose novel algorithms for addressing these problems. Centralized algorithms incur large communication overhead. To reduce this overhead, we further propose a distributed algorithm in which each node locally cuts off the links in a distributed manner after receiving the network-level aggressiveness parameter.

We propose an abstraction called FlexCut which includes a series of algorithms. aCut ($\alpha = 1$) and gCut ($0 \leq \alpha \leq 1$) are both centralized algorithms while dCut ($0 \leq \alpha \leq 1$) is a distributed algorithm. We implement our algorithms above the link layer and below the network layer (i.e., L2.5) so that they can potentially benefit many other routing protocols, e.g., the more recent RPL protocol [9]. We evaluate FlexCut extensively by both testbed experiments and simulations. Results show that FlexCut improves the performance by $40\% \sim 90\%$ compared with a baseline algorithm in terms of our optimization goal. Results also show that FlexCut can improve the network performance of a sensor network by $20\% \sim 35\%$, $30\% \sim 50\%$, 25% respectively, in terms of packet delivery ratio, transmission delay and radio duty cycle.

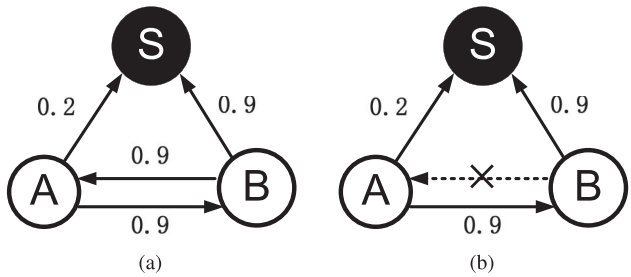


Fig. 2. A motivating example.

Our work differs from prior routing optimization works [10], [11], [12], [13] in two important ways. First, most existing work targets network developers while our work targets network operators. FlexCut can be regarded as a middleware above the routing protocol and could benefit many layer 3 routing protocols. Network operators have the choice to select different routing protocols and flexibly control the routing behavior. Second, most existing approaches are usually designed for a fixed tradeoff among different goals. For example, existing loop-free protocols [10], [11] ensure the loop free property of the network but sacrifice the routing flexibility to a large extent. Our work provides an abstraction which can trade arbitrary diversity for better network performance.

We summarize our contributions as follows:

- We propose an abstraction for controlling the routing behavior of a sensor network.
- We propose novel algorithms, both centralized and distributed, for cutting off user-defined number of wireless links.
- We implement FlexCut and evaluate its performance by both testbed experiment and simulations. Results show that FlexCut can significantly improve the network performance in terms of three primary metrics.

The rest of this paper is structured as follows. Section 2 gives a motivating example. Section 3 gives the network model and notations used in this paper. Section 4 formulates our problems. Section 5 presents the centralized algorithm. Section 6 presents the distributed algorithm. Section 7 shows the evaluation results. Section 8 introduces the related work, and finally, Section 9 concludes this paper and gives future research directions.

2 Motivating Example

To illustrate the motivation of our work, we consider the example network shown in Fig. 2a. In this figure, S denotes the sink node while A and B denote the ordinary nodes delivering data to the sink. The figures on the links denote the long-term link qualities. The quality of a link is usually measured as the packet reception ratio (PRR) over the link. Thus, a link quality of 0.9 means that there is a 0.9 probability for successfully delivering a packet over the link. Since ETX denotes the expected number of transmissions, the ETX of a link is calculated as the inverse of its PRR. The ETX of a path is calculated as the sum of ETXs of links residing on the path [2].

In most cases, B transmits data via path B–S and A transmits data via path A–B–S. A's ETX over the path A–B–S is 1/

0.9 + 1/0.9 ≈ 2.2 and B's ETX over the path B–S is 1/0.9 ≈ 1.1. However, it is possible that the link quality of BS suddenly degrades to 0.1. In this case, a temporal loop occurs since B will choose A as its parent (update its path-ETX as 3.3) and A insists on choosing B (update its path-ETX as 4.4). This is known as the classic *count-to-infinity* problem. In order to avoid the unpredictable performance caused by temporal loops, we would like to prevent the formation of loops by cutting off some links. For example, if the link BA is cut off, it is *guaranteed* that there will be no loops.

However, it is not always desired that we simply transform the network into a DAG. Cutting off links means fewer possibilities for parent selection. In Fig. 2a, B has two choices while in Fig. 2b, B has only one choice. It is possible that the original network (Fig. 2a) yields better performance than the DAG (Fig. 2b). Consider the case when the link BS suddenly degrades to 0.1 and the link AS suddenly increases to 1. For the DAG network, the data delivery performance of B also degrades. For the original network, the data delivery performance of B keeps high since there will be no loop as A chooses S directly.

This motivates us to design a general method for cutting off user-defined number of links. We would expect that network operators can use our abstractions via a configurable parameter: `Flexcut(`α`)`. The value of α determines the user-defined number of links. When $\alpha = 1$, our method will transform the original network into a DAG while maintaining the maximal forwarding diversity. When $\alpha = 0$, our method will cut off no links.

3 Network Model

We model the network as a directed graph $\mathcal{G}(V, E)$ where V is the set of nodes/vertices, and E is the set of directed links/edges, pointing from a node to its candidate forwarder/parent. The candidate forwarder set of a node u is denoted as $F(u)$. The child node set of a node u is denoted as $Ch(u)$. We denote the link that incidents to vertices u and v by uv where $v \in F(u)$. Each link has a link weight, being the long-term link quality of that link. We denote the link quality of uv as q_{uv}.

We would like to cut off a set of links in $\mathcal{G}$ to limit the formation of loops. We use C to denote the set of links to be cut off. The resultant graph is denoted as $\mathcal{G}'$.

We introduce the following notations:

- Diversity. The diversity of a node quantifies the probability that a node can successfully forward the data to one forwarder. We use the following equation to define u's diversity

$$D_F(u) = 1 - \prod_{v \in F}(1 - q_{uv}), \tag{1}$$

  where F denotes the forwarder set of u. For the example shown in Fig. 2b, B's diversity is 0.9. For the example shown in Fig. 2a, B's diversity is 0.99. We can see that increasing the forwarder set increases the diversity as a node has more choices for forwarding its data.
- Reduction ratio of diversity. The reduction radio of a node's diversity quantifies how much the diversity degrades by cutting off some links. We use the following equation to define u's reduction radio of diversity

$$R_C(u) = \frac{D_F(u) - D_{F'}(u)}{D_F(u)}, \tag{2}$$

  where C denotes the set of cutoff links, F denotes the forwarder set of u in the original graph and F' denotes the forwarder set of u in the graph with edges in C are removed.

4 Problem Formulation

We first formulate the problem in the most aggressive form, i.e., cut off links to *guarantee* no loops exist. We then formulate the problem in the general form in which the number of cutoff links are *user-defined*.

4.1 Problem in the Most Aggressive Form

In the most aggressive form, we would like to transform the original graph into a DAG so that no loops can occur. We also need to guarantee that every node has forwarding paths towards the sink. We would like to minimize the maximum diversity reduction ratio. This problem is similar to the max-min fair allocation of the bandwidth for multiple flows in a network. From the perspective of routing flexibility and performance, a small diversity reduction ratio is preferred for every node in the network. For fairness, we would like to minimize the maximum diversity reduction ratio so the parent selection flexibility will not be significantly affected for any node in the network. Note that minimizing the average diversity reduction ratio is not a desirable metric because there exist chances that a particular node has a very large diversity reduction ratio while all other nodes have small or no diversity reduction. This particular node may have very poor connectivity to the sink node.

The problem is formulated as follows:

Input The weighted directed graph $\mathcal{G}$
Output The set of cutoff links $C(\mathcal{G})$
Goal $\min \max_{u \in V} R_C(u)$
s.t. 1. $\forall u \in V, \exists$ a direct path from u to the sink in $\mathcal{G}'$.
2. $\mathcal{G}'$ is a DAG.

We denote the solution of this problem as C_1.

4.2 Problem in the General Form

In the general form, we would like to cut off user-defined number of links to limit the impact of potential routing loops. For this purpose, we introduce a user-defined aggressiveness parameter α which should be defined as a normalized factor quantifying different number of links to be pruned. We also need to guarantee that every node has forwarding paths towards the sink. The constraint is similar, i.e., to minimize the maximum diversity reduction ratio.

The problem is formulated as follows:

Input 1. The weighted directed graph $\mathcal{G}$.
2. The aggressiveness parameter α.
Output The set of cutoff links $C_\alpha(\mathcal{G})$.
Goal $\min \max_{u \in V} R_{C_\alpha}(u)$
s.t. 1. $\forall u \in V, \exists$ a direct path from u to the sink in $\mathcal{G}'$.
2. $C_\alpha \subset C_1$ and $|C_\alpha| = \alpha |C_1|$.

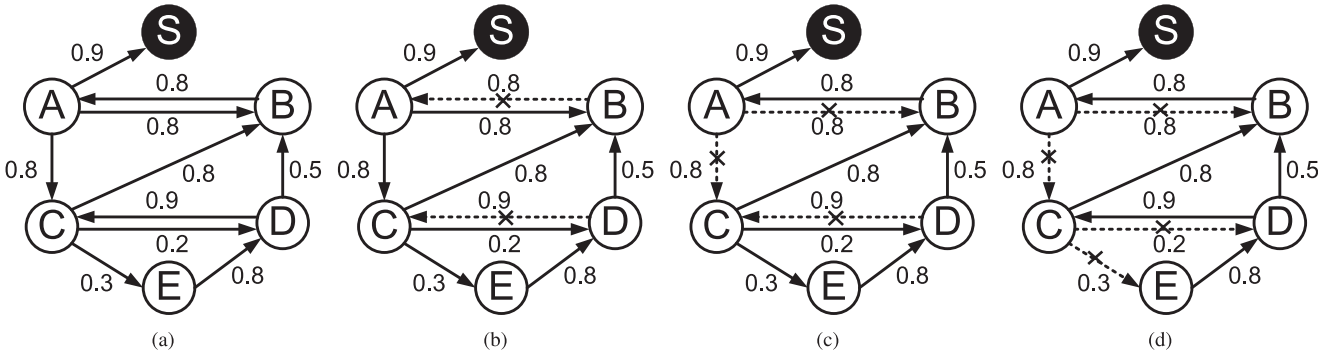


Fig. 3. Examples. (a) The original network. (b) The network after applying Eades' algorithm [6]. (c) The network after applying the enhanced Eades' algorithm (EEA). (d) The network after applying aCut.

We can see that this problem formulation is general: when $\alpha = 0$, $\mathcal{G}' = \mathcal{G}$ and the original network is unchanged; when $\alpha = 1$, the problem is similar to that in the most aggressive form, guaranteeing that no loop can occur.

Note that it is also possible to define α as a relative and normalized factor for the minimization purpose of the optimization goal. In our current work, we define α as a multiplying factor on the number of links to be pruned, i.e., $|C_\alpha| = \alpha|C_1|$, so that it can be easily manipulated in our algorithms (gCut and dCut).

At the first glance, our problem is closely related to the well-known Feedback Arc Set problem [6] whose goal is to find a *minimum* edge set C from a directed graph $\mathcal{G}$ such that $\mathcal{G} - C$ is a DAG. But in fact, these two problems are quite different. First, our problem requires that each node in the final graph $\mathcal{G} - C$ remains connected to the sink node so that its generated data packet can reach the sink node. This additional requirement changes radically the nature of the problem. It is easy to construct examples where the optimal FAS problem for a graph violates the connectivity of the network graph. Second, our optimization goal is to minimize the maximum diversity reduction ratio, instead of minimizing the number of edges to be removed. Therefore, existing algorithms for the FAS problem cannot be directly applied to our problem.

5 Centralized Algorithm

In this section, we would like to develop centralized algorithms to solve the problems formulated in the previous section. We first develop algorithms for the first problem, i.e., to find the most appropriate edge set C_1. We then develop algorithms for the second problem, i.e., to find the most appropriate subset of edges C_α from C_1. Finally, we discuss some practical issues.

5.1 Algorithm for the First Problem

We would like to borrow existing algorithms for the FAS problem to solve our problem. *Unfortunately*, existing algorithms do not address the following challenges:

- How to ensure a directed path from every node to the sink?
- How to consider link weights?

We use the example shown in Fig. 3 to illustrate these challenges. The key idea of a heuristic algorithm for the minimum FAS problem is to sort the network vertices into a sequence according to a *specific* strategy, from the highest rank (sequence head) to the lowest rank (sequence tail). The edges from a high rank to a low rank remain in the DAG, and edges from a low rank to a high rank are removed from the original graph. In this way, the resultant graph is a DAG. Formally, the following rules are applied to determine the output sequence.

- Put the nodes with zero out-degree at the tail of the sequence.
- Put the nodes with zero in-degree at the head of the sequence.
- Sort the nodes according to a metric m which is defined as the difference of in-degree and out-degree of a node, say u, i.e., $m(u) = d_{\text{in}}(u) - d_{\text{out}}(u)$. The nodes with large metric values are put near the tail of the sequence.
- Remove the node and its associated edges whenever it is put into the sequence.

After applying the above algorithm into the network shown in Fig. 3a, the output sequence will be (A, C, E, D, B, S). Therefore, the cutoff links are $C = \{BA, DC\}$. The resultant DAG is shown in Fig. 3b.

We can observe two significant problems in this algorithm. (1) It does not ensure a directed path from every node to the sink. For example, in the resultant DAG shown in Fig. 3b, nodes B, C, D, E have no directed path to the sink, implying that we can not collect data from these nodes. It is unacceptable. (2) It leads to low forwarding performance since it does not consider link weights. For example, in the resultant DAG shown in Fig. 3b, a high quality link DC (with link quality 0.9) is cut off, resulting in a high diversity reduction ratio at node D: $R_{\{BA,DC\}}(D) = \frac{0.95-0.5}{0.95} = \frac{0.45}{0.95}$. If two low quality links CD, CE were cut off while DC remains in the graph (the resultant graph is also a DAG), the impact to node C would be much smaller, i.e., $R_{\{BA,CD,CE\}}(C) = \frac{0.888-0.8}{0.888} = \frac{0.011}{0.111}$.

In order to address the above two challenges, we propose a novel algorithm. In each step, our algorithm *explicitly* considers connectivity to the sink when a node is to be added near the tail of the sequence, ensuring there exists a directed path between a node and the sink. Our algorithm adopts a new metric to consider wireless link qualities and node forwarding diversity so that the maximum diversity reduction ratio is kept small.

The new metric, *diversity preserving ratio*, is defined by the following formula:

$$m'(u) = \frac{D_{F'(u)}(u)}{D_{F(u)}(u)}, \quad (3)$$

where $F'(u)$ denotes the forwarder set of u with each element already in the tail of the sequence during the execution of the algorithm when a set of backedges have already been removed.

For example, we use Fig. 2a to show the case during the execution of the algorithm. Both nodes A and B have two outgoing edges in the original graph. In the current step, both A and B have outgoing edges pointing to S which is a node already in the tail of the sequence. If A was added into the tail of the sequence in this step, edge AS will remain in the final DAG and the other outgoing edge AB will not exist in the final DAG. In this step, $m'(A) = \frac{0.2}{0.92}$ and $m'(B) = \frac{0.9}{0.99}$ according to the definition of the metric defined in Eq. (3). The algorithm tends to put the node with large metric value near the tail of the sequence, i.e., B in this case. The reason is explained as follows:

1) If we put B into the tail of the sequence in this step, the diversity reduction ratio of B is $\frac{0.09}{0.99}$ and the diversity reduction ratio of A is at most $\frac{0.72}{0.92}$. Hence the maximum diversity reduction ratio among A and B is *at most* $\frac{0.72}{0.92}$.
2) If we put A into the tail of the sequence in this step, the diversity reduction ratio of A is $\frac{0.72}{0.92}$ and the diversity reduction ratio of B is at most $\frac{0.09}{0.99}$. Hence the maximum diversity reduction radio among A and B is *exactly* $\frac{0.72}{0.92}$.

Since our goal is to minimize the maximum diversity reduction ratio in the network, the first strategy is preferred since it yields better performance than the second strategy.

Algorithm 1 shows the pseudocode of our algorithm.

Algorithm 1. Aggressive Cutoff Algorithm (aCut)

Input: Weighted directed graph $\mathcal{G}(V, E)$
Output: Set of cut edges C
1: $C \leftarrow \phi$;
2: $s_1 \leftarrow \phi$; $s_2 \leftarrow$ sinkNode;
3: $P \leftarrow$ set of nodes having outgoing edges to sinkNode;
4: **while** $|s_1| + |s_2| < |V|$ **do**
5: **for** u: $u \in V - s_1 - s_2$ && $Ch(u) \subseteq (s_1 \bigcup s_2)$ **do**
6: $s_1 \leftarrow s_1 u$;
7: $u \leftarrow argmax_{u \in P} m'(u)$ where $m'(u)$ is given in Eq. (3).
8: $s_2 \leftarrow u s_2$;
9: $P \leftarrow P - \{u\}$;
10: **for** u: $u \in V - s_1 - s_2$ && $(\exists v \in s_2 : uv \in E)$ **do**
11: $P \leftarrow P \cup \{u\}$;
12: $s \leftarrow s_1 s_2$;
13: **for** each $uv \in E$ **do**
14: **if** v is on the left of u in s **then**
15: $C \leftarrow C \cup \{uv\}$;
16: return C.

The input of the algorithm is a weighted directed graph $\mathcal{G}$ which represents the network topology. The output of the algorithm is the set of cutoff links C whose removal transforms the graph into a DAG.

- *Lines* $1 \sim 3$: The algorithm initializes 4 variables. C is the set of cutoff links and it is initialized to empty. s_1 and s_2 are two ordered set of nodes: s_1 stores the processed nodes in the sequence head while s_2 stores the processed nodes in the sequence tail. The algorithm first puts the sinkNode into the sequence tail. Set P is used to store the set of nodes having directed paths to the sinkNode. The algorithm tries to add nodes from P to the front of s_2 in order to ensure that every node has a directed path to sinkNode. P is initialized to be the nodes having directed edges to the sinkNode.
- *Lines* $4 \sim 11$: The algorithm tries to add all the nodes in the network to s_1 or s_2. After the loop, the concatenation of s_1 and s_2 forms a new node sequence, implying the removed links (i.e., backedges from right to left).
  - *Lines* $5 \sim 6$: The algorithm selects all unprocessed nodes whose child nodes are already in the sequence head or tail. The algorithm then adds those nodes at the tail of s_1. According to Eq. (3), a node's position in the sequence can only affect its child nodes. For any node u whose child nodes are already in s_1 or s_2, its position will not affect the metric values of *unprocessed* nodes which cannot be child nodes of u. Therefore, it's better to put u near the head of the sequence, so its own metric value can be maximized.
  - *Lines* $7 \sim 11$: The algorithm selects from P the node which has the maximum metric value and add it to the front of s_2. Note that every node in P has directed paths to sinkNode. The algorithm also updates P by removing the added node and adding the added node's children nodes.

  Lines $12 \sim 15$: A new sequence s is formed by concatenating s_1 and s_2. The cutoff edges are those backedges defined by s.

We use the example shown in Fig. 3a to illustrate the working details of Algorithm 1. Initially, $C = \phi$, $s_1 = \phi$, $s_2 = (S)$, and $P = \{A\}$. Then we start the while loop.

- First iteration. There's no node satisfying condition specified in line 5. The algorithm adds A to the front of s_2 since A is the only node in P. P is updated to $\{B\}$ since B has outgoing edges to A. After this iteration, $s_1 = \phi$, $s_2 = (A, S)$.
- Second iteration. There's no node satisfying condition specified in line 5. The algorithm adds B to the front of s_2 since B is the only node in P. P is updated to $\{C, D\}$. After this iteration, $s_1 = \phi$, $s_2 = (B, A, S)$.
- Third iteration. There's no node satisfying condition specified in line 5. Now there are two candidates, C and D, to be processed. According to Eq. (3), $m'(C) = \frac{0.8}{0.888}$ and $m'(D) = \frac{0.5}{0.95}$. Hence, the algorithm adds C to the front of s_2. P is updated to $\{D\}$. After this iteration, $s_1 = \phi$, $s_2 = (C, B, A, S)$.
- Fourth iteration. The algorithm adds E to the tail of s_1 since its only child node C has already been in s_2. The algorithm continues to add D to the tail of s_1 since D's child nodes, C and E, are already in s_1 or s_2. P is updated to ϕ. After this iteration, $s_1 = (E, D)$ and $s_2 = (C, B, A, S)$. The algorithm terminates because s_1 and s_2 include all nodes in the network.

The final sequence is (E, D, C, B, A, S) and $C = \{AC, AB, CD, CE\}$. The resultant graph is shown in Fig. 3d. The maximum diversity reduction ratio is $R_{\{AC,AB,CD,CE\}}(C) = \frac{0.088}{0.888}$.

In order to see the benefits of our algorithm, we also implement an enhanced Eades' algorithm (EEA) which uses the metric of $m(u) = d_{\text{in}}(u) - d_{\text{out}}(u)$ while ensuring directed paths between every node to the sink. The resultant graph is shown in Fig. 3c. The maximum diversity reduction ratio is $R_{\{AB,AC,DC\}}(D) = \frac{0.45}{0.95}$. We can see that our algorithm results in significantly better performance than EEA.

It is important to point out that *the solution of aCut ensures that (1) each node remains connected to the sink node, (2) the final network is loop free.*

Proof. *Loop-free property.* The final network must be loop free since our algorithm generates a sequence in which all backedges are removed from the original graph.

Connectivity Property. In our algorithm, each node can be added to the final sequence via two ways: (1) it is added to s_2, (2) it is added to s_1.

First, if the node is added to s_2, it must have a directed path to the sink since it is selected from the set P (lines 7 and 8) which have directed paths to the sink by definition.

Second, if the node, say x, is added to s_1, it must be that the original graph $\mathcal{G}$ contains no backedges from x. Assume there is a backedge $x \rightarrow x_l$ in the original graph, i.e., $x \rightarrow x_l \in E$ and x_l is added to s_1 before x. When adding x_l to s_1, it must follows that $Ch(x_l) \in s_1 \cup s_2$ (line 5), i.e., x must be already in s_1 before x_l, contradicting with the assumption that there is a backedge $x \rightarrow x_l$ in graph $\mathcal{G}$. Since the original graph is connected, i.e., each node has a directed path to the sink and our algorithm does not remove any outedges for any node x in s_1 (i.e., these outedges do not appear as backedges in the final sequence), x remains connected to the sink. □

Time Complexity of aCut. The while loop processes all n nodes (line 4). For each node, the argmax operation may require $O(n)$ time complexity at worst (line 7). The overall time complexity is thus $O(n^2)$.

Algorithm 2. Generalized Cutoff Algorithm (gCut)

Input: Weighted directed graph $\mathcal{G}(V, E)$, parameter $\alpha \in [0, 1]$
Output: Set of cut edges C_α
1: $C_\alpha \leftarrow \text{aCut}(\mathcal{G})$
2: $NumLinks \leftarrow \alpha|C_\alpha|$
3: **while** $|C_\alpha| > NumLinks$ **do**
4: $P \leftarrow$ set of nodes whose outgoing links exist in C_α
5: $u \leftarrow arg\,max_{u \in P} R_{C_\alpha}(u)$
6: $v \leftarrow arg\,max_{uv \in C_\alpha} q_{uv}$
7: $C_\alpha \leftarrow C_\alpha - \{uv\}$;
8: return C_α

5.2 Algorithm for the Second Problem

Based on the algorithm developed in the previous section, we would like to develop an algorithm for the general problem formulated in Section 4. For this problem, there is an aggressive parameter α, which is used to limit the number of cutoff links. Our algorithm first finds out the set of cutoff links C_1 whose removal transforms the network into a DAG. Then the algorithm greedily removes links from C_1. Those links will be reserved in the resultant graph. In each step, the algorithm selects the link which can most effectively decrease the value of our optimization function, i.e., the maximum diversity reduction ratio.

Algorithm 2 shows the pseudocode of our algorithm.

The input of our algorithm is a weighted directed graph $\mathcal{G}$ and an aggressive parameter in the range of [0, 1]. The output of our algorithm is a set of cutoff links C_α. The number of cutoff links depends on the aggressiveness parameter: when $\alpha = 0$, $C_\alpha = \phi$; when $\alpha = 1$, $C_\alpha = C_1$.

- *Lines* $1 \sim 2$*:* The algorithm initializes C_α to C_1 which is the output of Algorithm 1. NumLinks is used to record the number of links which will remain in C_α.
- *Lines* $3 \sim 7$*:* The algorithm keeps removing links from C_α until the number of links in C_α equals to NumLinks. The algorithm finds out all nodes whose diversity decreases due to the removal of C_α, i.e., nodes whose outgoing edges exist in C_α. The results are stored in the set P. From set P, the algorithm finds out the node u whose diversity reduction ratio is the maximum. Node u is the node with the maximum diversity reduction ratio in the network. Since our goal is to minimize the maximum diversity reduction ratio, we should reduce u's diversity reduction ratio to the largest extent. That means the link uv having the best link quality in C_α should be reserved in the resultant graph. In other words, the link uv should be removed from C_α.

We use the example shown in Fig. 3d to illustrate the working details of Algorithm 2. Suppose $\alpha = 0.5$. Initially, $C_{0.5} = \{AB, AC, CD, CE\}$, NumLinks = 2.

- First iteration. $P = \{A, C\}$. $R_{C_{0.5}}(A) = \frac{0.096}{0.996}$ and $R_{C_{0.5}}(C) = \frac{0.088}{0.888} > R_{C_{0.5}}(A)$. We delete CE which is the best outgoing edge of node C in $C_{0.5}$.
- Second iteration. $R_{C_{0.5}}(C) = \frac{0.008}{0.888} < R_{C_{0.5}}(A)$. We delete edge AB from $C_{0.5}$.

As the result, $C_{0.5} = \{AC, CD\}$ as opposed to $C_1 = \{AB, AC, CD, CE\}$.

Time Complexity of gCut. Invocation to aCut causes a time complexity of $O(n^2)$. The while loop process $(1 - \alpha)|C_\alpha|$ links. In processing each link, the worst time complexity is $O(C_\alpha)$. Therefore, the time complexity of gCut is $O(n^2 + |C_\alpha|^2)$. Given the number of edges e and nodes n in the original network, the maximum value of C_α would be $e - (n - 1)$ since at least $(n - 1)$ edges should remain in the network to ensure connectivity. The worst time complexity of gCut becomes $O(n^2 + (e - n + 1)^2) \approx O(e^2)$ when e is much larger than n for dense sensor networks.

5.3 Practical Issues and Message Overhead

Practical Issues. The centralized algorithm runs at the PC backend. There are several practical issues.

How to collect the network topology with link weights (to the sink node and then to the PC backend)? Each node can maintain its neighborhood information. Such neighborhood information (e.g., its candidate forwarders and the corresponding long-term link qualities) can be transmitted to the sink node via multihop communication, using a data collection protocol (e.g., CTP [1]).

How to inform each individual node to cut off the links? The sink should find the path to an individual node to inform it to cut off the links originated from this node. For example, the network could employ the TeleAdjusting protocol [14] in which a packet used for remote control is forwarded along a cost-optimal path.

How to deal with network dynamics, such as node additions and deletions? When a node is added to the network, the node can report its neighborhood information to the sink. When sink finds that the network topology (i.e., the input) changes, it recomputes the cutoff links. The sink then informs each individual node to take the new actions, e.g., remove new links or recover old removed links. When a node is removed from the network, its neighbor can detect this phenomenon and inform the sink node which can again recompute the new result.

Message Overhead. Assume the network has n nodes. The average degree of a node is d. The subtree routed at node i has a size of $T(i)$.

The centralized algorithm needs to collect network topology information (including the link qualities) periodically. The size of topology information of each node is proportional to the degree, denoted as $O(d)$. Assume the period is τ_b. For node i, it needs to receive and transmit topology information from $T(i)$ nodes. Therefore, the collection overhead at node i is $T(i) * O(d) * T/\tau_b$ for a total time period of T.

After running the algorithm at the sink node, it disseminates the control information to remove individual links. Assume we need to remove m links which reside on the subtree rooted at i. The control overhead is $O(m)$.

Hence, the total overhead at node i is $T(i) * O(d) * T/\tau_b + O(m)$ for a time period of T.

6 Distributed Algorithm

As we have mentioned in the previous section, the centralized algorithm has relatively large communication overhead since it runs on the PC backend and requires collecting messages for building the whole network graph as well as transferring messages for controlling the routing behaviors of the individual node.

To address this limitation, we propose a distributed algorithm with which each node decides to cut off its own outgoing links based on its local information after receiving a network-wide aggressive parameter α.

The key strategy of the centralized algorithm is to sort the nodes according to its metric value into a sequence. The cutoff links are those backedges defined by the sequence, i.e., edges from right to left. In order to cut off links at an individual node, the node only needs to know its relative position to its candidate forwarders in the final sequence. To determine such relative positions, we need to address the following challenging problems.

When to compute the metric value? We let the node compute its metric value when one of its forwarders has been added to the tail of the final sequence. A node receives notifications about its neighbors' status (i.e., whether the neighbor enters the sequence tail) via message exchanges. In this way, a node can be added to the sequence tail only when at least one of its forwarders exists in the sequence tail, ensuring a directed path exists towards the sink.

How to compute the metric value locally? We let a node maintain its neighborhood information, e.g., its candidate forwarders, the link qualities to these forwarders. A node also receives its forwarder's status information, i.e., whether the forwarder has already entered the sequence tail. All links to the forwarders in the sequence tail are reserved in the resultant topology while links to other forwarders are temporally unavailable in the resultant topology. A node uses such information to compute and update its metric value. The metric value converges to the correct value only when the node possesses the maximum metric value and enters the sequence tail in the current step.

How to elect the node with the maximum metric value? A node will periodically broadcast its metric value. Other node receiving this metric value judges whether its metric value is larger than the metric value overheard. If yes, the node will remain in the contention phase, attempting to overhear more metric values. If a sufficiently long time period, say τ_c, has passed and the node does not hear any message having a larger metric value, the node is elected out and it is added to the sequence tail. Otherwise, the node helps propagating the overheard larger metric value so that more nodes can learn this fact. Since each node maintains the largest metric value it has learnt, this value must be invalidated once the corresponding node has been added to the sequence tail. It is possible that two or more nodes elect themselves out in the same step because they did not overhear other larger metric value. The distributed algorithm works correctly in this case since only forward edges remain in the network and there are no chances that there are routing loops between those nodes. However, there will be performance degradations since links between those nodes may be unnecessarily removed. In our implementation, we can optimize the setting of broadcast timer so that the chances of control message losses are small.

Algorithm 3 shows the pseudocode of our algorithm. The network-wide parameter α is disseminated into the network. After receiving this parameter, each node runs this algorithm locally and blacklists the requested number of links (or forwarders).

Each node has 3 states: IDLE, PROCESSING, and FINISH. IDLE is the initial state. When the node starts to compute its metric value, it enters the PROCESSING state. Once the node is added to the sequence tail, it enters the FINISH state.

There are 4 events to handle. (1) A control message is received. (2) The broadcast timer is fired. The broadcast timer is used to control the transmissions of the control messages. (3) The contend timer is fired. The contend timer is used for node election. If a node does not hear a larger metric value during the contend time period, it thinks it has the largest metric value in the network. (4) The invalidate timer is fired. The invalidate timer is used to invalidate the metric value of the elected node so that the next node can be elected and added to the sequence tail.

- *Lines 11–34:* The node handles the control message reception event. Lines 13–17: the node computes its metric value and enters the contention phase. Lines 18–20: the node invalidates its local information. Lines 22–28: the node helps propagate the invalidate

information. Lines 29–34: the node updates the overheard metric value. If it is larger than its own metric, it stops the contention.

- *Lines 35–47:* The node handles the broadcast timer fired event. Lines 36–38: The node broadcasts the information when it finishes (i.e., enters the sequence tail). Lines 39–42: The node broadcasts its own metric value. Lines 43–47: The node helps propagate the maximum metric value it overheard.
- *Lines 48–56:* The node handles the contention timer fired event. The node enters the FINISH state and cuts off the requested number of backedges pointing from the current node to forwarders in the PROCESSING state.
- *Lines 57–62:* The node handles the invalidate timer fired event.

Algorithm 3. Distributed Cutoff Algorithm (dCut)

```
1: enum {IDLE, PROCESSING, FINISH};
2: Array finishBitmap[N];
3: broadcastTimer = Timer(PERIODIC, τ_b);
4: contendTimer = Timer(ONESHOT, τ_c);
5: invalidateTimer = Timer(ONESHOT, τ_i);
6: state ← IDLE; metric ← 0;
7: maxMetric ← 0, maxMetricOwner ← −1;
8: finishBitmap ← 0;
9: broadcastTimer.start();
10: invalidateTimer.start();
11: Receive msg
12:   if msg.state == FINISH then
13:     if msg.nodeID is in my candidate forwarder set then
14:       Compute metric; state ← PROCESSING
15:     if state ≠ IDLE && metric ≥ maxMetric then
16:       contendTimer.start();
17:     finishBitmap[msg.nodeId] ← 1;
18:     if maxMetricOwner == msg.nodeID then
19:       maxMetric ← 0;
20:       maxMetricOwner ← −1;
21:   else if msg.state == PROCESSING then
22:     if finishBitmap[msg.nodeId] == 1 then
23:       newMsg.state = FINISH;
24:       newMsg.nodeID = msg.nodeID;
25:       broadcast(newMsg);
26:     else if msg.nodeId == maxMetricOwner then
27:       maxMetric = msg.metric;
28:       invalidateTimer.restart();
29:     else if msg.metric > maxMetric then
30:       maxMetric = msg.metric;
31:       maxMetricOwner = msg.nodeID;
32:       invalidateTimer.restart();
33:     if contendTimer.isRunning() && maxMetric >
    metric then
34:       contendTimer.stop();
35: broadcastTimer fired:
36:   if state == FINISH then
37:     newMsg.state = FINISH;
38:     newMsg.nodeID = MY_NODE_ID;
39:   else if metric > maxMetric then
40:     newMsg.state = PROCESSING;
41:     newMsg.nodeID = MY_NODE_ID;
42:     newMsg.metric = metric;
43:   else
44:     newMsg.state = PROCESSING;
45:     newMsg.nodeID = maxMetricOwner;
46:     newMsg.metric = maxMetric;
47:   broadcast(newMsg);
48: contendTimer fired:
49:   if state == PROCESSING then
50:     state = FINISH;
51:     finishBitmap[MY_NODE_ID] = 1;
52:     newMsg.state = FINISH;
53:     newMsg.nodeID = MY_NODE_ID;
54:     broadcast(newMsg);
55:     C ← forwarders whose state is PROCESSING;
56:     blacklist α|C| forwarders with the worst link qualities;
57: invalidateTimer fired:
58:   if state == PROCESSING then
59:     maxMetric = 0;
60:     maxMetricOwner = -1;
61:     if metric ≠ 0 then
62:       contendTimer.restart();
```

There are three parameters τ_b, τ_c, τ_i in the distributed algorithm. τ_b is the time period for broadcasting control messages. τ_c is the time period for electing the node with the maximum metric value. If τ_c were large enough for getting notification from any node in the network, the order of the final sequence will be the same as the centralized algorithm. However, a large τ_c value can cause a long execution time for cutting off the links. In practice, we set τ_c multiple times of τ_b. τ_i is the time period for invalidating the metric value of an already elected node. In practice, we set τ_i several times of τ_b. Without explicitly specified, we set $\tau_b = 1$ min, $\tau_c = 10$ min, and $\tau_i = 3$ min in the default implementation of the distributed algorithm.

Message Overhead of dCut. Each node transmits a small message (newMsg) every τ_b and τ_c. Moreover, it needs to help propagate information from other nodes. Assume the contention timer is set so that the message can propagate l hops far away. The total overhead for transmitting and propagating the metric value is $T/\tau_b + T/\tau_c + d^l$ for a time period of T.

After receiving the messages containing the metric value, each node runs the algorithm locally to remove links. The sink node only needs to disseminate a global α value to each node. Therefore, the control overhead of a node is $O(1)$.

Hence, the total overhead for any node is $T/\tau_b + T/\tau_c + d^l + O(1)$ for a time period of T.

Comparing the message overhead of the centralized algorithm and the distributed algorithm, we have the following observations: (1) The distributed algorithm has a smaller control overhead. (2) The distributed algorithm has better load balance than the centralized algorithm. For centralized algorithms, nodes near the sink may have a large number of nodes in its subtree and hence have high traffic for topology collection.

7 Evaluation

We implement FlexCut based on the TinyOS 2.1.2/TelosB platform. There are two versions: centralized and distributed. We evaluate our algorithms in terms of four metrics:

- Maximum diversity reduction ratio (MDRR): it is the optimization goal of our algorithm and is used to evaluate the theoretical performance of various algorithms.

Fig. 4. Indoor testbed consisting of 80 TelosB nodes.

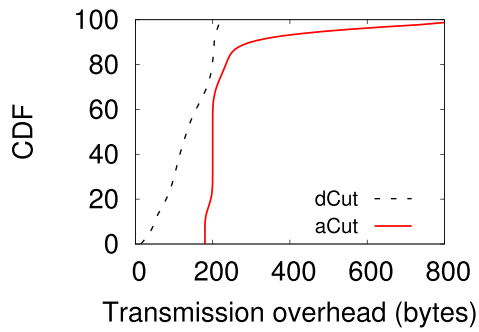


Fig. 5. Transmission overheads of centralized and distributed algorithms.

- Packet delivery ratio (PDR): it is the successful packet reception ratio from a given node to the sink node. It is one of the most important network metrics.
- Packet transmission delay: it is the total transmission delay from a given node to the sink node. It is one of the most important network metrics, especially for real-time applications.
- Radio duty cycle ratio: it is the percentage of radio on time for a given sensor node. Radio duty cycle ratio represents the energy consumption on a sensor node since typical sensor networks employ low duty-cycling to save energy and radio communication consumes the most energy on a sensor node.

In Section 7.1, we conduct experiment in one indoor testbed consisting of 80 TelosB nodes. In Section 7.2, we perform a trace-driven study on GreenOrbs—a real sensor network deployment consisting of over 400 sensor nodes. In Section 7.3, we conduct comparative study in different network configurations. In Section 7.4, we experimentally study the impact of α on the performance of our algorithms. All the above experiments evaluate FlexCut with the CTP protocol. Section 7.5 performs an initial exploration of the RPL protocol.

7.1 Testbed Experiment

We use an indoor experiment consisting of 80 TelosB nodes (see Fig. 4) for the experiment.

The inter-node spacing is 0.6 m and the power level of the radios is configured to 1 in order to simulate multihop behaviors. We use the CTP protocol for data collection: each node generates data packets every 30 seconds.

Fig. 5 shows the transmission overhead of the centralized algorithm and the distributed algorithm (with $\alpha = 1$). We can see that the transmission overhead of the centralized algorithm is much larger than the distributed algorithm due to extra overhead in the topology collection process and the remote control process. In the distributed algorithm, we also observe that (1) the control traffic load is distributed *evenly* in the network (unlike centralized algorithms with which the nodes near the sink *frequently* participate in forwarding the control packets). (2) the network nodes can *concurrently* cut off links at different locations, resulting in smaller response time for network control.

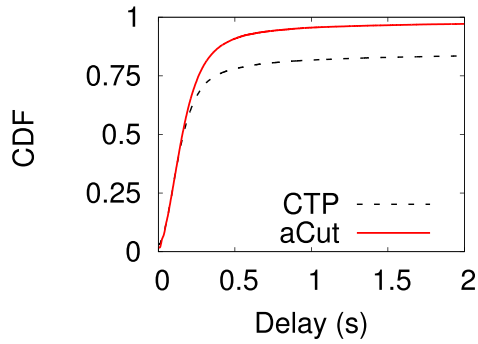


Fig. 6. CDF of transmission delay in the testbed experiment.

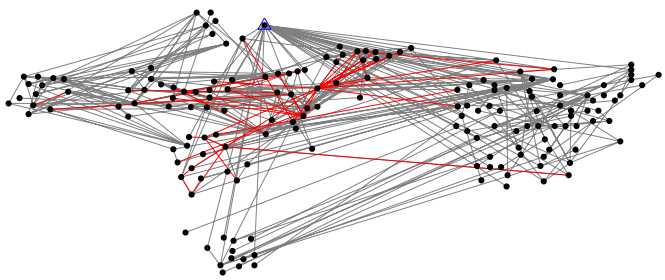


Fig. 7. Topology of GreenOrbs. The cutoff links are shown in red.

For the testbed experiment setting, it is difficult to show the improvement of our algorithm due to relatively small network scale and low routing dynamics. We artificially inject loops into the network by periodically increasing the path-ETX value of 20 nodes near the sink, increasing the looping probability of packets from the child nodes of those nodes. Fig. 6 shows the CDF of the transmission delay in the testbed experiment for the CTP protocol and aCut ($\alpha = 1$). We can see that aCut results in much lower transmission delays. Note that both CTP and aCut achieves a high packet delivery ratio due to the high retransmission threshold configured in the default CTP (i.e., 30).

7.2 Trace-Driven Study

We also perform trace-driven study on a real-world sensor network system—GreenOrbs. The GreenOrbs topology is extracted from the "parent" field of the data packets from each node. Each node also sends a special kind of packets, containing neighborhood information from each node (e.g., neighbor node ID, link ETX estimate to each neighbor, etc.). Please refer to [3] for the detailed descriptions about the collected packet trace. Fig. 7 shows the topology of GreenOrbs, with the solid line representing the routing links with quality better than a threshold (the quality is calculated from link ETX). After applying our centralized algorithm with $\alpha = 1$ to the network topology, we can obtain a set of cutoff

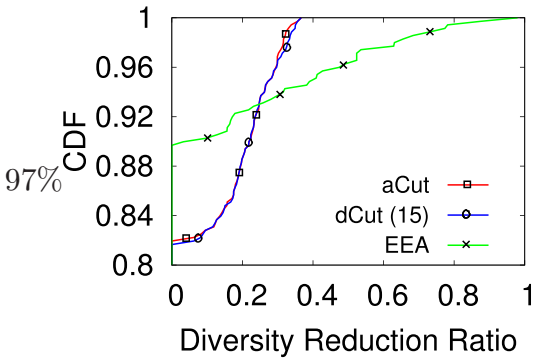


Fig. 8. CDF of diversity reduction ratio in GreenOrbs.

links, which are shown as red lines in Fig. 7. We can see that our algorithms "reshape" the network topology so that the possibilities of routing loops can be limited.

Fig. 8 shows the CDF of the diversity reduction ratio for all network nodes. For aCut, all nodes have diversity reduction ratios less than 0.4, implying that no node sacrifices too much diversity. For dCut ($\alpha = 1$), when $\tau_c = 15$ min, the performance is the same as aCut. We also consider the Enhanced Eades' Algorithm described in Section 5.1. The EEA algorithm tries to remove the minimum number of edges from the topology, resulting in the largest number of nodes with zero diversity reduction. However, it is possible that a very good link is removed by EEA since it does not consider link qualities. Therefore, we can find that the maximum diversity reduction ratio can reach as much as 0.9 for EEA.

7.3 Comparative Study

We perform comparative study on the following algorithms.

- Enhanced Eades' Algorithm. EEA ensures that every node has a directed path towards the sink. Note that EEA does not consider link weights.
- aCut. Our centralized algorithm with $\alpha = 1$ (Algorithm 1).
- gCut. Our centralized algorithm with $0 \leq \alpha \leq 1$ (Algorithm 2).
- dCut. Our distributed algorithm.
- Bloom-filter-based forwarding (BFF). BFF is a simple approach to attach each packet a bloom filter for recording the already traversed nodes. At each forwarder node, the routing decision should try to avoid selecting the next-hop nodes in the bloom filter. If the Bloom filter contains all the possible forwarders, the node randomly selects a forwarder. The Bloom filter is set to 64 bits.

We evaluate different algorithms in different topologies:

- 36-node. We generate this topology by the topology generation tool in the TinyOS distribution. We deploy 36 nodes in a 20m $\times$ 20m area, which is divided into 36 squares. Each node is randomly deployed in one square. We obtain the link weight by mapping the SNR to its corresponding packet reception ratio.
- 196-node. We also generate this topology by the topology generation tool in the TinyOS distribution. We deploy 196 nodes in a 35m $\times$ 35m area, which is divided into 196 squares. Each node is randomly deployed in one square.
- 400-node. We generate this topology by the topology generation tool in the TinyOS distribution. We deploy 400 nodes in a 40m $\times$ 40m area, which is divided into 400 squares. Each node is randomly deployed in one square.
- GreenOrbs. We extract the topology from a real sensor network deployment with over 400 sensor nodes.

Comparison in Terms of MDRR. Fig. 9 shows the performance of different algorithms (with $\alpha = 1$) in terms of the maximum diversity reduction ratio in four different network topologies. For the 36 node topology, we also show the optimal results which are obtained by enumerating all possibilities. And for the other larger topologies, the calculation overhead of enumerating algorithm is unacceptable. From Fig. 9a, we can see that the performance of our centralized algorithm is very similar to the optimal result. In all the network topologies, our centralized algorithm achieves the best performance. For the distributed algorithm, the parameter τ_c has a great impact. For a small value of $\tau_c = 5$ min (dCut(5)), it is possible that a node mistakenly believes it has the maximum metric value and is elected. Therefore, there is a large performance gap from the centralized algorithm. For a relatively large value of $\tau_c = 15$ min (dCut(15)), the performance of the distributed algorithm is close to that of the centralized algorithm.

Comparison in Terms of PDR and Delay. Fig. 10 shows the comparison results in terms of packet delivery ratio. We compare our distributed algorithm with EEA and BFF. We can see that (1) the EEA algorithm always results in a low PDR. This is because EEA does not consider link qualities. (2) BFF's PDR decreases in larger networks. It is possible that BFF may not be able to find a loop-free path towards the sink. For the example shown in Fig. 3a, packets from D will follow the path DCBAS under the normal condition. However, when the link CB severely degrades (e.g., disconnected), D may not instantly recognize this fact. D continues

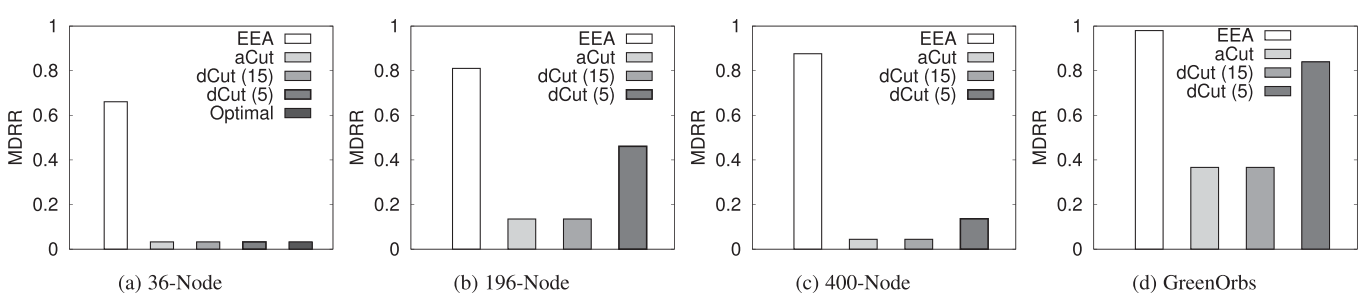


Fig. 9. Performance in terms of our optimization goal.

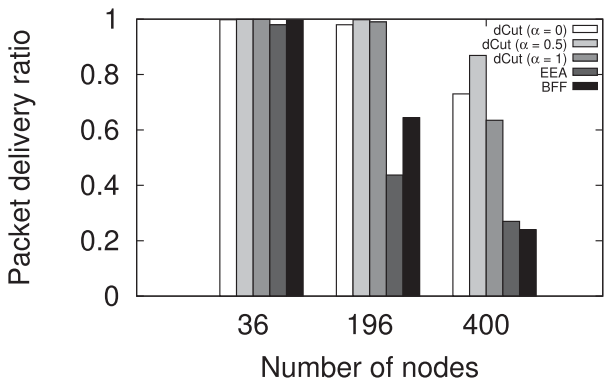


Fig. 10. Comparison results in terms of PDR.

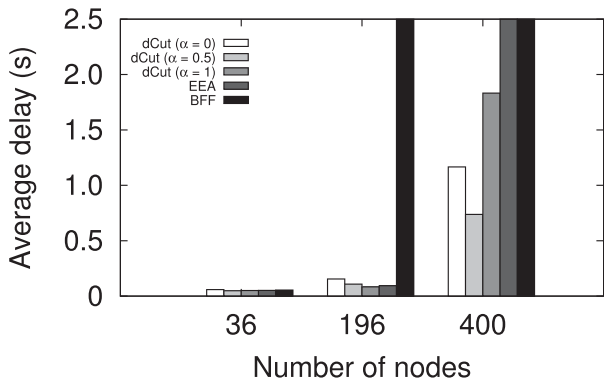


Fig. 11. Comparison results in terms of delay.

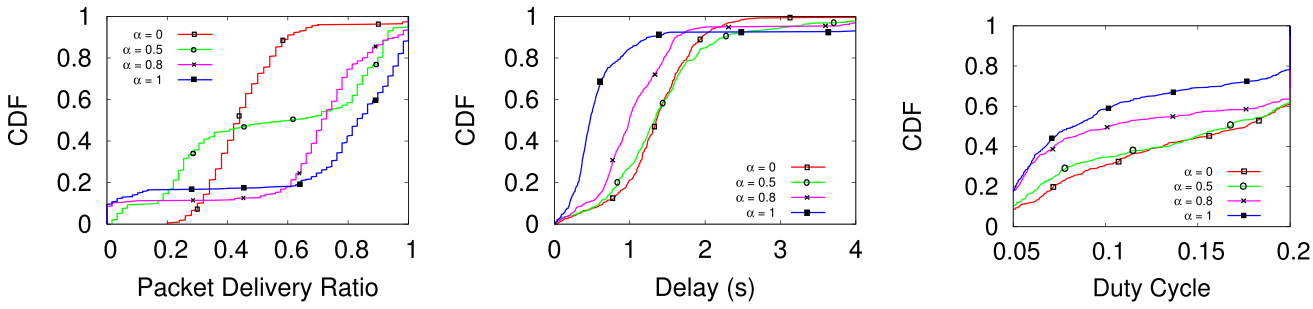
to forward packets to C while C forwards the packet to E since C recognizes that the link CB is already disconnected. At forwarder E, the only feasible next-hop node is D. However, the selection of D will inevitably cause a routing loop of DCED. (3) PDRs of FlexCut with different parameters are different. We will carefully study the impact of the α parameter in Section 7.4.

Fig. 11 shows the comparison results in terms of transmission delay. We observe (1) BFF results in a high transmission delay, especially in large networks. This is partly because the randomly chosen forwarders result in poor performance when Bloom filter is limited in size and already contains all possible forwarders. (2) Delays of FlexCut with different parameters are different. In particular, delays of EEA and FlexCut with $\alpha = 1$ increase in large networks (e.g., 400-node network).

7.4 Impact of α

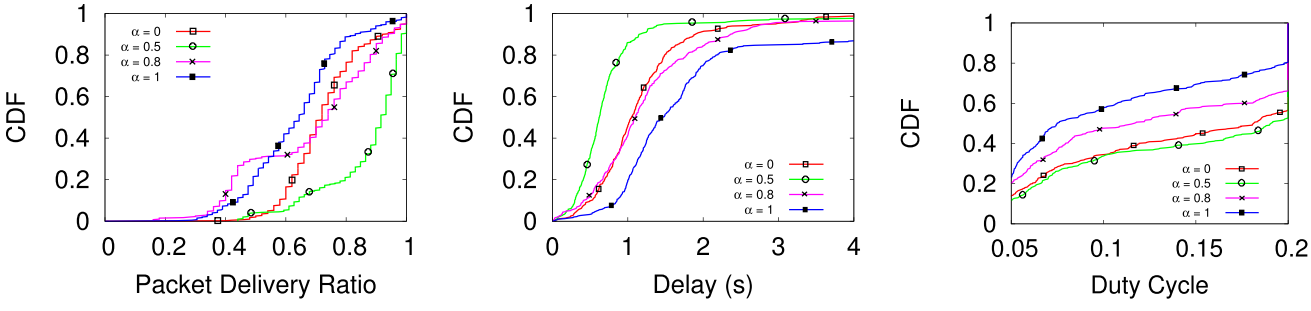
We examine the impact of the α parameter on three primary sensor network metrics, packet delivery ratio, transmission delay, and radio duty cycle. We perform simulation studies for 400-node network running the CTP protocol. We combine our distributed algorithm with CTP protocol. Each node generates a data packet every 5 min. In CTP, there is a threshold for switching parents for a given node. A node only switches to a new parent if the difference of the current parent's ETX and the new parent's ETX is larger than the threshold. Therefore, with a small threshold, we will see more parent changes and the network is more dynamic. We vary this threshold to generate network settings with different network dynamics.

Figs. 12 and 13 show the network performance of our algorithms with different α values in low dynamic network and high dynamic network. We find that our algorithm is effective in improving the network performance of a sensor network: our algorithm with the most appropriate α value improves the PDR performance by $20\% \sim 35\%$ in average, reduces the transmission delay by $30\% \sim 50\%$ in average, and reduces the radio duty cycle by 25 percent in average.



(a) PDR (b) Delay (c) Duty cycle

Fig. 12. Network performance of dCut for the 400-node topology with low dynamics (ETX threshold = 2).



(a) PDR (b) Delay (c) Duty cycle

Fig. 13. Network performance of dCut for the 400-node topology with high dynamics (ETX threshold = 1).

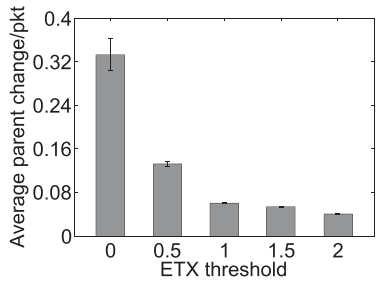


Fig. 14. Average parent change/packet under different ETX threshold.

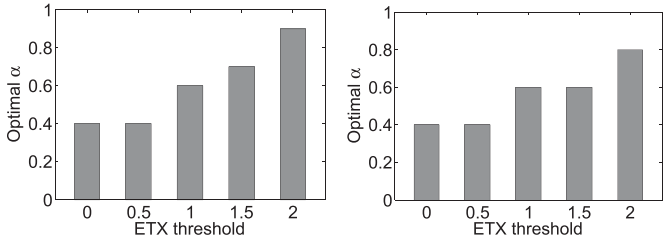


Fig. 15. Optimal α in terms of PDR (left) and delay (right).

TABLE 1
Optimal α in Terms of PDR (Left) and Delay (Right)

Parent Change/pkt	Optimal α	PDR
0.33	0.4	0.99
0.13	0.4	0.99
0.06	0.6	0.98
0.05	0.7	0.98
0.04	0.9	0.82

Parent Change/pkt	Optimal α	Delay(s)
0.33	0.4	1.41
0.13	0.4	1.12
0.06	0.6	0.92
0.05	0.6	0.90
0.04	0.8	0.70

TABLE 2
Network Performance Comparison Averaged over 400 Nodes between RPL and CTP

	Contiki-RPL		CTP	
Packet Interval	2 min	5 min	2 min	5 min
PDR	0.99	0.98	0.99	0.99
Parent Change	0.133	0.142	0.121	0.135
Loop Count	15.3	0.54	18.4	0.318

To help a network operator to configure α for real WSN, we conduct further experiments by adjusting α and the ETX threshold in a more fine-grained manner. Specifically, the ETX threshold varies from 0 to 2 with a step of 0.5, and the α value varies from 0 to 1 with a step of 0.1. Each experiment lasts for two hours. Fig. 14 shows how the average number of parent change per packet varies with different settings of the threshold. We see that different settings of the threshold indeed impact the number of parent change which can be measured and observed by the network operator. Fig. 15 shows the optimal α value in terms of the PDR and delay performance in different settings. We observe that (1) a high α value achieves a better performance in terms of PDR and delay in the low dynamic networks. This is because in low dynamic networks, removing many poor links is safe as the possibilities for these poor links to become good links is low. dCut with a high α value will result in fewer loops, limiting the negative impact of routing loops. (2) A low α value achieves better performance in high dynamic network. This is because in high dynamic networks, it is possible that poor links become good links. Therefore, removing too many poor links may not be beneficial. dCut with a low α value will preserve large routing diversities which are required for high dynamic networks.

Since the network dynamics can be quantified with the number of parent change which can be measured and observed by the network operator. The network operator can use the number of parent change to select the optimal value of α according to Table 1 which shows how the optimal α value relates to the number of parent change according to our experiment results.

7.5 FlexCut with RPL

RPL, the IPv6 Routing Protocol for Low-Power and Lossy Networks [9], was standardized by the IETF in 2011 to establish a common ground for the rapidly growing market of Internet of Things (IoT) featured with low-power and lossy networks (LLNs). Many ideas of the RPL was originally developed as part of the TinyOS collaboration and its default routing protocol, CTP [1]. In RPL, Objective Function (OF) determines the mechanism in which a parent is selected. Currently two different types of OFs are specified: (1) Objective Function Zero (OF0) which is simply a hop count-based metric, (2) Minimum Rank Objective (MRHOF) which selects the path with the smallest metric value (e.g., path ETX).

Most RPL implementations (e.g., TinyRPL [15] and ContikiRPL [16]) use MRHOF by default due to better energy efficiency. Since the MRHOF uses the same routing metric as CTP, we expect that FlexCut can also be applied to RPL with the similar results.

To confirm this, we conduct simulation studies to see the key metrics of CTP and RPL. For CTP, we use the TOSSIM simulator. For RPL, we use ContikiRPL with the Cooja simulator. In both simulations, we use a 400-node network topology. We deploy 400 nodes in a 40m $\times$ 40m area, which is divided into 400 squares. Each node is randomly deployed in one square. Each node generates a packet every 2 min or 5 min. Each experiment lasts for two simulation hours and is repeated for 10 times. Table 2 shows the average PDR, parent change (per packet), and the measured loop count of the two protocols.

Results show that RPL and CTP have similar performance. In particular, RPL nodes experienced slightly higher churn compared to CTP. For example, with 2 min interval, CTP experienced an average number of 0.121 parent changes per packet while RPL experienced a slightly higher 0.133 parent changes. Similar results were also observed by other researchers [15]. By making more frequent parent changes, the network was less stable and FlexCut is thus beneficial to achieve a better performance.

8 Related Work

Our work introduces flexible control over distributed routing in sensor networks. We first introduce representative routing protocol and then the control mechanisms over these protocols.

8.1 Routing Protocols

Most sensor networks employ distributed and dynamic routing protocols [1], [9], [17]. CTP [1] is the default routing

protocol in TinyOS. It is a distance vector routing protocol with the routing metric being ETX [2] which is the expected number of transmissions over a path. In their experiments, the authors of CTP also noticed that link dynamics and *transient loops* are two *dominant* factors for the poor data collection performance. The more recent RPL [9] protocol is a routing protocol specifically designed for Low power and Lossy Networks (LLN) compliant with the 6LoWPAN protocol. In RPL, loop could occur when a node loses its parents and selects the node in its own sub-DODAG (Destination Oriented DAG) as the new parent. This might happen particularly when DIO (DODAG Information Object) messages are lost. Both CTP and RPL employ data path validation and topology repair mechanisms when loops occur while our approach tries to limit the formation of loops in the first place.

There are routing protocols for sensor networks, e.g., opportunistic routing [18], [19], [20], and backpressure routing [21]. In opportunistic routing, any node in the forwarder set can help forward the packet when it opportunistically receives the packet. It is possible that temporary loops can occur. FlexCut can naturally be applied in opportunistic routing protocols so that the forwarder set can be optimized to avoid potential loops. In backpressure routing, routing decisions are made to (roughly) minimize the sum of squares of queue backlogs in the network from one timeslot to the next. Backpressure routing is theoretically proved throughput optimal and there are significant works on adapting it to different scenarios. However, backpressure routing may have poor delay performance when packets traverse loops in the network. Our work is helpful for both opportunistic routing and backpressure routing.

There are loop-free routing protocols. AODV [10] uses destination-generated sequence numbers to synchronize routing topology changes and prevent loops. The tradeoff is that when a link goes down, the entire subtree whose root used that link is disconnected until an alternate path is found. Loop free backpressure (LFBP) [11] is a protocol that forwards packets along directed acyclic graphs (DAGs) to avoid the looping problem. Our algorithm can also be used for constructing the needed DAG. Different loop-free routing protocols essentially trade different levels of diversity for forwarding the packets. For a particular loop-free routing protocol, a *fixed* tradeoff is usually made. Our current work provides an abstraction which can trade *arbitrary* diversity for better network performance. For example, FlexCut can provide the needed routing structure for backpressure. Such a routing structure may not be *necessarily* loop-free so that the routing algorithm can exploit more candidate forwarders to achieve better network performance. Moreover, our current work resides in the 2.5 layer and could benefit many other Layer 3 routing protocols.

Bloom filters can be employed to prevent loops. In [12], [13], an extra Bloom filter field is added into the data packets to record the routing path so that nodes already traversed would not be selected again. A key difference between FlexCut and bloom-filter-based forwarding is when to avoid the loops. FlexCut, in its most aggressive form, avoids loops *before* the routing actions taking place. Bloom-filter-based forwarding, on the contrary, avoids loops *during* the routing process. It is possible that bloom-filter-based forwarding may not be able to find a loop-free path towards the sink.

8.2 Control over Routing

There are many research efforts to enforce control over routing protocols. SDN is an approach to computer networking that allows network administrators to manage network services through abstraction of lower-level functionality. With SDN, administrators can easily control the forwarding behaviors of the network [5], [22]. SDN has also been applied in sensor networks [23], [24], [25].

Luo et al. propose Sensor OpenFlow [23], a software-defined routing architecture for sensor networks. This architecture is similar to the SDN architecture for the Internet. The routing action defines the specific routing action for the matched packets, e.g., forward to a specific port or drop the packets. A centralized controller uses a customized version of OpenFlow to interact with the sensor nodes. While Sensor OpenFlow borrows ideas from OpenFlow, it addresses WSN-specific challenges such as how to create flows, how to reduce control traffic, and how to incorporate in-network processing. SDN-WISE [24] is another recent work that presents a SDN solution for wireless sensor networks. Different from the existing SDN solutions for wireless sensor networks, SDN-WISE is stateful and makes sensor nodes programmable as finite state machines, thus enabling them to run operations that cannot be supported by stateless solutions. Although these approaches implement mechanisms for flexibly controlling each node's forwarding behavior, they do not provide high-level abstractions for achieving network-level requirements, e.g., loop-free. Moreover, these approaches require individually controlling each node's forwarding behavior, introducing large control overhead.

TeleAdjusting [14] is a ready-to-use protocol to remotely control any individual node in a WSN. Our work can employ the TeleAdjusting protocol for notifying each node the blacklisted forwarders. pTunes [26] is a framework for runtime adaptation of low-power MAC protocol parameters. pTunes improves the performance of WSN by automatically determining optimized parameters based on application requirements. Different from pTunes, our current work focuses on reshaping the network topology rather than MAC-layer parameters. As described in [27], the network topology can have significant impacts on the overall network performance. In our future work, we would like to consider other factors impacting the network performance.

9 Conclusion

In this paper, we introduce FlexCut, a flexible approach for cutting off wireless links, which essentially limits the candidate forwarder set of each node. Unlike existing SDN solutions, FlexCut introduces flexible control over existing distributed and dynamic routing protocols. FlexCut can trade arbitrary amounts of routing dynamics for better network performance by exposing to network operators a parameter which quantifying the aggressiveness. We model the network as a directed graph with link weights. Each node in the graph points to its candidate forwarder set. The goal of FlexCut is to cut off *user-defined* number of links so that loops can be alleviated while routing flexibility can be preserved to the largest extent. We propose novel algorithms, both centralized and distributed, for addressing these problems.

There are multiple future research directions. First, we would like to automate the selection of the α value for different network topologies. Second, we would like to apply

FlexCut to more Layer 3 network protocols to evaluate to what extent FlexCut can benefit these protocols.

Acknowledgments

This work is supported by the National Science Foundation of China under Grant No. 61772465, No. 61472360, and No. 61502417, Alibaba-Zhejiang University Joint Institute of Frontier Technologies, Zhejiang Provincial Key Research and Development Program (No. 2017C02044), and the Fundamental Research Funds for the Central Universities (No. 2017FZA5013 and No. 2018FZA5013).

References

[1] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, and P. Levis, "CTP: An efficient, robust, and reliable collection tree protocol for wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 10, no. 1, pp. 16:1–16:49, 2013.

[2] D. S. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing," *Wireless Netw.*, vol. 11, no. 4, pp. 419–434, 2005.

[3] W. Dong, Y. Liu, Y. He, T. Zhu, and C. Chen, "Measurement and analysis on the packet delivery performance in a large-scale sensor network," *IEEE/ACM Trans. Netw.*, vol. 22, no. 6, pp. 1952–1963, Dec. 2014.

[4] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. ACM/IEEE 4th Int. Symp. Inf. Process. Sensor Netw.*, 2005, pp. 364–369.

[5] B. Nunes, M. Mendonca, X.-N. Nguyen, K. Obraczka, T. Turletti, et al., "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, Jul.–Sep. 2014.

[6] P. Eades, X. Lin, and W. F. Smyth, "A fast effective heuristic for the feedback arc set problem," *Inf. Process. Lett.*, vol. 47, pp. 319–323, 1993.

[7] B. Berger and P. W. Shor, "Approximation algorithms for the maximum acyclic subgraph problem," in *Proc. 1st Annu. ACM-SIAM Symp. Discrete Algorithms*, 1990, pp. 236–243.

[8] G. Even, J. S. Naor, B. Schieber, and M. Sudan, "Approximating minimum feedback sets and multicuts in directed graphs," *Algorithmica*, vol. 20, pp. 151–174, 1998.

[9] O. Gaddour and A. Koubâa, "RPL in a nutshell: A survey," *Comput. Netw.*, vol. 56, no. 14, pp. 3163–3178, 2012.

[10] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proc. IEEE Workshop Mobile Comput. Syst. Appl.*, 1999, pp. 90–100.

[11] A. Rai, C.-P. Li, G. S. Paschos, and E. Modiano, "Loop-free backpressure routing using link-reversal algorithms," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2015, pp. 87–96.

[12] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," *Internet Math.*, vol. 1, no. 4, pp. 485–509, 2004.

[13] A. Whitaker and D. Wetherall, "Forwarding without loops in Icarus," in *Proc. IEEE Open Archit. Netw. Program.*, 2002, pp. 63–75.

[14] D. Liu, Z. Cao, X. Wu, Y. He, X. Ji, and M. Hou, "Tele adjusting: Using path coding and opportunistic forwarding for remote control in WSNs," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst.*, 2015, pp. 716–725.

[15] K. JeongGil, Stephen, G. Omprakash, C. David, and T. Andreas, "Evaluating the performance of RPL and 6LoWPAN in TinyOS," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2011, pp. 85–90

[16] N. Tsiftes, J. Eriksson, and A. Dunkels, "Low-power wireless IPv6 routing with ContikiRPL," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2010, pp. 406–407.

[17] Z. Li, W. Du, Y. Zheng, M. Li, and D. O. Wu, "From rateless to hopless," in *Proc. ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2015, pp. 107–116.

[18] O. Landsiedel, E. Ghadimi, S. Duquennoy, and M. Johansson, "Low power, low delay: Opportunistic routing meets duty cycling," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2012, pp. 185–196.

[19] S. Biswas and R. Morris, "ExOR: Opportunistic multi-hop routing for wireless networks," in *Proc. ACM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2005, pp. 133–144.

[20] D. Liu, Z. Cao, J. Wang, Y. He, M. Hou, and Y. Liu, "DOF: Duplicate detectable opportunistic forwarding in duty-cycled wireless sensor networks," in *Proc. IEEE Int. Conf. Netw. Protocols*, 2013, pp. 1–10.

[21] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: The backpressure collection protocol," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2010, pp. 279–290.

[22] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 43–56.

[23] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: Enabling software-defined wireless sensor networks," *IEEE Commun. Lett.*, vol. 16, no. 11, pp. 1896–1899, Nov. 2012.

[24] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for WIreless SEnsor networks," in *Proc. IEEE INFOCOM*, 2015, pp. 513–521.

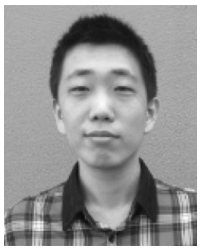
[25] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "UbiFlow: Mobility management in urban-scale software defined IoT," in *Proc. IEEE INFOCOM*, 2015, pp. 208–216.

[26] M. Zimmerling, F. Ferrari, L. Mottola, T. Voigt, and L. Thiele, "pTUNES: Runtime parameter adaptation for low-power MAC protocols," in *Proc. ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, 2012, pp. 173–184.

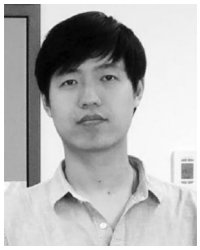
[27] D. Puccinelli, O. Gnawali, S. Yoon, S. Santini, U. Colesanti, S. Giordano, and L. Guibas, "The impact of network topology on collection performance," in *Proc. Eur. Conf. Wireless Sensor Netw.*, 2011, pp. 17–32.

Wei Dong received the BS and PhD degrees from the College of Computer Science, Zhejiang University, in 2005 and 2011, respectively. He is currently a full professor with the College of Computer Science, Zhejiang University. He leads the Embedded and Networked Systems (EmNets) Lab, Zhejiang University. He has published more than 100 papers in prestigious conferences and journals including MobiCom, INFOCOM, ICNP, the *IEEE/ACM Transactions on Networking*, the *IEEE Transactions on Mobile Computing*, etc. His research interests include Internet of Things and sensor networks, wireless and mobile computing, and network measurement. He is a member of the IEEE and ACM.

Gonglong Chen received the BS degree from the College of Criminal Justice, East China University of Political Science and Law. He is currently working toward the PhD degree in the College of Computer Science, Zhejiang University. His current research interests include wireless and mobile computing. He is a student member of the IEEE.



Xiaoyu Zhang received the MS degree from Zhejiang University, in 2017. He is now a software development engineer for NetEase games. His research interests include sensor networks and wireless sensing.



Yi Gao received the BS and PhD degrees from Zhejiang University, in 2009 and 2014, respectively. He is currently a research assistant professor with Zhejiang University, China. From 2015 to 2016, he visited McGill University as a visiting scholar. His research interests include network measurement, sensor networks, and Internet of Things. He is a member of the IEEE and ACM.