Research Article

DFDP: A Distributed Algorithm for Finding Disjoint Paths in Wireless Sensor Networks with Correctness Guarantee

Kejia Zhang, Guisheng Yin, Qilong Han, and Junyu Lin

College of Computer Science and Technology, Harbin Engineering University, Harbin 150001, China

Correspondence should be addressed to Kejia Zhang; kejiazhang@hrbeu.edu.cn

Received 13 January 2014; Revised 31 May 2014; Accepted 1 June 2014; Published 22 June 2014

Academic Editor: Ai Chen

Copyright © 2014 Kejia Zhang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In wireless sensor networks, routing messages through multiple (node) disjoint paths between two sensor nodes is a promising way to increase robustness, throughput, and load balance. This paper proposes an efficient distributed algorithm named distributedly finding disjoint paths (DFDP) to find $k$ disjoint paths connecting two given nodes $s$ and $t$. A set of paths connecting $s$ and $t$ are disjoint if any two of them do not have any common nodes except $s$ and $t$. Unlike the existing distributed algorithms, DFDP guarantees correctness; that is, it will output $k$ disjoint paths if there exist $k$ disjoint paths in the network or the maximum number of disjoint paths otherwise. Compared with the centralized algorithms which also guarantee correctness, DFDP is shown to have much better efficiency and load balance by theory analysis and simulation results.

1. Introduction

A typical wireless sensor network (WSN) comprises a large number of small-size battery-powered sensor nodes deployed in a large area. Since the radio transmitting range of sensor nodes is very limited, data exchanging in WSNs usually needs to be relayed in a multihop manner, which makes efficient message routing a crucial problem. Instead of routing messages through a single path, multipath routing, especially routing through multiple (node) disjoint paths, has become more and more popular [1–5]. Let $s$ and $t$ denote a routing request, that is, two sensor nodes that want to communicate with each other; a set of paths connecting $s$ and $t$ (i.e., $s \sim t$ paths) are (node) disjoint if any two of them do not have any common nodes except $s$ and $t$.

Compared to transmitting data through a single path, transmitting data through multiple disjoint paths can increase throughput, robustness, and load balance of routing in WSNs. Sending multiple data packets through multiple disjoint paths simultaneously will increase the throughput between $s$ and $t$ dramatically. Since link and node failures are very common in WSNs, messages may be easily lost while they are routed along a single path. We can increase reliability by sending messages redundantly through multiple disjoint paths. On the other hand, routing messages often through a fixed single path would deplete the energy of the nodes on the path quickly. By routing messages through several disjoint paths, alternately, the network can gain a better load balance. Moreover, in some applications, a sensitive message can be easily captured by eavesdropping nodes while it is routed through a single path. However, if we break the sensitive message into small pieces and send these pieces through multiple disjoint paths, capturing the entire message would be much more difficult.

This paper investigates the $k$-disjoint-paths ($k$-DP) problem: in a WSN, given two nodes $s$ and $t$, a positive integer $k$, finding $k$ disjoint $s \sim t$ paths, we expect that the $k$-DP problem can be solved with correctness guarantee; that is, an algorithm will output $\min\{k, k^*\}$ disjoint $s \sim t$ paths. Here, we assume that there exist up to $k^*$ disjoint $s \sim t$ paths.
in the given network. There are some works [1–12] considering the problem of finding multiple disjoint \( s \sim t \) paths. To the best of our knowledge, all the existing algorithms which can solve the \( k \)-DP problems with correctness guarantee are centralized algorithms. These centralized algorithms need to build the topology of the whole network and update it with every topology change, so their communication costs are overwhelming for a large-scale WSN. On the other hand, all the existing algorithms finding disjoint paths in a distributed way (so they are much more efficient) cannot guarantee correctness. Therefore, we have two challenges to solve the \( k \)-DP problem in WSNs: to guarantee correctness and to maintain efficiency.

This paper proposes a distributed algorithm named distributedly finding disjoint paths (DFDP) to solve the \( k \)-DP problem with correctness guarantee. DFDP finds disjoint paths in a totally distributed way; that is, each node participates by exchanging short messages with its one-hop neighbors (hereinafter, “neighbor” means one-hop neighbor) and the exchange of messages does not rely on any predefined structures. DFDP is very energy-efficient and can adapt to topology changes with very limited extra cost. Both theoretical analysis and simulation results confirm the energy-efficiency of DFDP. As far as we know, DFDP is the first totally distributed algorithm which can solve the \( k \)-DP problem with correctness guarantee.

The rest of the paper is organized as follows. Some related works are introduced in Section 2. After giving some preliminary definitions in Section 3, we describe the proposed distributed algorithm DFDP in Section 4. The simulation results which confirm the proposed algorithm’s efficiency are presented in Section 5. Finally, we conclude the whole paper in Section 6.

2. Related Works

Some works consider finding two \( s \sim t \) paths in a network. Two algorithms are presented in [13, 14] to find a pair of disjoint paths from each node to a fixed destination. The work in [15] considers the problem of finding two \( s \sim t \) paths with the following optimizing goals: (1) the two identified paths are as disjoint as possible; (2) the second path is as short as possible compared with the first path. The work in [16] gives a distributed algorithm to check if the network configuration contains at least two disjoint paths between all node pairs.

For general \( k \), some distributed algorithms [1, 3–5, 11, 12] are proposed for finding \( k \) disjoint \( s \sim t \) paths in a network. These algorithms have the same basic idea. They find paths one by one (most of them find the shortest \( s \sim t \) path). Once a path is found, it is fixed and all its internal nodes are removed from the network so that they cannot participate in another path. To solve the \( k \)-DP problem, these algorithms cannot guarantee correctness. Suppose that, in a network as shown in Figure 1, the users want to find \( k = 2 \) disjoint \( s \sim t \) paths, and we have already found the first path \( P_1 = (s, a, b, c, t) \) as shown by the bold lines in Figure 1(a). If we fix \( P_1 \), we cannot find another \( s \sim t \) path disjoint from it, and we have to report nonexistence of two disjoint \( s \sim t \) paths. However, there do exist two disjoint \( s \sim t \) paths as shown in Figure 1(d). In some cases, the above mentioned distributed approaches can be very wrong in answering the \( k \)-DP problem by underestimating the disjoint paths. Suppose that the topology graph of a given network is as shown in Figure 2, in which the dashed lines denote paths. If the users want to find \( k = 5 \) disjoint \( s \sim t \) paths and we find the first path as shown by the bold lines in Figure 2(a) and then fix this path, this path will be the only one \( s \sim t \) path we can find, even though five disjoint paths do exist as shown in Figure 2(b).

In [17], a distributed algorithm is proposed to find \( k \) disjoint paths from every node to a fixed destination in a network. The algorithm is based on a shortest-path tree
rooted at the destination. The work in [18] gives a distributed algorithm to find multiple disjoint $s \sim t$ paths for geographic routing. These two algorithms cannot guarantee correctness either.

We propose a pseudodistributed algorithm in [19] to solve the $k$-DP problem in WSNs with correctness guarantee. The algorithm does not process in a totally distributed manner. Although it does not need to collect the topology information of the whole network, it still needs to collect some topology information from the network, so it is not very efficient. Furthermore, the algorithm is complicated and hard to be implemented by sensor nodes.

In [2, 6–10, 20], some centralized algorithms are given for finding $k$ disjoint $s \sim t$ paths in graphs. In particular, the algorithms in [6, 9, 20] can find $k$ disjoint $s \sim t$ paths with the minimum total length. All these centralized algorithms guarantee correctness. If we use these centralized algorithms to solve the $k$-DP problem in a WSN, we will have to collect every node’s neighbor list to build the topology of the whole network. The communication cost of such operation is very high for a large-scale network. On the other hand, to guarantee correctness, we have to update the network’s topology with each topology change such as link failure, link recovering, node dying, and new node joining in. Since these topology changes happen frequently in WSNs, we need to pay a lot of communication to make the topology we built up-to-date.

The works in [21–23] consider the problem of finding the maximum number of length-bounded disjoint $s \sim t$ paths in graphs. A length-bounded path is a path whose length is no more than a user-specified threshold $l$. In [22], this problem is proven to be NP-hard for $l \geq 5$. In [21], this problem is further proven to be APX-complete for $l \geq 5$, which means that there is no polynomial time approximation scheme (PTAS) for this problem. A heuristic algorithm for this problem is given in [23] without approximation bound.

For $k$ multiple $s \sim t$ paths, we can give each path a sequence number. Each edge has $k$ costs to be in the $1$st, $\ldots$, $k$th path. Such a network is called a multicost network. Some centralized algorithms are given in [24–26] to find $k$ disjoint $s \sim t$ paths in multicost networks with the minimum total cost. The works in [27] consider the problem of reliable data transmission in underwater sensor networks. Its solution is based on coding techniques rather than finding disjoint paths. To transmit a series of data packets, by effective coding, its algorithm ensures that certain amount of the packets missing does not affect our access to the complete data.

So far, all the existing algorithms which can solve the $k$-DP problem with correctness guarantee are centralized algorithms. These centralized algorithms need to build the topology of the whole network and update it with every topology change, so they are not applicable in WSNs. On the other hand, all the existing distributed algorithms cannot guarantee correctness, although they are efficient.

### 3. Preliminaries

The given WSN can be expressed as an undirected graph $G = (V, E)$, where $V = \{v \mid v$ is a sensor node$\}$ and $E = \{(u, v) \mid$ there is a wireless link between $u \in V$ and $v \in V\}$. Here, we assume that all the links are bidirectional; that is, $u$ can hear $v$ if $v$ can hear $u$. Hereinafter, we do not distinguish the terms "network" and "graph."

A path $P$ in $G$ is a subgraph which can be expressed as a sequence of distinct nodes $(v_1, \ldots, v_m)$, where $v_i \in V$ and $(v_i, v_{i+1}) \in E$. We say path $P = (v_1, \ldots, v_m)$ is a $v_1 \sim v_m$ path. Although we model the given network as an undirected graph, we still emphasize the direction of a path by nodes’ order in its expression or the words "from" and "to." Thus, the direction of path $P = (v_1, v_2, v_3, \ldots, v_m)$ is from $v_1$ to $v_m$. $v_i$’s previous hop in $P$ is $v_{i-1}$ and its next hop in $P$ is $v_{i+1}$. The default direction of each $s \sim t$ path is from $s$ to $t$. A reverse segment of a $s \sim t$ path is a segment of the path along the direction from $t$ to $s$. In the example of Figure 1(a), $(c, b, a)$ is a reverse segment of $s \sim t$ path $P_1$.

Suppose that we have found $n$ disjoint $s \sim t$ paths $P_1, \ldots, P_n$ in the given network. The nodes (edges) in the found paths are called occupied nodes (edges), and the other nodes (edges) are called unoccupied nodes (edges). Let $v$ be an occupied node. We, respectively, use $\text{prev}(v)$ and $\text{next}(v)$ to denote $v$’s previous hop and next hop in the found path that $v$ belongs to. For example, in the network as shown in Figure 1(a), after finding $P_1$, $b$ is an occupied node. $\text{prev}(b) = a$ and $\text{next}(b) = c$.

After finding some disjoint $s \sim t$ paths, we convert the original graph $G$ to another graph $G^*$ by replacing each path edge with a directed edge along the direction from $t$ to $s$. In $G^*$, each found path becomes a “river” flowing from $t$ to $s$. In the example of Figure 1(a), after finding $P_1$, we convert $G$ to $G^*$ as shown in Figure 1(b). Actually, $G^*$ is another way to
interpreted $G$. In the following, we do not distinguish the path in $G$ and its corresponding path in $G^*$. We say that a path follows the crossing restriction if the path does not go through any occupied nodes except by going downstream on some river in $G^*$ for at least one hop (i.e., going through a reverse segment of some found path). A path following the crossing restriction is called a CR-path. After finding the first path $P_1$, let us examine some paths in the converted graph as shown in Figure 1(b). $Q_1 = (s, d, a, f)$ is not a CR-path because it goes through occupied node $a$ by directly crossing the river. $Q_2 = (s, d, a, b, c, e)$ is not a CR-path because it goes through occupied nodes $a, b, c$ by going upstream on the river. $Q_3 = (s, d, e, c, b, a, f)$ is a CR-path because it goes through occupied nodes $c, b, a$ by going downstream on the river for segment $(c, b, a)$.

A CR-path from $s$ to $t$ is an augmenting path. By adding an augmenting path $P_{n}$ to the found paths $P_1, \ldots, P_{n-1}$, we can get $n+1$ disjoint $s \sim t$ paths, where “add” means adding the edges that only belong to $P_n$ and removing the overlapping edges of $P_n$ and $P_1, \ldots, P_{n-1}$. In the example of Figure 1(a), after finding the first path $P_1 = (s, a, b, c, t)$, we can find an augmenting path $P_2 = (s, d, e, c, b, a, f, g, t)$ as shown by bold lines in Figure 1(c). By adding $P_n$ to $P_1$, we can get two disjoint $s \sim t$ paths $P_n' = (s, d, e, c, t)$ and $P_n'' = (s, a, f, g, t)$ as shown by bold lines in Figure 1(d).

Lemma 1. After finding $n$ disjoint $s \sim t$ paths, there exist more than $n$ disjoint $s \sim t$ paths if and only if there exists an augmenting path [8].

Lemma 1 can be easily proven by transforming the problem of finding disjoint paths to a maximum flow problem. It means that we can find the maximum number of disjoint $s \sim t$ paths by searching for augmenting path repeatedly.

4. Distributed Algorithm DFDP

In the given network $G$, the users specify two nodes $s$ and $t$ and a positive integer $k$; the proposed algorithm DFDP will output $\min\{k, k^*\}$ disjoint $s \sim t$ paths. Here, we assume that there exist up to $k^*$ disjoint $s \sim t$ paths in the given network. DFDP is iteratively searching for augmenting path and adding it to the paths found before. There are two procedures in each iteration. Procedure 1 searches for an augmenting path $P_n$. Procedure 2 adds $P_n$ to the paths found before to generate more disjoint paths. DFDP is given by Algorithm 1. We assume that each message contains the sender's ID. Each occupied node $v$ knows $\text{prev}(v)$ and $\text{next}(v)$.

4.1. Finding Augmenting Path. Suppose that we have found $n$ disjoint $s \sim t$ paths $P_1, \ldots, P_n$ in the first $n$ iterations. In the $(n+1)$th iteration, Procedure 1 searches for an augmenting path by finding a CR-path from $s$ to each node in the network. To make sure that the paths we found follow the crossing restriction, we divide the nodes in a CR-path into three categories: unoccupied node (UN), first hop of a reverse segment (FHR), and other hops in a reverse segment (OHR) of some found path. In the example of Figure 1(b), for CR-path $P_n = (s, d, e, c, b, a, f, g, t)$, the nodes $\{d, e, f, g\}$ are UNs, node $c$ is a FHR, and nodes $\{b, a\}$ are OHRs.

Each unoccupied node $v$ maintains variable UN$_\text{phcr}(v)$ to record $v$'s previous hop in the CR-path we found from $s$ to $v$. Each occupied node $v$ maintains variables FHR$_\text{phcr}(v)$ and OHR$_\text{phcr}(v)$. FHR$_\text{phcr}(v)$ records $v$'s previous hop in the CR-path we found from $s$ to $v$ with $v$ as a FHR. OHR$_\text{phcr}(v)$ records $v$'s previous hop in the CR-path we found from $s$ to $v$ with $v$ as an OHR.

Procedure 1 starts with $s$ broadcasting a FIND-AP message. The other nodes find a CR-path from $s$ to themselves by exchanging FIND-AP messages. Each FIND-AP message has an empty data field. Next, we discuss the following four cases.

Case 1. Unoccupied node $v$ receives a FIND-AP message (let the sender be $u$). It means that we find a CR-path (denoted by $Q$) from $s$ to $v$ and $u$ is $v$'s previous hop in $Q$. Since $v$ is an unoccupied node, $v$ is an UN in $Q$. To further extend $Q$ as a CR-path, $v$'s next hop in this path could be any neighbor of $v$.

Case 2. Occupied node $v$ receives a FIND-AP message from $u$ and $u$ is $\text{prev}(v)$. It means that we find a path $Q$ from $s$ to $v$ and $u$ is $v$'s previous hop in $Q$. Since $u$ is $\text{prev}(v)$, $Q$ does not follow the crossing restriction. In this case, we let $v$ discard the message.
Input: \( n \) disjoint \( s \sim t \) paths \( P_1, \ldots, P_n \)
Output: an augmenting path \( P_a \)

/* Each node \( v \neq s \) executes the following pseudo-codes while receiving a FIND-AP message (Let the sender be \( u \)); Start with \( s \) broadcasting a FIND-AP message. */

(1) case \( v \) is an unoccupied node:
(2) if \( \text{UN}_{\text{phcr}}(v) = \text{NULL} \) then
    (3) \( \text{UN}_{\text{phcr}}(v) \leftarrow u; \)
    (4) Broadcast a FIND-AP message;
(5) case \( v \) is an occupied node and \( u \) is not \( \text{prev}(v) \) or \( \text{next}(v) \):
    (6) if \( \text{FHR}_{\text{phcr}}(v) = \text{NULL} \) then
        (7) \( \text{FHR}_{\text{phcr}}(v) \leftarrow u; \)
        (8) if \( v = t \) then
            (9) Start Procedure 2;
        (10) else
            (11) Send a FIND-AP message to \( \text{prev}(v) \);
    (12) case \( v \) is an occupied node and \( u = \text{next}(v) \):
        (13) if \( \text{OHR}_{\text{phcr}}(v) = \text{NULL} \) then
            (14) \( \text{OHR}_{\text{phcr}}(v) \leftarrow u; \)
            (15) Broadcast a FIND-AP message;

Procedure 1: Finding augmenting path.

Input: \( n \) disjoint \( s \sim t \) paths \( P_1, \ldots, P_n \); an augmenting path \( P_a \)
Output: \( n + 1 \) disjoint \( s \sim t \) paths \( P_1', \ldots, P_{n+1}' \)

/* Each node \( v \neq t \) executes the following pseudo-codes while receiving a TRACE-AP message (Let the sender be \( u \)); Starts with \( t \) sending a TRACE-AP message to \( \text{FHR}_{\text{phcr}}(t) \). */

(1) case \( v \) is an unoccupied node:
(2) Mark itself as an occupied node;
(3) \( \text{prev}(v) \leftarrow \text{UN}_{\text{phcr}}(v); \)
(4) \( \text{next}(v) \leftarrow u; \)
(5) Send a TRACE-AP message to \( \text{UN}_{\text{phcr}}(v); \)
(6) case \( v \) is an occupied node and \( u \neq \text{prev}(v) \):
    (7) Still mark itself as an occupied node;
    (8) \( \text{next}(v) \leftarrow u; \)
    (9) Send a TRACE-AP message to \( \text{OHR}_{\text{phcr}}(v); \)
    (10) case \( v \) is an occupied node and \( u = \text{prev}(v) \) and \( \text{FHR}_{\text{phcr}}(v) = \text{NULL} \):
       (11) Mark itself as an unoccupied node;
       (12) Send a TRACE-AP message to \( \text{OHR}_{\text{phcr}}(v); \)
       (13) case \( v \) is an occupied node and \( u = \text{prev}(v) \) and \( \text{FHR}_{\text{phcr}}(v) \neq \text{NULL} \):
           (14) Still mark itself as an occupied node;
           (15) \( \text{prev}(v) \leftarrow \text{FHR}_{\text{phcr}}(v); \)
           (16) Send a TRACE-AP message to \( \text{FHR}_{\text{phcr}}(v); \)

Procedure 2: Tracing augmenting path.

Case 3. Occupied node \( v \) receives a FIND-AP message from \( u \) and \( u \) is not \( \text{prev}(v) \) or \( \text{next}(v) \). It means that we find a CR-path \( Q \) from \( s \) to \( v \) and \( u \) is \( v \)'s previous hop in \( Q \). Since \( u \) is not \( \text{prev}(v) \) or \( \text{next}(v) \), \( v \) is a FHR in \( Q \). To further extend \( Q \) as a CR-path, \( v \)'s next hop in this path must be \( \text{prev}(v) \). In this case, if \( \text{FHR}_{\text{phcr}}(v) \) is not set yet, we set \( \text{FHR}_{\text{phcr}}(v) \) to \( u \) and let \( u \) send a FIND-AP message to \( \text{prev}(v) \) (lines 6–11 in Procedure 1).

Case 4. Occupied node \( v \) receives a FIND-AP message from \( u \) and \( u \) is \( \text{next}(v) \). It means that we find a CR-path \( Q \) from \( s \) to \( v \) and \( u \) is \( v \)'s previous hop in \( Q \). Since \( u = \text{next}(v) \), \( v \) is an OHR...
in Q. To further extend Q as a CR-path, v's next hop in this path could be any neighbor of v except next(v). In this Case, if OHR_pchr(v) is not set yet, we set OHR_pchr(v) to u and let u broadcast a FIND-AP message (lines 13–15 in Procedure 1). Although this message will be redundantly received by next(v), next(v) will discard this message according to Case 2.

We assume that, at the beginning of each iteration, UN_pchr(v) = NULL for each unoccupied node and FHR_pchr(v) = OHR_pchr(v) = NULL for each occupied node. For example, UN_pchr(v) = NULL means that v (as an UN) has not received a FIND-AP message yet in the current iteration. It can be implemented by the following mechanism. Each node v maintains a variable round(v) to denote the number of the current iteration. At the beginning of each new iteration, s updates the iteration number by adding 1 to the number of the last iteration. This number is included by every FIND-AP message so that each node can update its round(v).

4.2. Tracing Augmenting Path. At first, let us describe the last hop in a reverse segment of some found path. In the example of Figure 1, after finding the augmenting path P_a as shown in Figure 1(c), we say a is the last hop of the reverse segment (c, b, a). By adding P_a to the paths found before, a should be an occupied node, prev(a) does not change, and next(a) changes to its next hop in P_a, that is, f. In a CR-path, the last hop of a reverse segment is certainly an OHR.

Suppose that we find an augmenting path P_a in Procedure 1. Procedure 2 adds P_a to the paths found before. Starting from t, in sequence, each node in P_a sends a TRACE-AP message (with empty data field) to its previous hop in P_a. At first, t sends a TRACE-AP message to FHR_pchr(t) to start the procedure. Next, we discuss the following four cases.

Case 5. Unoccupied node v receives a TRACE-AP message (let the sender be u). Then v is an UN in P_a and its next hop in P_a is u. Its previous hop in P_a should be UN_pchr(v). To add P_a to the paths found before, v becomes an occupied node. prev(v) should be UN_pchr(v) and next(v) should be u. In this case, we let v execute lines 2–5 in Procedure 2.

Case 6. Occupied node v receives a TRACE-AP message from u and u is not prev(v). Then v's next hop in P_a is u. Since u is not prev(v), v is the last hop in a reverse segment of some found path. Of course, v is an OHR in P_a. v's previous hop in P_a should be OHR_pchr(v). By adding P_a to the paths found before, v is still an occupied node. prev(v) does not change and next(v) changes to u. In this case, we let v execute lines 7–9 in Procedure 2.

Case 7. Occupied node v receives a TRACE-AP message from u and u is prev(v) and FHR_pchr(v) = NULL. Then v's next hop in P_a is u. Since u is prev(v), v is not the last hop in a reverse segment. Since FHR_pchr(v) is NULL, v is an OHR in P_a and its previous hop in P_a should be OHR_pchr(v). By adding P_a to the paths found before, v becomes an unoccupied node. In this case, we let v execute lines 11–12 in Procedure 2.

Case 8. Occupied node v receives a TRACE-AP message from u and u is prev(v) and FHR_pchr(v) ≠ NULL. Then v's next hop in P_a is u. Since u is prev(v), v is not the last hop in a reverse segment. Since FHR_pchr(v) is not NULL, v is a FHR in P_a and its previous hop in P_a should be FHR_pchr(v). By adding P_a to the paths found before, v is still an occupied node. prev(v) changes to FHR_pchr(v) and next(v) does not change. In this case, we let v execute lines 14–16 in Procedure 2.

4.3. An Example of DFDP's Execution. In the example of Figure 1, after finding the first s ～ t path P_1 as shown in Figure 1(a), the nodes execute DFDP as follows.

Procedure 1 starts with s broadcasting a FIND-AP message, which is received by a and d. a will discard the message according to Case 2.

Receiving the FIND-AP message from s, according to Case 1, d sets UN_pchr(d) = s and then broadcasts a FIND-AP message, which is received by a and e. Although the message will be also received by s, s will discard the message.

Receiving the FIND-AP message from d, according to Case 3, a sets FHR_pchr(a) = d and then sends a FIND-AP message to s (s will discard the message).

Receiving the FIND-AP message from d, according to Case 1, e sets UN_pchr(e) = d and then broadcasts a FIND-AP message, which is received by d and c. Since UN_pchr(d) is set yet, d discards the FIND-AP message from e.

Receiving the FIND-AP message from e, according to Case 3, c sets FHR_pchr(c) = e and then sends a FIND-AP message to b.

Receiving the FIND-AP message from c, according to Case 4, b sets OHR_pchr(b) = c and then broadcasts a FIND-AP message, which is received by a.

Receiving the FIND-AP message from b, according to Case 4, a sets OHR_pchr(a) = b and then broadcasts a FIND-AP message, which is received by s and f (s will discard the message).

In this way, f sets UN_pchr(f) = a and g sets UN_pchr(g) = f. Receiving FIND-AP message from g, t sets FHR_pchr(t) = g. Therefore, augmenting path P_2 = (s, d, e, c, b, a, f, g, t) is found as shown by bold lines in Figure 1(c).

Procedure 2 starts with t sending a TRACE-AP message to FHR_pchr(t) = g.

Receiving the TRACE-AP message from t, according to Case 5, g marks itself as an occupied node, sets prev(g) = UN_pchr(g) = f and next(g) = t, and then sends a TRACE-AP message to f.

Receiving the TRACE-AP message from g, according to Case 5, f marks itself as an occupied node, sets prev(f) = UN_pchr(f) = a and next(f) = g, and then sends a TRACE-AP message to a.

Receiving the TRACE-AP message from f, according to Case 6, a still marks itself as an occupied node, sets next(a) =
At first, let us compare the efficiency of four algorithms in theory. Suppose that there are $N$ nodes and $M$ edges in the given network. In Procedure 1, each unoccupied node sends at most one FIND-AP message and each occupied node sends at most two FIND-AP messages. In Procedure 2, each node in $P_a$ sends one TRACE-AP message and the other nodes do not send any messages. Both FIND-AP message and TRACE-AP message have constant length (empty data field), so the communication complexity of DFDP is $O(N \cdot \min[k, k^*])$. The time complexity of DFDP is $O(N \cdot \min[k, k^*])$ too.

4.4. Correctness and Complexity. By the description of Sections 4.1 and 4.2, after finding $n$ disjoint $s \sim t$ paths $P_1, \ldots, P_n$, Procedure 1 will find an augmenting path if there exists one, and Procedure 2 is a correct way to add the augmenting path to $P_1, \ldots, P_n$. By Lemma 1, we can get the following theorem.

**Theorem 2.** DFDP will output $\min\{k, k^*\}$ disjoint $s \sim t$ paths, where $k^*$ is the maximum number of disjoint $s \sim t$ paths in the given network.

Suppose that there are $N$ nodes in the given network. In Procedure 1, each unoccupied node sends at most one FIND-AP message and each occupied node sends at most two FIND-AP messages. In Procedure 2, each node in $P_a$ sends one TRACE-AP message and the other nodes do not send any messages. Both FIND-AP message and TRACE-AP message have constant length (empty data field), so the communication complexity of DFDP is $O(N \cdot \min[k, k^*])$. The time complexity of DFDP is $O(N \cdot \min[k, k^*])$ too.

4.5. Dealing with Topology Changes. Let us consider three types of topology changes.

**Type 1.** During the processing of DFDP, the nodes or links in some found path fail. This type of topology changes will cause $s$ not to receive the expected number of TRACE-AP messages (one plus the number of disjoint paths found in the last iteration) in the current iteration. Therefore, we can set a timer for the execution of each iteration. The timer is triggered by $s$ at the beginning of each iteration. If the timer expires and $s$ has not received the expected number of TRACE-AP messages, $s$ knows that there is node or link failure in the path from which it has not received TRACE-AP message. Let the failed path be $P'_2$. Before going into the next iteration, DFDP frees the nodes in $P'_2$ by tracing them from both $s$ and $t$. With the topology changes, DFDP still guarantees correctness. The extra cost for DFDP to deal with this type of topology changes is tracing the failed path (which can be neglected compared to the total cost of the algorithm) and one more iteration of the algorithm (which is inevitable).

**Type 2.** During the processing of DFDP, the topology beyond the found paths changes. Since each node participates in DFDP by communicating with its neighbors and the exchange of messages does not rely on any predefined structures, DFDP can handle this type of topology changes while guaranteeing correctness without any modification.

**Type 3.** After the execution of DFDP (min{$k, k^*$} paths have been used for routing), the nodes or links in some found path fail. After detecting the path failure, we can free the nodes in the failed path and execute one more iteration of DFDP to find one more disjoint path. The extra cost for DFDP to deal with this type of topology changes is also tracing the failed path and one more iteration of the algorithm.

On the contrary, to handle all the three types of topology changes and guarantee correctness, centralized algorithms have to collect the whole network's topology once again, which is a very expensive operation for WSN.

5. Simulation Results

We use a simulator written in C++ to evaluate the proposed algorithm. The simulator is a high level simulator which neglects the MAC layer. The effective transmitting radius of each sensor node is set to 50 m. Each packet has a 6-byte header containing the information of destination's ID (2 bytes), sender's ID (2 bytes), packet type (1 byte), and packet length (1 byte). The data field of each packet contains up to 50 bytes.

To simulate energy consumption, we set the parameters of sensor nodes according to TelosB motes [28]. The supply voltage is 3.0 V. When MCU is on and radio is on TX or RX mode, the node's current is 22 mA. The transmit bit rate is 250 kbps. Therefore, the energy consumption for the sender to send 1-bit data (or for the receiver to receive 1-bit data) successfully is $2.64 \times 10^{-7}$ J. For the TelosB sensor node, the energy consumption to transmit one-bit data can be used for MCU to execute more than 400 instructions, so we neglect the energy consumption for computing.

We compare our algorithm DFDP with the centralized algorithm in [6] (denoted by CA), the pseudodistributed algorithm in [19] (denoted by EA), and the distributed algorithm in [5] (denoted by DA). CA collects the topology information of the whole network (each node's neighbor list) to the sink and finds disjoint paths at the sink. CA can find $k$ disjoint $s \sim t$ paths with the minimum total length. CA is a good representation of the centralized algorithms in [2, 6–10]. All these centralized algorithms have to collect the topology information of the whole network, so they have the same communication cost. On the other hand, all of them guarantee correctness, so they find the same number of disjoint paths. DA finds paths iteratively. In each iteration, DA finds a $s \sim t$ path and removes the internal nodes of the path from the network. As we know, DA is the most efficient algorithm for finding disjoint paths with no other constraints. Note that DA cannot guarantee correctness; that is, DA may only find $n < k$ paths when there do exist $k$ disjoint $s \sim t$ paths in the given network.

At first, let us compare the efficiency of four algorithms in theory. Suppose that there are $N$ nodes and $M$ edges in the network.
network. Let $d$ be the diameter of the network. The energy consumption of the algorithms is mainly determined by data communication. We already know that the communication complexity of DFDP is $O(N \cdot \min(k, k^*))$. CA collects the topology information of the whole network. Each node is $O(d)$ hops away from the sink on average, so the information of each edge is transmitted $O(d)$ hops to the sink on average and the communication complexity of CA is $O(dM)$. EA has bad upper bound on communication complexity. In the worst case, EA has to collect the topology information of the whole network in each iteration, so its communication complexity is $O(dM \cdot \min(k, k^*))$. DA is the simplest algorithm to find disjoint path although it cannot guarantee correctness. In each iteration, each node sends one message with constant length to find a new path from $s$ to $t$, so the communication complexity of DA is $O(N \cdot \min(k, k^*))$. We know that $k \ll d$ in most applications. In the aspect of communication complexity, DFDP and DA are much better than CA and EA. Although with the same communication complexity compared with DA, DFDP can guarantee correctness, whereas DA cannot.

We evaluate the performances of the algorithms at four aspects: (1) path number, (2) average path length, (3) energy-efficiency, and (4) load balance. “Path number” is the number of disjoint $s \rightarrow t$ paths found by the algorithms. “Average path length” represents the quality of the disjoint paths. We use “average energy/path number” to measure algorithms’ energy-efficiency, where “average energy” is the average energy consumption of each node in the network. Load balance is measured by “max energy/path number,” where “max energy” is the maximum energy consumption of any node in the network.

There are two groups of simulations. In Group 1, 2500 sensor nodes are randomly deployed in a $1500 \times 1500$ m area. In Group 2, 2500 sensor nodes are deployed as a $50 \times 50$ grid and each node dies or sleeps with 25% probability. As a result, 1859 nodes are left as alive nodes in the grid. The comparison of the algorithms is conducted for different $k$, that is, the number of the paths the users want to find. To get each result, we performed 10 simulations with different $(s, t)$ pairs and took their average value. The $(s, t)$ pairs are generated randomly.

The comparison results for Group 1 are given in Figure 3. The path number of the four algorithms is given in Figure 3(a). With the growth of $k$, all the four algorithms find more disjoint paths, until they reach their “limits.” DA finds no more disjoint paths when $k \geq 7$. CA, EA, and DFDP find no more disjoint paths when $k \geq 10$. Since CA, EA, and DFDP all guarantee correctness, they always find the same number of disjoint paths. When $k \leq 3$, all the four algorithms can find $k$ disjoint paths. When $k \geq 4$, compared with DA, the other three algorithms find more disjoint paths. CA, EA, and DFDP can find the maximum number of the disjoint paths in the network, whereas DA cannot. For $k = 10$, compared with DA, the other three algorithms find 20.3% more disjoint paths.

Next, we want to compare the average path length of the four algorithms. For different $(s, t)$ pairs, the difference of paths’ average lengths may be very large. To compare fairly, for each $(s, t)$ pair, we set the value of CA to 1 and normalize the values of the other three algorithms. The average path length of the four algorithms for Group 1 is given in Figure 3(b). The centralized algorithm (CA) always finds disjoint paths with the minimum total length. EA and DFDP almost have the same performance as CA in this aspect. In all cases, the average path length of DFDP is at most 3% higher than the one of CA. When $k = 10$, the average path length of DA is even less than the one of CA. This is because DA finds less disjoint paths and all the algorithms have the trends to find shorter path first.

The energy-efficiency of the four algorithms for Group 1 is given in Figure 3(c). No matter what value $k$ is, CA collects the topology information of the whole network, so the energy cost of CA does not change with $k$. With CA finding more disjoint paths, it is more energy-efficient, until it finds the maximum number of disjoint paths. EA finds paths iteratively and collects an abstract of the network topology in each iteration. With finding more disjoint paths, EA needs to collect more and more complicated topology information and becomes less and less energy-efficient. When $k > 7$, EA is even worse than CA in this aspect. DFDP and DA also find paths iteratively and their energy cost for each iteration is steady. DFDP and DA almost have the same performance in this aspect, and both of them are much more energy-efficient than CA and EA.

The load balance of the four algorithms for Group 1 is given in Figure 3(d). Note that the value of the $y$-axis increases logarithmically. Since CA collects the topology information of the whole network, the nodes near the sink need to send and receive a large amount of data, so CA is two orders of magnitude worse than the other three algorithms in this aspect. Although EA is much better than CA, it is still

### Table 1: The simulation details for Group 1 when $k = 10$.

| Performance | Algorithms |
|-------------|------------|
| Path number | CA         | EA         | DFDP       | DA         |
| 7.1         | 7.1        | 7.1        | 5.9        |
| Average path length | 1          | 1.03       | 1.03       | 0.99       |
| Average Comm. Bytes | 845        | 1023       | 353        | 291        |
| Max Comm. Bytes | 58495      | 1616       | 814        | 683        |
| Average energy ($\times 10^{-7}$ J) | 17842      | 21605      | 7462       | 6136       |
| Max energy ($\times 10^{-7}$ J) | 1235407    | 34123      | 17196      | 14431      |

### Table 2: The simulation details for Group 2 when $k = 4$.

| Performance | Algorithms |
|-------------|------------|
| Path number | CA         | EA         | DFDP       | DA         |
| 3.4         | 3.4        | 3.4        | 2.3        |
| Average path length | 1          | 1.04       | 1.02       | 0.84       |
| Average Comm. Bytes | 348        | 170        | 56         | 40         |
| Max Comm. Bytes | 14792      | 568        | 249        | 172        |
| Average energy ($\times 10^{-7}$ J) | 7351      | 3590       | 1183       | 844        |
| Max energy ($\times 10^{-7}$ J) | 312416    | 11995      | 5263       | 3629       |
a “collecting” algorithm, so it is not as good as DFDP and DA in terms of load balance. By finding more disjoint paths, sometimes DFDP even gains a better load balance compared with the simplest distributed algorithm (DA).

The specific simulation results for Group 1 when \( k = 10 \) are shown in Table 1. “Average Comm. Bytes” is the average number of bytes sent and received by each node in the network. “Max Comm. Bytes” is the maximum number of bytes sent and received by any node in the network.

The comparison results for Group 2 are given in Figure 4. The path number of the four algorithms is given in Figure 4(a). With the growth of \( k \), all the four algorithms find more disjoint paths. DA reaches its “limit” when \( k = 3 \), whereas CA, EA, and DFDP reach their “limits” when \( k = 4 \). Since CA, EA, and DFDP all guarantee correctness, they always find the same number of disjoint paths. When \( k = 4 \), compared with DA, the other three algorithms find 47.8% more disjoint paths.

The average path length of the four algorithms for Group 2 is given in Figure 4(b). The centralized algorithm (CA) always finds disjoint paths with the minimum total length. Note that CA, EA, and DFDP find the same number of disjoint paths. No matter what the value of \( k \) is, the average path length of DFDP is at most 2% higher than the one of CA.
The “average energy/path number” of the four algorithms for Group 2 is given in Figure 4(c). In the aspect of energy-efficiency, DFDP and DA are much better than CA and EA. DFDP and DA almost have the same energy-efficiency, but DFDP finds more disjoint paths.

The “max energy/path number” of the four algorithms for Group 2 is given in Figure 4(d). In the aspect of load balance, CA is two orders of magnitude worse than the other three algorithms. DA and DFDP almost have the same performances and both of them are significantly better than EA. The specific simulation for Group 2 when \( k = 4 \) is shown in Table 2.

Let us summarize the simulation results for Group 1 and Group 2. Compared with DA, DFDP finds significantly more disjoint paths. In the aspect of energy-efficiency and load balance, DFDP is almost the same as DA. CA, EA, and DFDP find the same number of disjoint paths with almost the same length. However, in the aspect of energy-efficiency and load balance, DFDP is much better than CA and EA.

6. Conclusion

This paper proposes an efficient distributed algorithm DFDP to find \( k \) disjoint paths connecting two given nodes \( s \) and \( t \) in a WSN. Unlike the existing distributed algorithms for this problem, DFDP guarantees correctness. Compared to the existing centralized algorithms which also guarantee correctness, DFDP is much more efficient. Compared to the state-of-the-art distributed algorithms for the same problem, DFDP
has the same energy-efficiency and load balance, with the important advantage that DFDP guarantees a correct answer, whereas the compared algorithms do not. In conclusion, DFDP is correct and efficient to find disjoint paths between two nodes in a WSN.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant no. 61300207, Grant no. 61272186, and Grant no. 61370084) and Fundamental Research Funds for the Central Universities (Grant no. HEUCF100610).

References

[1] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, "Highly-resilient, energy-efficient multipath routing in wireless sensor networks," in Proceedings of the 2nd ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc '01), pp. 251–254, Long Beach, Calif, USA, October 2001.

[2] A. Srinivas and E. Modiano, "Minimum energy disjoint path routing in wireless ad-hoc networks," in Proceedings of the 9th Annual International Conference on Mobile Computing and Networking (MobiCom '03), pp. 122–133, September 2003.

[3] S. Li and Z. Wu, "Node-disjoint parallel multi-path routing in wireless sensor networks," in Proceedings of the 2nd International Conference on Embedded Software and Systems (ICESS ’05), pp. 432–437, December 2005.

[4] J. W. Baek, Y. J. Nam, and D.-W. Seo, "An energy-efficient k-disjoint-path routing algorithm for reliable wireless sensor networks," in Proceedings of the 5th IFIP WG 10.2 International Conference on Software Technologies for Embedded and Ubiquitous Systems (SEUS'07), 2007.

[5] B. Deb, S. Bhatnagar, and B. Nath, "Reinform: reliable information forwarding using multiple paths in sensor networks," in Proceedings of the 28th Annual IEEE International Conference on Local Computer Networks, 2003.

[6] R. Bhandari, "Optimal physical diversity algorithms and survivable networks," in Proceedings of the 2nd IEEE Symposium on Computers and Communications (ISCC ’97), pp. 433–441, Alexandria, Egypt, July 1997.

[7] S. Khuller and B. Schieber, "Efficient parallel algorithms for testing connectivity and finding disjoint s-t paths in graphs," in Proceedings of the 30th Annual Symposium on Foundations of Computer Science, pp. 288–293, November 1989.

[8] K. Iwama, C. Iwamoto, and T. Ohswara, "A faster parallel algorithm for k-connectivity," Information Processing Letters, vol. 61, no. 5, pp. 265–269, 1997.

[9] J. W. Sturmballe, "Disjoint paths in a network," Networks, vol. 4, pp. 125–145, 1974.

[10] Y. Chen, X. Guo, and Q. Zeng, "Amr: a multipath routing algorithm based on maximum flow in Ad-Hoc networks," Acta Electronica Sinica, vol. 32, no. 8, pp. 1297–1301, 2004.

[11] X. Fang, S. Shi, and J. Li, "A disjoint multi-path routing algorithm in wireless sensor network," Journal Computer Research and Development, vol. 46, no. 12, pp. 2053–2061, 2009.

[12] S. Omar, M. Zoulkha, and B. Cousin, "Energy efficiency in ad hoc wireless networks with node-disjoint path routing," in Proceedings of the 7th International Workshop on Systems, Signal Processing and their Applications (WoSSPA ’11), pp. 127–130, May 2011.

[13] K. Ishida, Y. Kakuda, and T. Kikuno, "A routing protocol for finding two node-disjoint paths in computer networks," in Proceedings of the International Conference on Network Protocols (ICNP ’95), pp. 340–347, Tokyo, Japan, November 1995.

[14] R. Ogier and N. Shacham, "A distributed algorithm for finding shortest pairs of disjoint paths," in Proceedings of the 8th Annual Joint Conference of the IEEE Computer and Communications Societies. Technology: Emerging or Converging (INFOCOM ’89), vol. 2, pp. 173–182, Ottawa, Canada, April 1989.

[15] Y. O. Lee and A. L. N. Reddy, "Disjoint multi-path routing and failure recovery," in Proceedings of the IEEE International Conference on Communications (ICC ’10), pp. 1–6, IEEE, Cape Town, South Africa, May 2010.

[16] M. Mansouri, H. Snoussi, and C. Richard, "Optimal path selection for quantized target tracking in distributed sensor networks," in Proceeding of the 7th International Wireless Communications and Mobile Computing Conference (IWCMC ’11), 2011.

[17] D. Sidhu, R. Nair, and S. Abdallah, "Finding disjoint paths in networks," in Proceedings of the Conference on Communications Architecture Protocols (SIGCOMM ’91), 1991.

[18] A. Kumar and S. Varma, "Geographic node-disjoint path routing for wireless sensor networks," IEEE Sensors Journal, vol. 10, no. 6, pp. 1138–1139, 2010.

[19] K. Zhang and H. Gao, "Efficient distributed algorithm for correctly finding disjoint paths in wireless sensor networks," International Journal of Sensor Networks, vol. 13, no. 3, pp. 173–184, 2013.

[20] T. Hashiguchi, K. Tajima, Y. Takita, and T. Naito, "Node-disjoint paths search in WDM networks with asymmetric nodes," in Proceedings of the 15th Conference on Optical Network Design and Modeling (ONDM’10), pp. 1–6, Bologna, Italy, February 2011.

[21] A. Bley, "On the complexity of vertex-disjoint length-restricted path problems," Computational Complexity, vol. 12, no. 3-4, pp. 131-149, 2003.

[22] A. Itai, Y. Perl, and Y. Shiloach, "The complexity of finding maximum disjoint paths with length constraints," Networks, vol. 12, no. 3, pp. 277–286, 1982.

[23] D. Ronen and Y. Perl, "Heuristics for finding a maximum number of disjoint bounded paths," Networks, vol. 14, no. 4, pp. 531–544, 1984.

[24] T. Gomes, J. Craveirinha, and L. Jorge, "An effective algorithm for obtaining the minimal cost pair of disjoint paths with dual arc costs," Computers and Operations Research, vol. 36, no. 5, pp. 1670–1682, 2009.

[25] J. Rok, "K-penalty: a novel approach to find—disjoint paths with differentiated path costs," IEEE Communications Letters, vol. 14, no. 4, pp. 354–356, 2010.

[26] R. Leepila, E. Oki, and N. Kishi, "Scheme to find k disjoint paths in multi–cost networks," in Proceedings of the IEEE International Conference on Communications (ICC ’11), pp. 1–5, Kyoto, Japan, June 2011.

[27] H. Chen, B. Liu, X. Lei, F. Ren, and K. Sezaki, "Internode distance-based redundancy reliable transport in underwater sensor networks," EURASIP Journal on Wireless Communications and Networking, vol. 2010, Article ID 358071, 2010.

[28] http://www2.ece.ohio-state.edu/~bibyk/ee582/telosMote.pdf.
Submit your manuscripts at http://www.hindawi.com