Wireless Sensor Network, 2011, 3, 384-390
doi:10.4236/wsn.2011.312045 Published Online December 2011 (http://www.SciRP.org/journal/wsn)

Finding Multiple Length-Bounded Disjoint Paths in Wireless Sensor Networks

Kejia Zhang, Hong Gao
School of Computer Science and Technology, *Harbin Institute of Technology*, *Harbin*, *China*
E-mail: *kejiazhang.hit@gmail.com*, *honggao@hit.edu.cn*
Received September 15, 2011; *revised October* 21, 2011; *accepted November* 24, 2011

Abstract

In a wireless sensor network, routing messages between two nodes *s* and *t* with multiple disjoint paths will increase the throughput, robustness and load balance of the network. The existing researches focus on finding multiple disjoint paths connecting *s* and *t* efficiently, but they do not consider length constraint of the paths. A too long path will be useless because of high latency and high packet loss rate. This paper deals with such a problem: given two nodes *s* and *t* in a sensor network, finding as many as possible disjoint paths connecting *s* and *t* whose lengths are no more than *L*, where *L* is the length bound set by the users. By now, we know that this problem is not only NP hard but also APX complete [1,2], which means that there is no PTAS for this problem. To the best of our knowledge, there is only one heuristic algorithm proposed for this problem [3], and it is not suitable for sensor network because it processes in a centralized way. This paper proposes an efficient distributed algorithm for this problem. By processing in a distributed way, the algorithm is very communication efficient. Simulation results show that our algorithm outperforms the existing algorithm in both aspects of found path number and communication efficiency.

Keywords: Disjoint Paths, Sensor Networks, Length-Bounded Paths

1. Introduction

A typical Wireless Sensor Network (WSN) comprises of thousands of small sensor nodes deployed in the monitoring area. Since communication range of sensor node is very limited, most of data transmissions in WSN need to be forwarded by many intermediate nodes. Usually, sensor nodes are battery-powered and it is unrealistic to replace their batteries, so saving energy is the main objective of WSN design. Researches [4,5] show that the energy consumption of sending and receiving data is several orders of magnitude higher than the energy consumption of computing. Therefore, how to design efficient routing strategies becomes a key issue in WSN. In recent years, routing with multiple disjoint paths has been more and more widely studied [6-13]. For two nodes *s* and *t* in the network, a path connecting *s* and *t* is a *s* - *t* path. A set of *s* - *t* paths are disjoint paths if any two of them do not any common nodes besides *s* and *t*. The problem studied in this paper is:

Problem. In a WSN, given two nodes *s* and *t* and a user-specified length bound *L*, find as many as possible disjoint *s* - *t* paths whose length $\leq L$.

There are several motivations for routing through multiple disjoint paths in WSNs. Routing several data packets through multiple disjoint paths simultaneously will increase the throughput between the given node pair dramatically. Since link and node failures are very common in WSNs, messages can easily be lost while they are routed through a single path. We can increase reliability by sending messages redundantly through multiple disjoint paths. On the other hand, routing messages often through a single fixed path would use up the energy of the nodes in the path quickly. By routing messages through several disjoint paths alternately, the network can gain a better balanced load. Moreover, in some applications, a sensitive message can easily be 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, the task of capturing the whole message would be much more difficult.

It is necessary to limit the length of the paths. If two nodes send messages to each other through a too long path, the transmission latency is likely to be unbearable for the users. In addition, since each wireless link has a

certain amount of packet loss rate, packets have little chance to reach their destination if they are routed through a too long path.

The problem studied by this paper has been proved not only NP-hard but also APX-complete [1,2], which means that there is no Polynomial Time Approximation Scheme (PTAS) for it. To the best of our knowledge, in addition to a heuristic centralized algorithm [3], no other algorithms have been proposed for this problem by now. The algorithm in [3] is not suitable for WSNs because of high communication cost. This paper proposes a distributed algorithm for this problem. Simulation results show that our algorithm outperforms the existing algorithm in both aspects of found path number and communication efficiency.

The problem we consider uses the simple path length (the number of the links in the path) as the metric of the paths. Sometimes, we have to consider a more complicated path metric. For example, we assume that each link has a delivery ratio and use the product of per-link delivery ratios as the metric of a path, which is the delivery ratio of the path. The algorithm proposed in this paper can be easily extended to this situation by setting each edge a weight and using the product of per-edge weights as path metric.

The remainder of the paper is organized as follows: Some related works are introduced in Section 2. After giving preliminary definitions in Section 3, we describe the proposed distributed algorithm in Section 4. Simulation results which confirm the proposed algorithm's efficiency are given in Section 5. Finally, we conclude the paper in Section 6.

2. Related Works

In graph theory, there are several researches that study the problem of finding multiple disjoint s - t paths with length constraint. Reference [1] proves that this problem is NP-hard when $L \geq 5$. Reference [2] further proves that this problem is APX-complete when $L \geq 5$, so there is no PTAS for this problem and it will be very hard to give an approximation algorithm with constant approximation ratio. Therefore, in addition to the heuristic algorithm in [3], no other algorithms are proposed for this problem. The algorithm in [3] is a centralized algorithm which can output optimal solution when $L \leq 4$. If we use it in WSNs, we have to collect the topology information of the whole network to the sink node. The communication cost of such an operation is unbearable for WSNs.

In network area, there are many researches study the problem of finding multiple disjoint s - t paths. References [11,14] propose algorithms to find 2 disjoint s - t paths. References [6,8-10,12,13,15] give efficient distributed algorithms to find k disjoint s - t paths in the given network, where k is a positive integer set by the users. These algorithms have the same basic idea: find a s - t path and delete the nodes in the path, then find the next s - t path and delete its path nodes, until finding k disjoint paths. These works have a common disadvantage: they do not consider the length constraint of the paths. Some paths they found will be not suitable for transmitting data between s and t because of high latency and high packet loss rate.

Some centralized algorithms [7,16-19] are also proposed for finding k disjoint s - t paths. The algorithms in [7,16,19] can find k disjoint paths with minimum total length. However, they can also output some too long paths which are not suitable for routing data. Moreover, since they process in a centralized way, they are not suitable for WSNs.

So far, in addition to the algorithm in [3], all the algorithms for finding disjoint s - t paths do not consider length constraint of paths. The algorithm in [3] is not applicable in WSNs since it is a centralized algorithm. Therefore, this paper propose an efficient distributed algorithm for finding disjoint s - t paths with length constraint. Simulation results show that our algorithm is much better than the algorithm in [3] in both aspects of found path number and communication cost.

3. Preliminaries

We use a graph $G = (V, E)$ to denote the given sensor network, where $V=\{v \mid v$ is a sensor node$\}$ and $E = \{(u, v)$ there is a wireless link between $u \in V$ and $v \in V\}$.

When the algorithm executes, it constructs a tree T_s rooted at s and a tree T_t rooted at t. For each node v in T_s: its parent in T_s is denoted by $parent_s(v)$; its *s-origin* is its ancestor in T_s who is child of s, and we use $origin_s(v)$ to denote it; $dis_s(v)$ is the length of the path from s to v in T_s. For each node v in T_t : its parent is $parent_t(v)$; its *t-origin* is its ancestor in T_t who is child of t, and we use $origin_t(v)$ to denote it; $dis_t(v)$ is the distance from t to v in T_t. Suppose that the algorithm constructs T_s and T_t as shown in **Figure 1**. For node g: $dis_s(g) = 4$, $dis_t(g) = 4$, $origin_s(g) = a$, $origin_t(g) = d$. For node h: $dis_s(h) = 4$, $dis_t(h) = 4$, $origin_s(h) = b$, $origin_t(h) = d$.

If v is a node in both T_s and T_t, we say that v is an *intersecting node* of T_s and T_t. In the example of **Figure 1**, g, h, i, j, k are intersecting nodes of T_s and T_t.

For two nodes u and v in T_s, we use uT_sv to denote the path connecting u and v in T_s. For two paths P_1 and P_2 with a common end node, we use $P_1 + P_2$ to denote their concatenation.

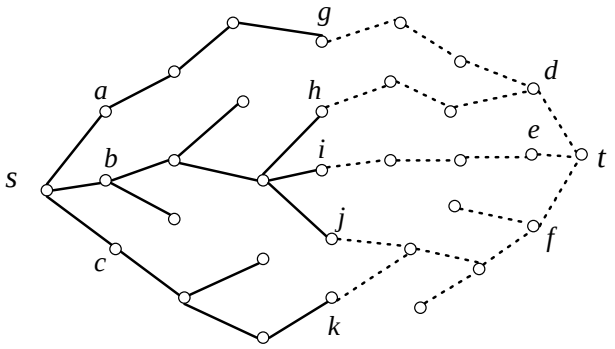

Figure 1. T_s and T_t constructed by the algorithm (the solid lines denote the edges in T_s and the dashed lines denote the edges in T_t).

4. Distributed Algorithm

The proposed algorithm bases on such an observation: In a WSN, whether two nodes can communicate with each other is mainly determined by the physical distance between the two nodes. If the nodes in a WSN distribute evenly, the areas near *s* and *t* are usually the bottle necks of the existence of multiple disjoint *s* - *t* paths, *i.e.*, the number of disjoint *s* - *t* paths is usually constrained by the number of the neighbors of *s* and *t*.

There are 3 steps in the algorithm. In Step 1, we construct trees T_s and T_t rooted at *s* and *t* respectively. The construction of the trees ends at their intersecting nodes. In Step 2, some information of the intersecting nodes is collected to *s*. The information reveals how T_s and T_t intersect with each other. In Step 3, *s* computes an optimal solution with the collected intersecting information. According to the solution, multiple disjoint *s* - *t* paths are built along the two trees.

4.1. Building Trees

In this step, we build trees T_s and T_t rooted at *s* and *t* respectively. The two trees are constructed simultaneously. In the process of tree construction, each node *v* in T_s records the following information: $parent_s(v)$, $origin_s(v)$ and $dis_s(v)$. Similarly, each node *v* in T_t records the following information: $parent_t(v)$, $origin_t(v)$ and $dis_t(v)$.

The construction of T_s/T_t starts at *s*/*t*, then gradually extends outward. Use the construction of T_s as an example: At first, *s* broadcasts an initialization message. Each node receiving the initialization message joins T_s as a child of *s*. Each new node *v* in T_s broadcasts a Build-Tree message including its ID, $origin_s(v)$ and $dis_s(v)$, so the neighbors of *v* can join T_s as *v*'s children.

The construction of T_s ends at: 1) the intersecting nodes of T_s and T_t; 2) the node *v* in T_s such that $dis_s(v) \geq L$. Similarly, the construction of T_t ends at: 1) the intersecting nodes of T_s and T_t; 2) the node *v* in T_t such that $dis_t(v) \geq L$.

For finding more paths, we hope that the trees are built evenly. For instance, if *s* has children *a*, *b*, *c* in T_s, we hope that the sets of the nodes whose *s*-origin is *a*, *b*, *c* respectively have the same size. For this reason, we design a build-tree delaying mechanism. Use the construction of T_s as an example: when node *v* who is not in T_s receives a Build-Tree message for the first time (the sender of the message is *u*), *v* randomly delays for a while to receive more Build-Tree messages rather than joins T_s as *u*'s child immediately. The range of random delay is determined by the current distance from *s* to *v* in T_s, *i.e.*, $dis_s(u) + 1$. The greater the distance is the greater the maximum delay time is. During the delay, *v* records every Build-Tree message it receives. Suppose that *v* receives 4 Build-Tree messages during the delay and the sender of these messages are u_1, u_2, u_3, u_4 respectively. Let the *s*-origins of u_1, u_2, u_3, u_4 be *a*, *b*, *b*, *c* respectively. When the delay ends, *v* randomly chooses one of $\{u_1, u_2, u_3, u_4\}$ as its parent so that $Pr\{origin_s(v) = a\} = Pr\{origin_s(v) = b\} = Pr\{origin_s(v) = c\}$, where $Pr\{origin_s(v) = a\}$ is the probability that *v*'s *s*-origin is *a*.

Algorithm 1: Building T_s

```
1. if v is a neighbor of s then
2.     parent_s(v) = s, origin_s(v) = v, dis_s(v) = 1;
3.     Broadcast Build-Tree message {v, origin_s(v),
       dis_s(v)};
4. else
5.     if v receives a Build-Tree message {u, origin_s(u),
       dis_s(u)} then
6.         Record u as a potential parent;
7.         if it is v's first time to receive a Build-Tree
           message then
8.             Randomly delay for a while according to
               dis_s(u) + 1;
9.         if the delay ends then
               /* Suppose that v' potential parents come from
               n different s-origins a_1, ..., a_n */
10.            Randomly choose a potential parent w so that
               Pr{origin_s(v)=a_1} =...= Pr{origin_s(v) = a_n};
11.            parent_s(v) = w, origin_s(v) = origin_s(w),
               dis_s(v) = dis_s(w) + 1;
12.            if v is not in T_t and dis_s(v) < L then
13.                Broadcast a Build-Tree message
                   {v, origin_s(v), dis_s(v)};
```

The algorithm of building trees is given by Algorithm 1. In Algorithm 1, we only give the pseudo-codes for building T_s. The pseudo-codes for building T_t can be

gotten from it. The whole process starts by s broadcasting an initialization message. The nodes who receive the initialization message join T_s as children of s (Line 1 - 3 in Algorithm 1). If node v receives a Build-Tree message for the first time, it randomly delays for a while according its current distance to s in T_s (Line 7 - 8 in Algorithm 1). During the delay, for each Build-Tree message $\{u, origin_s(u), dis_s(u)\}$ received by v, v records the sender u as its potential parent (Line 5 - 6 in Algorithm 1). Suppose that the potential parents of v come from n different s-origins $a_1, \cdots, a_n$. When the delay ends, v randomly chooses a potential parent as its parent so that $Pr\{origin_s(v) = a_1\} = \cdots = Pr\{origin_s(v) = a_n\}$ (Line 9 - 12 in Algorithm 1). The building of T_s ends at: 1) the intersecting nodes of T_s and T_t; 2) the node v in T_s such that $dis_s(v) \geq L$ (Line 13 - 14 in Algorithm 1).

4.2. Collecting Intersecting Information

At the end of Step 1, every intersecting node v checks whether $dis_s(v)+dis_t(v) \leq L$. If so, we can get a path meeting the length constraint by concatenating path sT_sv and path vT_tt, so v sends the intersecting information $\{origin_s(v), origin_t(v)\}$ to s along T_s. The intersecting information indicates that the path we found goes through s's neighbor $origin_s(v)$ and t's neighbor $origin_t(v)$. When relaying the intersecting information, each ancestor of v records the information and its sender. The whole process of this step is given by Algorithm 2.

Algorithm 2: Collecting Intersecting Information

```
1. for each intersecting node v do
2.     if dis_s(v) + dis_t(v) ≤ L then
3.         Send the intersecting information
           {origin_s(v), origin_t(v)} to parent_s(v);
4. for each node v in T_s do
5.     if receive intersecting information
       {origin_s(v), origin_t(v)} then
6.         Records the information and its sender;
7.         Send the information to parent_s(v);
```

In an example, the users set the length bound as $L = 8$. The algorithm builds T_s and T_t in the given network as shown in **Figure 1**. In Step 2, the intersecting node g sends the intersecting information $\{a, d\}$ to s along T_s. In the same way, h, i, j, k respectively send the intersecting information $\{b, d\}$, $\{b, e\}$, $\{b, f\}$, $\{c, f\}$ to s along T_s.

4.3. Finding Disjoint Paths

Before introducing Step 3, we give a proposition, which can easily be gotten by the nature of T_s and T_t.

Proposition 1. For two intersecting nodes v_1 and v_2, if $origin_s(v_1) \neq origin_s(v_2)$ and $origin_t(v_1) \neq origin_t(v_2)$, then $P_1 = sT_sv_1 + v_1T_tt$ and $P_2 = sT_sv_2 + v_2T_tt$ are two disjoint paths in the network.

Algorithm 3: Finding Disjoint Paths

```
/* Pseudo-codes for s                                  */
1. Builds a graph G* = (V*, E*), where V* = {all the
   neighbors of s and t}, E* = ∅;
2. for each received intersecting information
   {origin_s(v), origin_t(v)} do
3.     E* = E* ∪ {origin_s(v), origin_t(v)};
4. Find the maximum matching in G*;
5. for each edge (origin_s(v), origin_t(v)) in the maximum
   matching do
6.     Send a Build-Path message to v to
       build path;
```

This step is given by Algorithm 3. At first, s builds a bipartite graph G^* with empty edge set (Line 1 in Algorithm 3). One partite set of G^* contains all the neighbors of s. The other partite set of G^* contains all the neighbors of t. If s receives an intersecting information $\{origin_s(v), origin_t(v)\}$, it adds an edge in G^* connecting $origin_s(v)$ and $origin_t(v)$ (Line 2 - 3 in Algorithm 3). According to Proposition 1, we know that each matching in G^* denotes a set of disjoint s - t paths in the network. After receiving all the intersecting information, s searches for the maximum matching in G^* with existing algorithm (Line 4 in Algorithm 3), like the algorithm in [20]. For each edge $(origin_s(v), origin_t(v))$ in the maximum matching, s sends a Build-Path message to v to construct a path from s to t (Line 5-6 in Algorithm 3). The Build-Path message is forwarded to v along T_s. Meanwhile, sT_sv is added to the path. After receiving this message, v sends a Build-Path message to t along T_t. In this process, vT_tt is added to the path.

In the example shown by **Figure 1**, after receiving the intersecting information, s builds a bipartite graph G^* as shown in **Figure 2(a)**. In G^*, s finds the maximum matching as shown in **Figure 2(b)** with existing algorithm. Since g, i, k are the sources of the intersecting information

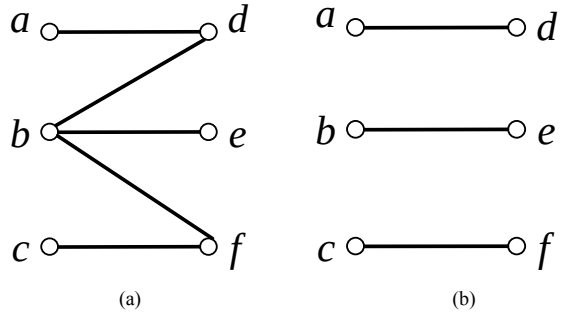

Figure 2. The constructed G^* and the maximum matching in it.

(a, d), (b, e), (c, f), s sends Build-Path messages to g, i, k along T_s. After receiving these messages, g, i, k send Build-Path messages to t along T_t. At the end, 3 disjoint s - t paths as shown in **Figure 3** are built in the network.

5. Simulation Results

We use a simulator written in C++ codes to evaluate the efficiency of the algorithm. In the simulation, we deploy 1000 nodes in a 1000 m × 1000 m area. The locations of the nodes are generated randomly. The transmitting radius of each node is set to 50 m. In MAC layer, we apply CSMA/CA mechanism to avoid signal conflicts. In application layer, the header of each data packet contains 4 bytes to denote the destination's ID, the sender's ID, the length and the type of the packet. Each receiving and timer-fire event is added a time stamp and put into a heap. The events in the heap are executed in the order of their time stamps. In this way, we can simulate the parallel processing of the nodes.

We compare our Efficient Distributed Algorithm (denoted by EDA) with the Heuristic Centralized Algorithm in [13] (denoted by HCA) and a Naive algorithm. The basic idea of the Naive algorithm is: find the shortest s - t path in the network; remove the nodes in the path from the network; find the shortest s - t path in the remaining network and delete its path nodes. The iteration is executed until the length of the found path is greater than L.

The efficiency of an algorithm is measured in two aspects: 1) the number of the paths found by the algorithm; 2) the communication cost of the algorithm, *i.e.*, the average bytes sent and received per node. To get each data in the simulation results, we did 10 groups of experiments. In each group of experiments: the node locations are regenerated; the ID of s and t are 1 and 1000 respectively; we compare the 3 algorithms with the changes of L (L = 10, 20, 30, 40, 50). The simulation results are given by **Figures 4(a)** and **(b)**. Each data in the figures is the average of the results of the 10 groups.

With the changes of L, the numbers of the paths found by the 3 algorithms are as shown in **Figure 4(a)**. We can see that EDA has the best performance among these 3 algorithms in the aspect of found path number. Naive has the poorest performance because it searches for paths in a greedy way and does not consider the possibility of finding multiple paths. EDA finds more paths because it builds tree in a balanced way and adopts an optimal searching strategy.

With the changes of L, the communication costs of the 3 algorithms are as shown in **Figure 4(b)**. We can see that EDA are much better in the aspect of communication cost compared to HCA and Naive. The communication cost of HCA is great and steady because HCA collects the topology information of the whole network no matter what value L is. Naive searches for path repeatedly. In each searching, all the nodes in the network exchange messages to find the shortest path, so the communication cost of Naive rises with finding more paths. Since EDA processes in a distributed way and exchange message once, it is the most efficient one in the 3 algorithms.

Overall, the algorithm EDA is better than the algorithm HCA and the Naïve algorithm in both aspects of found path number and communication cost.

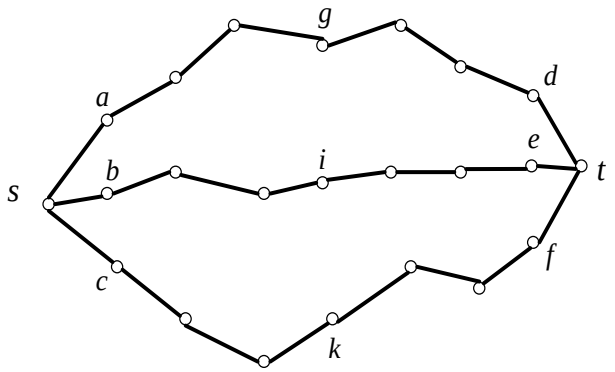

Figure 3. The 3 disjoint paths built by the algorithm.

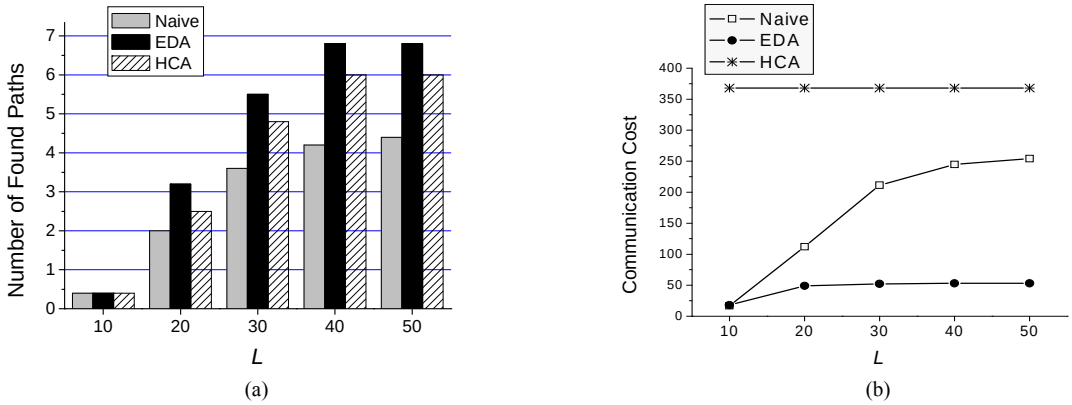

Figure 4. The simulation results.

6. Conclusions

In this paper, we study the following problem: in a sensor network, given two nodes *s* and *t*; a user-specified length bound *L*, find as many as possible disjoint *s* - *t* paths whose lengths are no more than *L*. The existing algorithms for finding multiple disjoint paths rarely consider the length constraint of the paths. For finding multiple length-bounded disjoint paths, there is only one heuristic centralized algorithm. In this paper, we propose an efficient distributed algorithm for this problem. The simulation results show that our algorithm outperforms the existing algorithm in both aspects of found path number and communication cost.

7. Acknowledgements

This work is supported by Key Program of the National Natural Science Foundation of China (Grand No. 61033015), the National Natural Science Foundation of China (Grant No. 60831160525), the National Natural Science Foundation of China (Grant No. 60933001) and the National Natural Science Foundation of China (Grant No. 61100030).

8. References

[1] A. Bley, "On the Complexity of Vertex-Disjoint Length-Restricted Path Problems," *Computational Complexity*, Vol. 12, No. 3, 2003, pp. 131-149. doi:10.1007/s00037-003-0179-6

[2] D. Ronen and Y. Perl, "Heuristics for Finding a Maximum Number of Disjoint Bounded Paths," *Networks*, Vol. 14, No. 4, 1984, pp. 531-544. doi:10.1002/net.3230140405

[3] K. Ishida, Y. Kakuda and T. Kikuno, "A Routing Protocol for Finding Two Node-Disjoint Paths in Computer Networks," *Proceedings of* 1995 *International Conference on Network Protocols*, Tokyo, 7-10 November 1995, pp. 340-347. doi:10.1109/ICNP.1995.524850

[4] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen and M. Welsh, "Simulating the Power Consumption of Large-Scale Sensor Network Applications," *Proceedings of the* 2*nd International Conference on Embedded Networked Sensor Systems*, Baltimore, 3-5 November 2004, pp. 188-200.

[5] P. Gupta and P. Kumar, "The Capacity of Wireless Networks," *IEEE Transactions on Information Theory*, Vol. 46, No. 2, 2000, pp. 388-404. doi:10.1109/18.825799

[6] D. Ganesan, R. Govindan, S. Shenker and D. Estrin, "Highlyresilient, Energy-Efficient Multipath Routing in Wireless Sensor Networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, Vol. 5, No. 4, 2001, pp. 11-25. doi:10.1145/509506.509514

[7] A. Srinivas, G. Theory and E. Modiano, "Minimum Energy Disjoint Path Routing in Wireless Ad-Hoc Networks," *Proceedings of the* 9*th Annual International Conference on Mobile Computing and Networking*, San Diego, 14-19 September 2003, pp. 122-133. doi:10.1145/938985.938999

[8] S. Li and Z. Wu, "Node-Disjoint Parallel Multi-Path Routing in Wireless Sensor Networks," *The* 2*nd International Conference on Embedded Software and Systems*, Xi'an, 16-18 December 2005. p. 6.

[9] J. W. Baek, Y. J. Nam and D. W. Seo, "An Energyefficient k-Disjoint-Path Routing Algorithm for Reliable Wireless Sensor Networks," *Software Technologies for Embedded and Ubiquitous Systems of Lecture Notes in Computer Science*, Vol. 4761, 2007, pp. 399-408. doi:10.1007/978-3-540-75664-4_42

[10] B. Deb, S. Bhatnagar and B. Nath, "Reinform: Reliable Information Forwarding Using Multiple Paths in Sensor Networks," *Proceedings of* 28*th Annual IEEE International Conference on Local Computer Networks*, Bonn/ Königswinter, 20-24 October 2003, pp. 406-415. doi:10.1109/LCN.2003.1243166

[11] R. Ogier, V. Rutenburg and N. Shacham, "Distributed Algorithms for Computing Shortest Pairs of Disjoint Paths, " *IEEE Transactions on Information Theory*, Vol. 39, No. 2, 1993, PP. 443-455.

[12] Y. Chen, X. Guo, Q. Zeng and G. Chen, "AMR: A Multipath Routing Algorithm Based on Maximum Flow in Ad-Hoc Networks," *Acta Electronica Sinica*, Vol. 32, No. 8, 2004, pp. 1297-1301.

[13] X. Fang, S. Shi and J. Li, "A Disjoint Multi-Path Routing Algorithm in Wireless Sensor Network," *Journal of Computer Research and Development*, Vol. 46, No. 12, 2009, pp. 2053-2061.

[14] A. Itai, Y. Perl and Y. Shiloach, "The Complexity of Finding Maximum Disjoint Paths with Length Constraints," *Networks*, Vol. 12, No. 3, 1982, pp. 277-286. doi:10.1002/net.3230120306

[15] D. Sidhu, R. Nair and S. Abdallah, "Finding Disjoint Paths in Networks," *SIGCOMM Computer Communication Review*, Vol. 21, No. 4, 1991, pp. 43-51. doi:10.1145/115994.115998

[16] R. Bhandari, "Optimal Physical Diversity Algorithms and Survivable Networks," *Proceedings of the* 2*nd IEEE Symposium on Computers and Communications*, Alexandria, 1-3 July 1997, pp. 433-441. doi:10.1109/ISCC.1997.616037

[17] S. Khuller and B. Schieber, "Efficient Parallel Algorithms for Testing Connectivity and Finding Disjoint *s* - *t* Paths in Graphs," *Proceedings of the* 30*th Annual Symposium on Foundations of Computer Science*, Research Triangle Park, 30 October - 1 November 1989, pp. 288-293. doi:10.1109/SFCS.1989.63492

[18] K. Iwama, C. Iwamoto and T. Ohsawa, "A Faster Parallel Algorithm for k-Connectivity," *Information Processing Letters*, Vol. 61, No. 5, 1997, pp. 265-269. doi:10.1016/S0020-0190(97)00015-X

[19] J. W. Suurballe, "Disjoint Paths in a Network," *Networks*, Vol. 4, No. 2, 1974, pp. 125-145. doi:10.1002/net.3230040204

[20] J. Edmonds, "Paths, Trees and Flowers," *Canadian Journal of mathematics*, Vol. 17, No. 3, 1965, pp. 449-467.

