Proceedings of APCC2008 copyright © 2008 IEICE 08 SB 0083

Fast Link-Disjoint Path Algorithm on Parallel Reconfigurable Processor DAPDNA-2

Taku KIHARA, Sho SHIMIZU,
Yutaka ARAKAWA, Naoaki YAMANAKA, Kosuke SHIBA
Department of Information and Computer Science, Faculty of Science and Technology,
Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, 223-8522, Japan
Email: kihara@yamanaka.ics.keio.ac.jp

***Abstract*—This paper proposes fast parallel link-disjoint path algorithm using dynamically reconfigurable processor and implements it on DAPDNA-2 (IPFlex Inc) which is newly structured. The conventional *k*-shortest path algorithm finds multiple link-disjoint paths between the source node and the destination node. When the network scale is large, the calculation time of *k*-shortest path algorithm increases rapidly. Moreover, in the worst case, *k*-shortest path algorithm can not find optimum link-disjoint path pair because this algorithm always finds the shortest path at first and removes those links from network. Our proposed algorithm collects all path information in the network and calculates optimum link-disjoint path pair (i.e. minimum cost link-disjoint path pair) at high speed by using parallel operation. Additionally, our proposed algorithm finds optimum link-disjoint path pair at a high rate in a limited of calculation time. The evaluation shows our proposed algorithm can decrease the calculation clock about 90%.**

I. Introduction

In recent year, high-reliable network is emphasized and networks focused on QoS such as next generation network (NGN) attract a lot of attention. There are various technologies that guarantee QoS, and survivability is one of them. In non-survivability network, the interruption which happened by change of network topology or link-down due to disaster causes fatal data loss. Therefore, it becomes an important issue to prepare a backup path for survivability in short time.

A backup path should have the constraint that it can't use the same link as the primary path uses. These path pair which doesn't uses the same link is called the link-disjoint path pair. *K*-shortest path algorithm [1] was widely used to calculate a link-disjoint path pair. However, *k*-shortest path algorithm is not suitable for a large-scale network because it is based on Dijkstra's algorithm [5]. The computation complexity of Dijkstra's algorithm is $O(N^2)$ where the number of nodes is N. So the computational complexity of *k*-shortest path algorithm rapidly increases as the number of nodes increases [2], [3]. Moreover, the *k*-shortest path algorithm doesn't always obtain optimum link-disjoint paths [4]. Therefore, *k*-shortest path algorithm is unsuitable for a large-scale network.

In this paper, we propose a high-speed parallel link-disjoint path search algorithm called "Fast Link-disjoint Path Algorithm (FLDPA)" using dynamically reconfigurable processor and implements it on DAPDNA-2 [7] developed by IPFlex Inc [6]. FLDPA collects all path information which include cost and link information at high speed by simultaneous multiple path search. Then FLDPA calculates optimum link-disjoint path among them. Additionaly, our proposed algorithm collects path information in ascending order of cost, so the probabilty that the algorithm finds optimum link-disjoint path pair is very high even if the calculation time is limited.

The rest of this paper is organized as follows. In section II, we introduce *k*-shortest path algorithm, and explain its problem. In section III we describe the details of FLDPA. Following section IV, we evaluate and compare the calculation clocks of *k*-shortest path algorithm and FLDPA. The conclusions are summerized in section V.

II. *k*-shortest path algorithm

K-shortest path algorithm is widely used in network. It finds *k* link-disjoint paths with *k*-times executions of Dijkstra's algorithm. The outline of the algorithm are shown below.

- Step1) Calculate the shortest path between the source node and the destination node with the Dijkstra's Algorithm.
- Step2) Remove all links which belong to step 1 from the network.
- Step3) Repeat step 1 and step 2 *k*-times, and then *k* link-disjoint paths are found. If we can't execute the Dijkstra's algorithm due to removal the links, the algorithm is finished.

Figure 1 shows the example of *k*-shortest path algorithm. The source node is node A and the destination node is node F. First, the algorithm execute the Dijkstra's algorithm, and it find the shortest path between node A and node F. As a result, the path *A-B-C-D-F*, whose cost is 5 is found. Next, link AB, link BC, link CD, link DF are removed from the network shown in Figure 1. After the removal of links, the Dijkstra's algorithm is executed again. The 1st link-disjoint path *A-E-F* is found. Link AE, link EF are removed again from the network. The *k*-shortest path algorithm repeats above operation until it finds the *k*th link-disjoint path. However in this example, it becomes impossible to find 2nd link-disjoint path because all links connected to node A were removed and then the algorithm is finshed.

In this example, we find a link-disjoint path pair in the network. One is *A-B-C-D-F* and the other is *A-E-F*. Thus, *k*-shortest path algorithm finds *k* link-disjoint paths, with *k*-time executions of Dijkstra's algorithm. However, the *k*-shortest path algorithm is inefficient because the

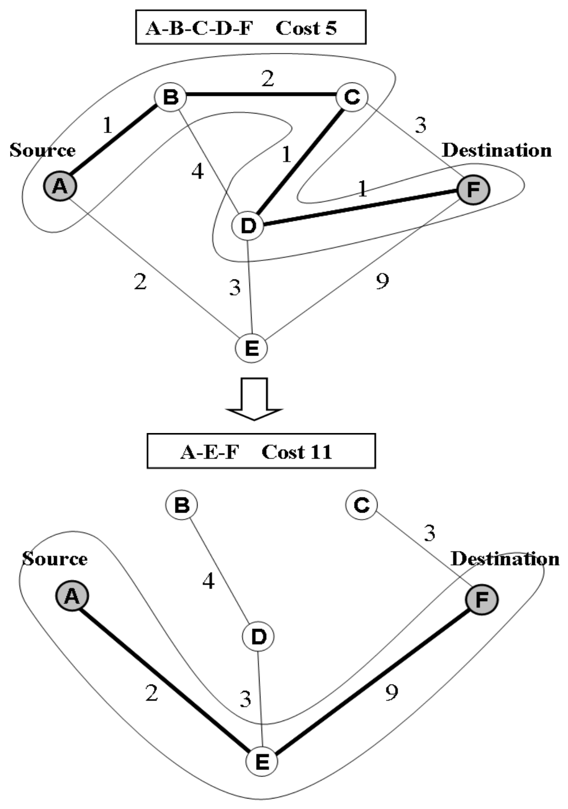


Fig. 1. *k*-shortest path algorithm based on Dijkstra's algorithm

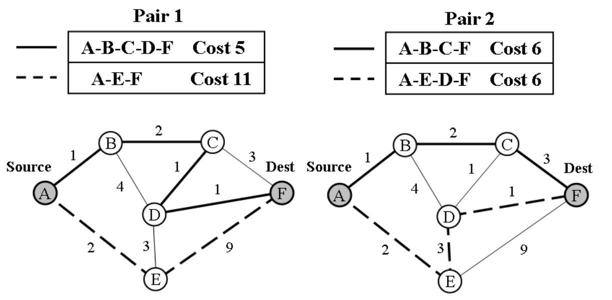


Fig. 2. Case of failure to find link-disjoint path in *k*-shortest path algorithm the optimum solution. Pair 1 shows the solution of *k*-shortest path algorithm and Pair 2 shows optimum solution.

calculation time of Dijkstra's algorithm increases as the value of *k* or the number of nodes increases. Moreover, in Figure 2, better link-disjoint path pair exists, which is lower cost than the pair of example such as *A-B-C-F* and *A-E-D-F*. The *k*-shortest path algorithm never finds this link-disjoint path pair, since it always finds shortest path at first and these links are removed from the network.

III. FLDPA

A. Overview

FLDPA consists of two phases, the candidate path collection phase and the path selection phase. In the candidate path collection phase, FLDPA collects multiple candidate path information which include the link information and the path-cost information at high speed by simultaneous multi-path search. Following the path selection phase, using the path information which collected at the candidate

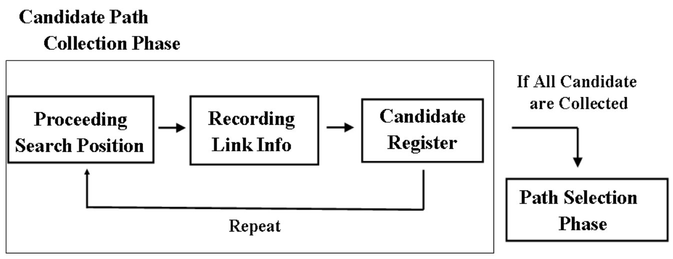

Candidate Path Collection Phase
Proceeding Search Position
Recording Link Info
Candidate Register
Repeat
If All Candidate are Collected
Path Selection Phase

Fig. 3. FLDPA consists of two phases, the candidate path collection phase and the path selection phase.

path collection phase, the optimum link-disjoint paths is calculated. Figure 3 shows the flow chart of FLDPA.

B. Candidate path collection phase

The main idea of this phase is a simultaneous multiple path search from the source node to the destination node. The pace that the algorithm proceeds a current search position toward all connected links is equal to one cost, and every step after proceeding, search results are recorded as path information. After a while, all path information are collected at the destination node, and registered as a candidate path. Here the path information includes the total link cost and which links are used. The path information *p*1(*AB-BC*, 10) means the path passed link AB, BC and its cost was 10.

Figure 4 shows an example of action in the candidate path collection phase. The source node is node A and the destination node is node D. The algorithm collects path information between the source node and the destination node. First, there is two path information *p*1(*AB*, 1) and *p*2(*nothing* , 1) after proceeding operation. Link information of *p*2 is "nothing", since *p*2 still have not passed the link AC. Next, the current search position is proceeded one cost, and the three path information *p*1(*AB*, 2), *p*2(*AC*, 2), *p*3(*AB*, 2), exists. *p*1 and *p*3 seem to the same result, but in fact they are different. *p*1 is on link BD and will make for node *D*, on the other hands, *p*3 is on link BC and will make for node *C*. The current search position is proceeded one cost as well as previous step, there are five path information *p*1(*AB-BD*, 3), *p*2(*AC-CE*, 3), *p*3(*AB-BC*, 3), *p*4(*AC*, 3), *p*5(*AC*, 3). At this time, *p*1 have reached the destination, and *p*1 is registered as a candidate path *c*1(*AB-BD*, 3). Finally, six candidate paths *c*1(*AB-BD*, 3), *c*2(*AC-CE-ED*, 5), *c*3(*AC-CB-BD*, 6), *c*4(*AB-BC-CE-ED*, 6), *c*5(*AC-CD*, 7), *c*6(*AB-BC-CD*, 8) are collected. Note that the candidate paths are sorted in ascending order of cost because the pace of proceeding is "one cost". This characteristic is important for implementation described in section III-D.

In this way, the algorithm repeats the parallel proceeding operation and records multiple path information until all path information are collected.

C. Path selection phase

In the path selection phase, FLDPA calculates the optimum link-disjoint paths from the candidate paths collected at the candidate path collection phase. The algorithm calculates all link-disjoint path pairs among all candidate

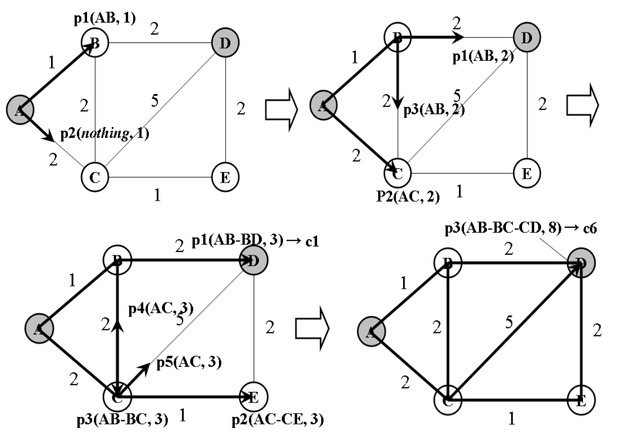
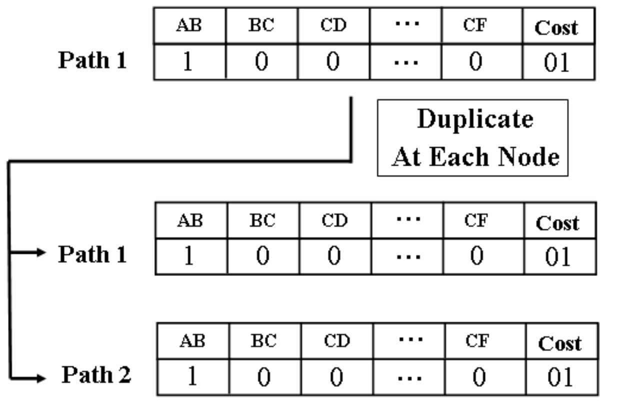


Fig. 4. Multiple path information is collected and recorded in the candidate path collection phase.



	AB	BC	CD	···	CF	Cost
Path 1	1	0	0	···	0	01

Duplicate At Each Node

	AB	BC	CD	···	CF	Cost
Path 1	1	0	0	···	0	01

	AB	BC	CD	···	CF	Cost
Path 2	1	0	0	···	0	01

Fig. 5. The bitmap is duplicated when the current search position reached at the node.

paths from passed link information included in the candidate path information. Then, the algorithm compares their cost and chooses optimum link-disjoint path pair. We explain the concrete procedure in this phase at the following example.

The network topology in this example is shown in Figure 4. Six candidate paths, $c1$, $c2$, ..., $c6$ exist in this network (See III-B). First, the algorithm checks weather the pair $[c1, c2]$ is link-disjoint or not. $c1$ passes Link AB, BD and $c2$ passes link AC, CE, ED. So the pair $[c1, c2]$ is link-disjoint because neither $c1$ nor $c2$ passes the same link. Next, the algorithm checks the pair $[c1, c3]$. The pair $[c1, c3]$ is not link-disjoint, since $c3$ passes link AC, CB, BD and link BD is common link for both candidates. Thus, the algorithm checks all candidate pairs in turn whether link-disjoint or not. As a result, four pairs $[c1, c2]$, $[c1, c5]$, $[c2, c6]$, $[c4, c5]$ are selected. Finally, the minimum-cost pair $[c1, c2]$ is selected among them.

FLDPA can always calculate optimum link-disjoint path pair because FLDPA can list all link-disjoint path pairs.

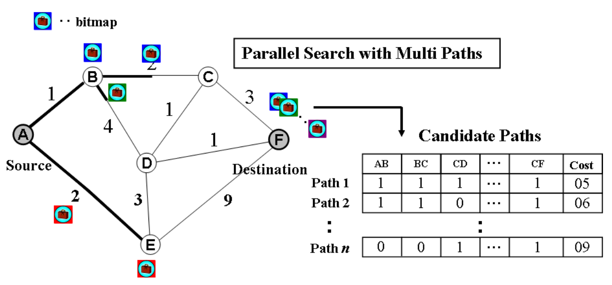
D. Implimentation on DAPDNA-2

In this paper, we implement our proposed algorithm on dynamically reconfigurable processor DAPDNA-2 developed IPFlex Inc. DAPDNA-2 collects much path information and calculates the optimum link-disjoint path pair by parallel operation using bitmaps. A bitmap represents path information recorded in candidate path collection phase, and consists of bit sequence as shown in Figure 5. In the bitmap the least significant bit is total cost of a path, and the other bits represent wheather corresponding path passed that link or not. We describe about the proposed algorithm on DAPDNA-2 below.

1) Candidate path collection phase on DAPDNA-2: We explain the example of candidate path collection phase on DAPDNA-2 with the topology as shown in Figure 1. First, the current search position is set to the source node and the bitmap is stored in memory. Next, DAPDNA-2 proceeds the current search position one step (i.e. one cost) toward the all connected links. If the current search position reaches a new node, the bit corresponding to the passed link is changed from "0" to "1". In this moment, DAPDNA-2 duplicates the current bitmap state equal to the number of branches (i.e. the number of links that gone out of the node). After the duplication, DAPDNA-2 proceeds the current search positions toward the all connected links and update the all bitmaps simultaneously in every one clock. In a path search operation, the path information with the loop is generated due to duplication of bitmaps. To solve this problem, DAPDNA-2 discards the bitmaps that the bit of the same place is changed twice because a loop path passes the same link more than twice. This manner prevents a loop path being collected. If the bit corresponding to the destination link (i.e. the link connected to the destination node) is changed to "1", the bitmap is registered as a candidate path. For example, a bitmap of candidate paths as shown in Figure 6 represents that the candidate path 1 is a 5-cost path which passes link AB, BC, CD, DF, and the candidate path 2 is a 6-cost path which passes link AB, BC, CF. In this way, DAPDNA-2 can collect multiple candidate path information in parallel. Finally, the candidate path collection phase ends when the determined numbers of candidate paths are collected or the determined number of clocks passed without collecting sufficient candidate paths.

In the candidate path collection phase, the cost information of the collected candidate paths is equal to the total clocks since FLDPA began to this phase, because current search positions are updated every one clock and all bitmaps are recorded every one clock too. Therefore, DAPDNA-2 can easily limit to collect candidate paths which are less likely to be selected for optimum link-disjoint paths because the all candidate paths can be collected at destination node in ascending order of cost.

2) Limit to collect path information: In the candidate path collection phase, the bitmap is duplicated frequently with time passage, therefore, DAPDNA-2 must keep up to $N!$ bitmaps in worst case. To solve this problem, DAPDNA-2 limits the number of collected candidate path to α_{limit} at all nodes. For instance, when α_{limit} is set to 30, all bitmaps which reached each node the 31st are discarded. This manner can limit the total number of bitmaps which DAPDNA-2 finally keeps to $N \cdot \alpha_{limit}$. Though the effect of the limitation is very large, the case where the link-disjoint path cannot be found may exists due to value of α_{limit} is small. Therefore, it is necessary



	AB	BC	CD	⋯	CF	Cost
Path 1	1	1	1	⋯	1	05
Path 2	1	1	0	⋯	1	06
:			:			
Path *n*	0	0	1	⋯	1	09

Fig. 6. DAPDNA-2 registers a bitmap which passed destination link as candidate path.

to set the best value of α_{limit} according to the network topology or the number of node in network.

3) Path selection phase on DAPDNA-2: FLDPA starts the path selection phase when the determined number of candidate paths are collected. In the path selection phase, DAPDNA-2 calculates all link-disjoint path pairs from all candidate paths using bit-based operation, and calculates the best pair which is the lowest cost among them. The example of path selection is shown below.

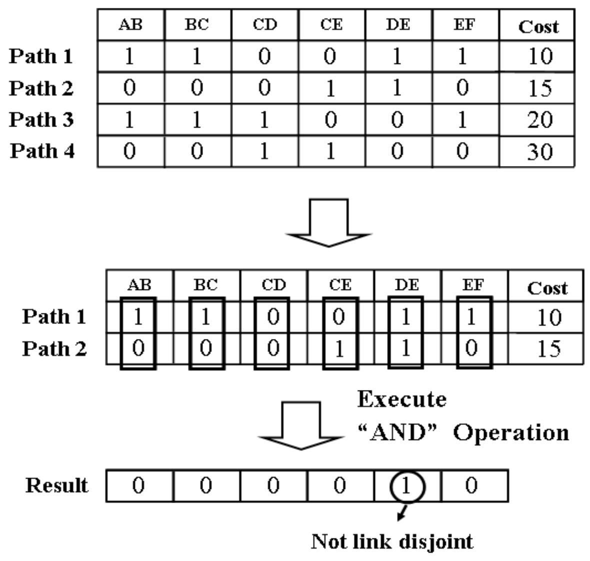
Figure 7 shows that DAPDNA-2 calculates the link-disjoint path pair from four candidate paths. At first, DAPDNA-2 searches the link-disjoint path pair of path1 from the all candidate paths. DAPDNA-2 checks whether the pair of path 1 and path 2 is link-disjoint or not. DAPDNA-2 executes AND operation between bitmap of path 1 and bitmap of path 2. As a result, the bit of link CE is 1. This means the path 1 and the path 2 passed common link CE. Therefore both paths are not link-disjoint. Thus, in AND operation results of each bit in the bitmaps, if the bit changes to 1 even one place, the pair of path is not link-disjoint. On the contrary, when the bit doesn't stand any place, the pair of path is exactly link-disjoint. DAPDNA-2 repeats AND operation for all path pairs such as path 1-path 3, path 1-path 4, path 2-path 3...etc, and all link-disjoint path pairs can be calculated. Finally, the pair of paths with the minimum total cost among them is selected as the optimum link-disjoint path pair.

IV. Performance evaluation

In this section, we evaluate the calculation clocks of *k*-shortest path algorithm and FLDPA.

A. Calculation clocks

In FLDPA, it takes 50 clocks to execute the dataflow which shown in Figure 3. When the number of collected candidate paths is set to α_{limit}, FLDPA executes until α_{limit} paths between the source node and the destination node are collected. The clocks that α_{limit} paths are collected equal to the clocks that $(\alpha_{limit} - 1)$ th higher cost path counts from the shortest path. Now, we consider square-mesh network whose link-cost is L. The execution clocks that FLDPA collects shortest path is $50 \cdot L \cdot 2(\sqrt{N} - 1)$, when the number of nodes is N. Moreover, the path which is a $\alpha_{limit} - 1$ th higher cost than shortest path takes more L clock in the worst case. In addition, FLDPA needs AND operation at the path selection which takes three clocks, so total clocks



	AB	BC	CD	CE	DE	EF	Cost
Path 1	1	1	0	0	1	1	10
Path 2	0	0	0	1	1	0	15
Path 3	1	1	1	0	0	1	20
Path 4	0	0	1	1	0	0	30

	AB	BC	CD	CE	DE	EF	Cost
Path 1	1	1	0	0	1	1	10
Path 2	0	0	0	1	1	0	15

Result	0	0	0	0	1	0

Fig. 7. In path selection phase, DAPDNA-2 calculates optimum link-disjoint path pair using bit-based operation.

to select the k link-disjoint paths is $3 \cdot_{\alpha\ limit} C_k$. Therefore, total calculation clocks of FLDPA is $50 \cdot L \cdot 2(\sqrt{N} - 1) + 2L(\alpha_{limit} - 1) + 3 \cdot_{\alpha\ limit} C_k$.

On the other hand, the calculation clocks are proportional to the second power of the number of nodes, since the *k*-shortest path algorithm calculates based on the Dijkstra's algorithm. So total calculation clocks of *k*-shortest path algorithm is $\beta \cdot k \cdot N^2$. Here, β is a constant coefficient which execute the Dijkstra's algorithm.

B. Performance comparison

Figure 8 shows the calculation clocks that FLDPA and the *k*-shortest path algorithm find a link-disjoint path pair ($k = 2$) in mesh topology ($L = 10$). We find that the calculation clocks of FLDPA is far lower than the *k*-shortest path algorithm when N is large. For instance, in 196-node network, FLDPA decreases calculation clocks about 90% compared with the *k*-shortest path algorithm. This reason is that FLDPA increase calculation clocks very gradually although N increase by parallel processing, on the other hand, the *k*-shortest path algorithm based on the Dijkstra's algorithm increases calculation clocks rapidly as N increase. Moreover, FLDPA has less influence in α_{limit} and can find the best link-disjoint path pair at high speed compared with the *k*-shortest path algorithm because FLDPA is able to collect much candidate paths in a large-scale network.

V. Conclusion

In this paper, we have proposed fast new parallel link-disjoint path search algorithm. It is suitable for dynamically reconfigurable processor like DAPDNA-2. In the *k*-shortest path algorithm, the computation complexity of calculation process increases rapidly as the number of nodes increases. Moreover, it is difficult for *k*-shortest path algorithm to find the optimum link-disjoint path since *k*-shortest path algorithm always finds the shortest path at first and removes these links from the network. On the

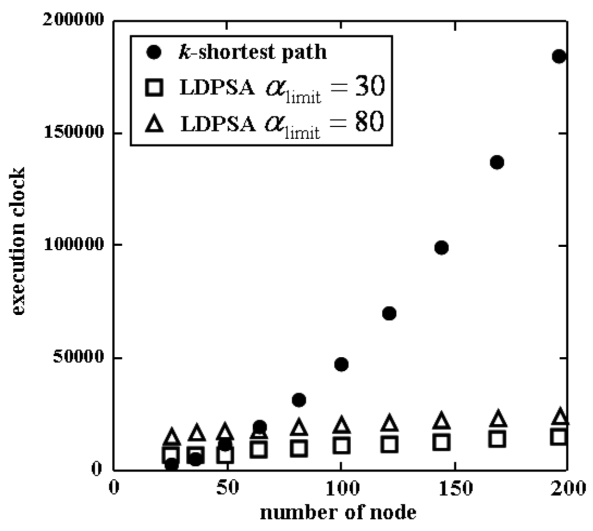


Fig. 8. Number of calculation clocks dependence of number of nodes in the network

contrary, our proposed algorithm can collect all path information in short time by simultaneous multi-path search using DAPDNA-2. Therefore, FLDPA can find the optimum link-disjoint path pair at high speed compared with the *k*-shortest path algorithm. In performance evaluation, We show that the calculation clocks of FLDPA decreased greatly compared with the *k*-shortest path algorithm when the network is large. Similarly, we show that FLDPA's calculation clocks by increase of α_{limit} increased very gradual, and FLDPA can take a lot of candidate paths. We show that FLDPA is very efficient algorithm for link-disjoint path selection in a future network.

Acknowledgment

The authors thank Tomomi Sato and other staff for helping with implementation on DAPDNA-2 (IPFlex Inc). This work was supported by Global COE Program "High-Level Global Cooperation for Leading-Edge Platform on Access Spaces (C12)" and by the Japan Society for the Promotion of Science's (JSPS) Grant-in-aid for Scientific Research.

References

[1] Takeshi TOMOCHIKA, Yuichi IKEJIRI, Tomoaki KOBAYAKAWA, Internet Routing, Shoeisha, Tokyo, 2005.
[2] Dahai Xu, Member, IEEE, Yang Chen, Student Member, IEEE, Yizhi Xiong, Chunming Qiao, Member, IEEE, and Xin He, Member, IEEE, "On the Complexity of and Algorithms for Finding the Shortest Path With a Disjoint Counterpart", *IEEEACM TRANSACTIONS ON NETWORKING*, Vol. 14, No. 1, Feb. 2006.
[3] A. Sen, B. Shen, S. Bandyopadhyay, and J. Capone, "Survivability of lightwave networks path lengths in WDM protection scheme", *Jernal of High Speed Network*, vol. 10, no, 4, pp. 303-315, 2001.
[4] Sudip Misra, Member, IEEE, and B. John Oommen, Fellow, IEEE, "An Efficient Dynamic Algorithm for Maintaining All-Pairs Shortest Paths in Stochastic Networks", *IEEE TRANSACTIONS ON COMPUTERS*, Vol. 55, No. 6, Jun. 2006.
[5] E.W.Dijkstra's, "A note on two problems in connection with graphs," Numerische Mathematik, vol.1, pp.269-271, 1959.
[6] IPFlex Inc (http://www.ipflex.com)
[7] Toshinori SUEYOSHI, Hideharu AMANO, Reconfigurable system, Ohmsha, Tokyo, 2005.