개선된 분산 Delay-Constrained Unicast Routing 알고리듬

周曉正*, 서희종
여수대학교 전자통신전기공학부
*북경석유화공학원 정보공학과

An Improved Distributed Algorithm for Delay-Constrained Unicast Routing

Xiaozheng Zhou*, Heejong Suh
Division of Electronic Communication and Electrical Engineering
Yosu National University
*Department of Information Engineering
Beijing Institstute of Petro-Chemical Technology
E-mail : *zhouxiaozheng@bipt.edu.cn, hjsuh@yosu.ac.kr

Abstract

In this paper, we propose an improved delay-constrained unicast routing (I-DCUR) algorithm for real-time networks which is based on the delay-constrained unicast routing (DCUR) algorithm. Our I-DCUR algorithm is quite different from DCUR algorithm, because the node will choose the link between the active node and the previous node, and it will replace the original loop path when it detects a loop. Thus, firstly consider to choose the link between the active node and the previous node to replace the original loop path when a node detects a loop. So our algorithm can make the construction of path more efficiently, as compared to DCUR algorithm. We could see that the performance of I-DCUR algorithm is much better than DCUR algorithm in the experimental results. There were over 40% improvement in 100 nodes, 60% in 200 nodes, and 9% reduction of costs.

I. Introduction

Both Bellman-Ford's and Dijkstra's SP algorithms are exact and run in polynomial time. As the name indicates, an SP algorithm minimizes the sum of the lengths of the individual links on the path from source to destination. If the length of a link is used to measure the delay on that link, then an SP algorithm computes the least-delay (LD) path, and if the link length is set equal to the link cost, then an SP algorithm computes the least-cost (LC) path. Douglas. S. Reeves et al. study the problem of unicast routing of real-time traffic subject to an end-to-end delay constraint in connection-oriented networks [1]. They formulate the problem as a Delay-Constrained Least-Cost (DCLC) path problem which is NP-hard. Therefore, they propose a distributed heuristic solution: the delay-constrained unicast routing (DCUR) algorithm.

In the DCUR algorithm, the active node chooses either LC path or LD path. When a loop is detected, the active node sends a *remove- loop* message traversing the loop backwards to delete all links between the active node and the node whose routing table entry's flag is set to LCPATH, and find another path at this node towards the destination.

But, when detecting a loop, if a node will choose the link between the active node and the previous node to replace the original loop path, the algorithm will be improved. To do this, we shall propose an improved DCUR algorithm.

This paper is organized as follows: In Section II, we propose our improved algorithm based on DCUR algorithm. Finally, the conclusion is presented in Section III and, the next is the references.

II. Improved DCUR algorithm (I-DCUR)

From the DCUR algorithm, we can see that, in DCUR, the paths constructed by existing distance-vector protocols are guaranteed to be loop-free if the contents of the distance vectors at all nodes are up-to-date and the network is in stable condition. However, up-to-date cost vectors and delay vectors contents and stable network condition are not sufficient to guarantee loop-free operation for DCUR. In DCUR, each node involved in the path construction operation selects either the LC path direction or the LD path direction, as has been explained above. If all nodes choose the LC path direction, or all nodes choose the LD path direction, then no loops can occur because the resulting paths are the LC path or LD path respectively. However, if some nodes choose the LD path direction while others choose the LC path direction, loops may occur. In the following section, we will discuss how DCUR detects and eliminates loops, and then propose our improved algorithm. Fig. 1(a)-(c) show a scenario which results in a loop.

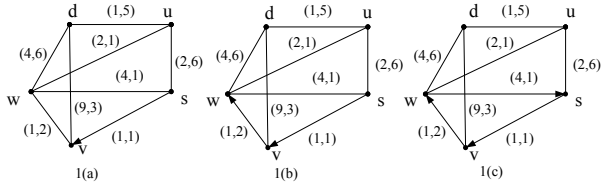

Fig. 1. Example of a loop scenario. s is the source and d the destination. Link costs and link delay are shown next to each link as (cost, delay), σ =8.

The source node *s* initiates the construction of a path towards the destination node *d* with an imposed delay constraint value of 8. Fig. 1(a), 1(b), and 1(c) show successive stages of path construction until a loop is created. The source *s* looks up its LD vector and LC vector, apparently the LC path is *s*→*u*→*d*, but the total delay on the path is equal to 11, which is greater than the constraint value 8. Then the source *s* follows the LD path direction towards the destination *d* (*s*→*v*→*d*), and link (*s, v*) becomes the first link in the path. Node *v* follows the LC path direction towards *d* (*v*→*w*→*d*) and adds link (*v, w*) to the path. Node *w* follows the LD path direction (*w*→*s*→*v*→*d)* and adds link (*w, s*) to the path. This creates a loop (*s*→*v*→*w*→*s*), as shown in Fig. 1(c). When node *s* receives a *construct_path* message, it searches its routing table. Because a *construct_path* message includes the ID of the source, and the ID of the destination, node s can detect the loop if a routing table entry already exists for the source_destination pair specified in the *construct_path* message. It reacts by sending a *remove_loop* message that traverses the loop backwards. Node *w* receives the *remove_loop* message from node *s*, but node *w* is already following the LD path direction towards the destination, so all it does is to send the *remove_loop* message further backwards to *v*, and delete its routing table entry, thereby removing link (*s, w*) from the path(Fig. 2(a)). Node *v* receives the *remove_loop* message. It is following the LC path direction towards the destination, so it decides to follow the LD path direction instead, and modifies its routing table entry accordingly, thus removing link (*v, w*) from the path(Fig. 2(b)). At last, node *v* follows the LD path and adds link (*v, d*) to the path. The final delay_constrained path from *s* to *d* is the one shown in Fig. 2(c). we can imagine that this removing process must prolong the construction path time, especially for large size networks. But when the networks become larger, the loops might include more links than the smaller networks. So, this is the main drawback of DCUR algorithm.

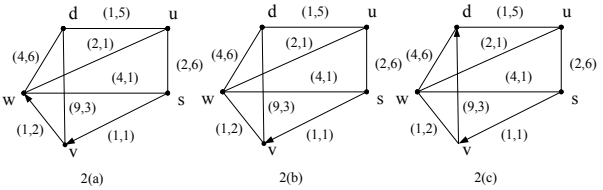

Fig. 2. The process of DCUR removing a loop and reconstructing a new path.

Our improved algorithm is different from DCUR mainly in the case when a loop occurs. It is as follows:

When any node *y* receives a *construct_path* message from node *x*, and becomes current *active_ node*, and node *x* becomes *previous_ active_node*. If no loop is detected at node *y*, it executes the same procedure as described above. If, however, a loop is detected, node *y* firstly looks for its *least_delay* vector to read the delay value between node *y* and *x* which sent the *construct_path* message. In DCUR procedure described above, the *previous_active_node* follows the LD path direction to add

link(x, y) to the path. It is easy to certify that link(x, y) must be the LD path between node x and node y, too. So node y must have kept the delay value between x and node y, D(x, y), in its LD path vector. Then node y sends back a *find_loop* message that contains node y's *delay_so_far* (y) and D(x, y). After receiving the *find _loop* message, node x reads the ID of the next hop node on the LC path towards d, *lc_nhop*, from its cost vector. Then x sends a Query message to *lc_nhop*, requesting the LD value from *lc_nhop* to d. *lc_nhop* looks up the requested value, *ld_ value*(*lc_nhop*, d),from its delay vector, and sends a Response message back to node x with this information. After node x receives the Response message, it checks if

$$delay\_so\_far(y) + D(x,y) + D(x, lc\_nhop) + ld\_value(lc\_nhop, d) \le \sigma \quad (1)$$

If the inequality is satisfied, there exist delay_constrained paths from source s to destination d which use the path(s, y), the link(y, x), and the link(x, *lc_nhop*). Then x sends a *reconstruct_path* message to *previous_ node, y,* to inform that the path from $s \rightarrow d$ following link(x, y) is available. At the same time, both x and y will modify their routing entry made before with following information:

At node x

- the ID of s,
- the ID of d,
- *previous_node*=the ID of node y, which sent the *find_loop* message to it
- *next_node*= *lc_nhop*
- *previous_ delay*= *delay_ so_far(y)+D(x, y)*
- flag=*LCPATH*

At node y

- the ID of s,
- the ID of d,
- *previous_node*=not changed
- *next_node*= node x
- *previous_ delay*=not changed
- flag=*NEITHER_LCPATH_NOR_LDPATH*

For any node, when its routing table entry's flag is set to *NEITHER_LCPATH_NOR_LDPATH,* this indicates that the node has detected a loop and after removing the loop the node has chosen neither the LC path nor LD path, but the another proper path.

After modifying its routing entry, node x sends a *construct_path* message to *next_node*, the following procedure is the same as DCUR.

If inequality (1) is not satisfied, node x executes the same procedure when a loop is found in DCUR algorithm.

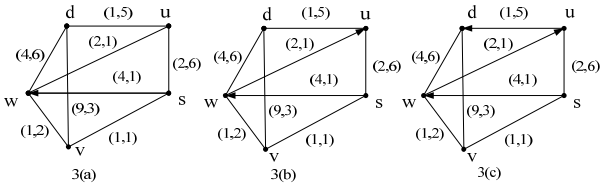

Fig. 3. The process of I-DCUR removing a loop and reconstructing a new path.

In Fig. 3, when node s detects a loop, it will look into its *least_delay* vector to read the delay value between node s and w, D(w, s), in its LD path vector. Then node s sends back a *find_loop* message that contains the *delay_so_far(s)* and D(w, s). After receiving the *find _loop* message, node w reads the ID of the next hop node on the LC path towards d ($w \rightarrow u \rightarrow d$), *lc_nhop*, from its cost vector. Then w sends a Query message to *lc_nhop*, u, requesting the LD value from *lc_nhop* to d. *lc_nhop* looks up the requested value, *ld_ value*(*lc_nhop*, d),from its delay vector, and sends a response message back to node w with this information. Since the total delay from s to d following the path P($s \rightarrow w \rightarrow u \rightarrow d$) is

$$\begin{aligned} &delay\_so\_far(s) + D(w,s) + D(w, lc\_nhop) + ld\_value(lc\_nhop, d) \\ &= delay\_so\_far(s) + D(w,s) + D(w,u) + ld\_value(u,d) \\ &= 7 \le 8 \end{aligned} \quad (2)$$

Therefore, node w follows the LC path direction and adds link (w, u) to the path. At last, node u adds link (u, d) to the path. Here is the link (u, d) is both the LC and LD path directions. Fig. 3(a) ~ 3(c) show the path constructed by I-DCUR algorithm. We can see that, although both paths constructed by DCUR and I-DCUR satisfy the delay constraint, the total cost of the two paths are different. For the path constructed by DCUR the total cost is 10, while it is 7 for the path made by I-DCUR.

III. Simulation results

To evaluate the performance of I-DCUR, We made simulations to compare I-DUCR with DCUR algorithm. To do this, we setup the networks according to doctor Salama's method [1]. In the first experiment, we measured the average proportion of loops which are handled by I-DCUR algorithm in the total occurred loops (e.g., the total occurred loops equal to the sum of loops handled by I-DCUR and DCUR respectively). 1000 successful runs of DCUR or I-DCUR were simulated for each delay constraint in Fig. 4.

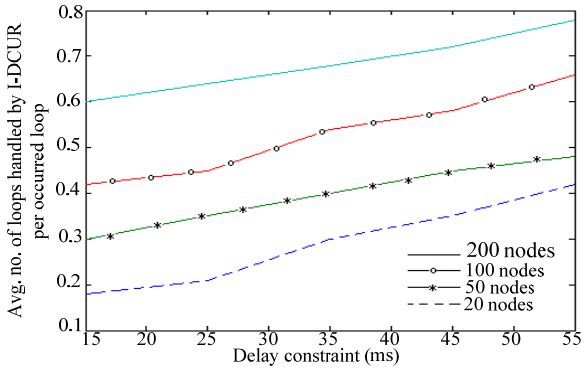

Fig. 4. Average proportion of loops which are handled by I-DCUR algorithm in the total occurred loops.

The second experiment shows how we can measure the average cost of the paths from a source node to all the reachable destination nodes, and prepare I-DCUR with DCUR. To do this experiment, we built 100 random networks with 40 nodes and an average node degree of 4. The results of the experiments have been averaged. The topology of the networks was generated by a Random Graph Generator program [2]. The cost value on links varies from 1 to 15 based on uniform distribution. The propagation delay on links is selected from three ranges to resemble delay characteristics of a nationwide network e.g. in the US. The first range (1-5 ms) represents short local links, the second range (5-8 ms) represents longer local links, while the third range (20-30 ms) represents continental links. The ratio of long local links was configured to be 20 percent, while the ratio of continental connections in networks was configured to be 5 percent. In the simulations, we changed the delay constraint σ and investigated its effect on the found paths of different algorithms.

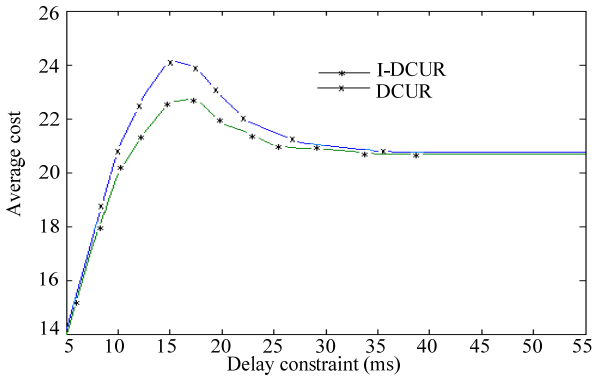

Fig. 5. Average cost.

IV. Conclusion

Obviously, I-DCUR can provide more chances in choosing a path, and can make path construction more efficient. From Fig. 1(c),1(d),1(e), according to DCUR, when node *s* finds a loop, the *remove−loop* message will traverse the loop backwards to node v to remove link(*s*, *w*), link(*w*, *v*) from the path, and adds link(*v*, *d*) to the path. The final path constructed is *s*→*v*→*d*, and the total cost is 10. While with our I-DCUR, the previous node, *w,* will first check whether link(*s*, *w*) can replace the another path(*s*→*v*→*w*). In our example, the link(*s*, *w*) is added to the path, and the path (*s*→*v*→*w)* is deleted, and the final path from source node s to destination node is (*s*→*w*→*u*→*d*) which total cost is 6.

References

[1] Douglas. S. Reeves and Hussein F. Salama, "A distributed algorithm for Delay-Constrained unicast routing," *IEEE Trans. Networking*, vol.8, pp.239-250, Apr. 2000.

[2] Bal´azs G´abor J´ozsa and D´aniel Orincsay, "Random Graph Generator (for telecommunication networks)," Technical Report, April 1999.