DBpia

論 文

A Reliable Distributed Shortest Path Routing Algorithm for Computer Networks

Sung Woo Park*, Young Chon Kim** *Regular Members*

컴퓨터 네트워크를 위한 신뢰성 있는 분산 최단경로 설정 알고리즘

正會員 朴 聖 宇* 正會員 金 永 川**

요 약

대부분의 컴퓨터 네트워크에서, 각 교환 노드는 데이타 패킷 전송시 최단 경로를 찾기 위해 정확한 경로 정보를 갖는 것이 필요하다. 그러나, 분산화된 환경하에서는 전체 네트워크를 통해 항상 일관성 있는 경로 정보를 유지하기가 어렵다. 따라서, 시간이 지남에 따라 이미 쓸모없게 된 경로 정보로 인하여 경로상의 루우프가 형성될 수 있으며, 이로 인하여 전체 네트워크의 심각한 성능 저하를 초래할 수도 있다.

본 논문에서는 이러한 경로상의 루우프 형성 문제를 해결하기 위한 새로운 경로 설정 알고리즘에 대해 논하고자 한다. 제안된 알고리즘은 현재 컴퓨터 네트워크에서 경로 설정을 위해 가장 많이 이용되고 있는 분산화된 Bellman-Ford 알고리즘에 근거하고 있다. 제안되는 알고리즘은 Bellman-Ford 알고리즘의 간편성을 유지하면서 분산화된 환경하에서 (두 노드간 또는 여러 노드들간의) 모든 종류의 루우프를 일정 시간안에 발견하고 이를 해결한다.

Abstract

In most computer networks, each node needs to have correct routing information for finding shortest paths to forward data packets. In a distributed environment, however, it is very difficult to keep consistent routing information throughout the whole network at all times. The presence of out-dated routing information can cause loop-forming which in turn causes the significant degradation of network performance.

In this paper, a new class of routing algorithm for loop detection and resolution is discussed. The proposed algorithm is based on the distributed Bellman-Ford algorithm which is popularly adopted for routing in computer networks. The proposed algorithm detects and resolves all kinds(two-node and multi-node) of loop in a dstributed environment within finite time while maintaining the simplicity of the distributed Bellman-Ford algorithm.

*Dept. of Information and Communication Eng. HanNam University
한남대학교 정보통신공학과
**Dept. of Computer Eng. ChonBuk National University
전북대학교 컴퓨터공학과

論文番號 : 94-3

www.dbpia.co.kr

I. Introduction

The routing algorithm of the original ARPA-NET was developed on the basis of Bellman-Ford algorithm[1]. In the original ARPANET, the shortest paths are computed for every node to every other node in a distributed manner. Due to the simplicity and the robustness of the distributed Bellman-Ford algorithm, many of other computer networks adapted routing algorithms similar to the original ARPANET routing algorithm. Examples of such networks include TIDAS [2], Datapac[3], DECnet[4], etc. During the operating years (1969−1978) the original ARPANET routing algorithm had been known to have some deficiencies: poor adaptability to network changes and unnecessary looping caused by link/node failures, etc. The new ARPANET routing algorithm[5] takes a totally different approach based on Dijkstra algorithm[6]. Complete topological information about a network is maintained at every node, and the routing protocol is supported for broadcasting changes in topology throughout the network. In large-scale computer networks, however, it is prohibitive to maintain complete topological information at each node and broadcast every topological change throughout the entire network[7].

There have been continuous efforts to reduce or eliminate looping problems of the distributed Bellman-Ford algorithm. Merlin and Segall[8][9] first proposed loop-free algorithms which, like the original ARPANET, maintain only local routing information as opposed to managing global topology at each node. Jaffe and Moss[10] proposed another loop-free algorithm similar to Merlin-Segall algorithm but superior in terms of recovery speed. With Jaffe-Moss algorithm, loops can be completely avoided during update periods. However, it enforces a protocol which can cause long respone time and congestions due to topology changes. The protocol also requires updat messages to arrive in strict order(FIFO).

Some alternatives have been proposed to reduce looping effects. For example, TIDAS and Datapace networks operate the so-called *splithorizon* algorithm. When a node passes update messages to one of its neighbors, it replaces the shortest path lengths through that neighbor by the second ones. Schwartz also proposed the *predecessor* algorithm[11]. In the predecessor algorithm, each node identifies its predecessor on the shortest path to a destination. This can be done by sending a special update message to the current route successor on the shortest path. These loop-detection algorithms maintain the simplicity of the distributed Bellman-Ford algorithm, but they only prevent two-node loops from forming and cannot detect multi-node loops. It is known that multi-node loops usually degrade network performance more seriously than two-node loops.

In this paper, we propose a new routing algorithm which is the modified version of the distributed Bellman-Ford algorithm. The proposed algorithm is based on the same idea as in [12], but it uses a different scheme for loop detection, preserving the distributed nature of Bellman-Ford algorithm. In the proposed algorithm, a node locally constructs the shortest path trees(SPTs) rooted at each neighbor using update messages and checks the existence of looping for each concerned path. An update message is modified to include new information needed to construct the STPs. The proposed algorithm detects (resolves) any kinds of loop within finite time and requires relatively small amount of additional memory and computation time.

The rest of this paper proceeds as follows. The distributed Bellman-Ford algorithm and the associated looping problem are discussed in Section 2. Section 3 presents the proposed algorithm and its improved versions. The proposed algorithm is analyzed and compared with other competent algorithms in Section 4. Finally, Section 5 summarizes this paper.

II. Problem Formulation

www.dbpia.co.kr

2.1 Distributed Bellman-Ford Algorithm

In the distributed Bellman-Ford algorithm, update messages carry routing information among nodes. Let us denote the minimum distance from node j to node d by $D_i^k(j, d)$ at an iteration step k. The subscript i indicates the location where routing decision is made. An update message sent out from node j for destination d after an iteration step k includes the identity of destination and the minimum distance from node j to destination d:

$$\{d, D_j^k(j,d)\}$$

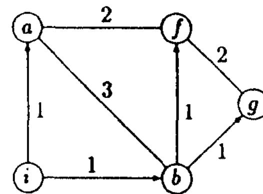
Update messages are initiated by a node when an adjacent out-going link lenght changes and are sent out to all neighbors.

Let $l_i^k(j)$ denote an out going link length from node i to node $j \in N(j)$ where $N(i)$ is the set of neighbors adjacent to node i, and $S_i^k(d)$ is the route successor on the shortest path from node i to destination d at an iteration step k. Node i now uses the latest estimate $D_j^k(j,d)$ sent from neighbor j and $l_i^k(j)$ for deciding the minimum distance from node i to destination d. Then node i broadcasts the latest estimate $D_j^k(i,d)$ to all neighbors if it is different from the previous one ($D_i^k(i,d) \neq D_i^{k-1}(i,d)$). The update protocol performed at node i for destination d is as follows:

Distance Update(d):

if (node i receives $\{d, D_j^{k-1}(j,d)\}$ from neighbor j)
  $D_i^k(j,d) \leftarrow D_j^{k-1}(j,d)$;
if (node i detects $l_i^k(j) \neq l_i^{k-1}(j)$)
  $l_i^k(j) \leftarrow l_i^{k-1}(j)$.
Route Update(d):
  $D_i^k(i,d) \leftarrow \min_{j \in N(i)}\{D_i^k(j,d) + l_i^k(j)\}$;
  with $p \in N(i)$ chosen,
  $S_i^k(d) \leftarrow p$.
Message Broadcast(d):
  if ($D_i^k(i,d) \neq D_i^{k-1}(i,d)$)
  send $\{d, D_i^k(i,d)\}$ to all neighbors $j \in N(i)$.

To perform the above procedures, each node maintains two tables, the so-called *distance* table and route table. The distance table has local routing information (distance ($D_i^k(j,d) + l_i^k(j)$)) from node i to destination d via each neighbor $j \in N(i)$. After performing the Route Update procedure for destination d, the result is stored in the route table (minimum distance ($D_i^k(i,d)$) and route successor ($S_i^k(d)$)) on the shortest pathe from node i to destination d. Figure 1 describes an example of separate (distance and route) tables after convergence. The lines with arrow indicate the shortest paths from node i to all other nodes.



(a)

Destination	Distance	
	via a	via b
a	1	X4
f	3	2
b	4	1
g	5	2

(b)

Destination	Successor	Distance
a	a	1
f	b	2
b	b	1
g	b	2

(c)

Fig 1. (a) 5 node network.
(b) Distance table at node i.
(c) Route table at node i.

www.dbpia.co.kr

2.2 Looping Problem

Due to the distributed nature of the algorithm, route tables among some nodes may be inconsistent with one another at certain instants of time during an update period. This inconsistency happens due to out-dated routing information and is the main cause of looping. As long as a loop exists, packets are forced to circulate the loop without reaching their destinations.

Consider a 3-node network in Figure 2. Assume that $l_a^k(b)=1$, $l_b^k(c)=1$ and $l_a^k(c)=3$. There are two possible paths from node a to node c: one via node c suddenly fails but node a continuously sends data packets to node b. Node b now changes its route successor for destination c from node c to node a and sends back packets to node a with an update message $[c, 3]$. After receiving $[c, 3]$ from node b, node a changes its shortest path to the upper direct link. Returned packets from node b are now re-routed through the new shortest path to node c. This temporary loop does not affect network performance seriously and can be avoided by no distributed routing algorithms since they are inherent in a distributed environment.

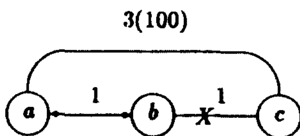


Fig 2. Example of looping.

On the other hand, assume that $l_a^k(b)=1$, $l_b^k(c)=1$ and $l_a^k(c)=100$ before the link between node b and node c fails. After the failure of that link, node b sends an update messge $[c, 3]$ to node a same as before. Even after receiving $[c, 3]$ from node b, however, node a remains on the previous shortest path and sends back $[c, 4]$ to node b and node b again returns $[c, 5]$, and so on. Node a and node b continuously exchange update messages until $D_a^{k+99}(b, c)=101$. In this case, node a continues to send data packets to node b and vice versa during 99 iteration steps, forming a loop. This kind of loop exists for a while, sometimes permanently, and are more severe in degrading network performance than the temporary loops discussed before.

Ⅲ. Proposed Algorithm

3.1 Motivation

The main idea of the proposed algorithm is that each node i constructs a set of SPTs, each of which is rooted at its neighbor $j \in N(i)$. The SPTs are constructed only with the update messages received from neighbors. Let an update message for a destination contain the identity of the destination and the estimated minimum path distance as well as the predecessor of the destination on the shortest path. Since update messages are broadcasted via neighbors throughout the network, every node in the network knows the individual predecessors of all destinations on the shortest paths after a while. Note that every destination becomes the predecessor of another destination unless the destination is in the leaf of a SPT. Then node i can construct the SPT by concatenating the predecessors of all destinations. The SPT built only with the update messages sent from neighbor j is the one rooted at neighbor j.

The SPT rooted at neighbor j must include every node in the network. Since the SPT is constructed at node i, it can be also viewed as the one rooted at node i. That is, the root of SPT, neighbor j, is extended into node i by a direct link between them. The shortest path from node i to destination d via neighbor j is included in this SPT. To be a loop-free shortest path, the same node should not appear more than once on any path. To determine the existence of a loop, all intermediate nodes on the path are checked by a procedure, called *back-tracking*, starting from destination $d \neq i$. A loop is now declared to be "formed" if node i appears more than once on the path. Thus the neighbor j providing the loop-

www.dbpia.co.kr

forming path should be excluded from being the route successor of node i on the shortest path to destination d.

3.2 Description

For the proposed algorithm to work properly, the SPTs at each node should be correctly maintained at all times. Since routing information concerning the predecessors of all destinations is essential to constructing the SPTs, this information should be known to every node in the network. For this purpose, a new field called the *predecessor* field is added into an original update message. Let $P_i^k(j, d)$ denote the predecessor of destination d known to node i on the shortest path from node j to destination d at iteration k. Then, the new update message for destination d generated from node j after an iteration step k takes the following form :

$$[d, D_j^k(j, d), P_j^k(j, d)]$$

where $P_j^k(j, d) = P_j^k(S_j^k(d), d)$.

Before we proceed further to describe the proposed algorthm, it is assumed that the distance table has been already updated due to either detecting some changes of out going link lengths or receiving update messages from neighbors. The following two(predecessor and eligibility) procedures are needed only when a node receives update messages from neighbors. If the algorithm is initiated by the former case, the control of the algorithm directly proceeds to update the route table.

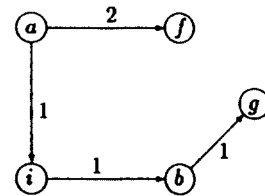
3.2.1 Predecessor Update

After exchanging update messages with neighbors, a node can construct the SPTs rooted at each neighbor using the update messages sent by the neighbors. For a destination, the shortest paths included in the different SPTs may show different predecessors of the destination. To keep this information concerning the predecessors of destinations on each SPT, a node needs another table, called the *predecessor* table. Each column of a predecessor table represents the SPT rooted at each neighbor. Figure 3 shows the predecessor table and the corresponding SPT rooted at each neighbor of node i in the previous 5-node network.

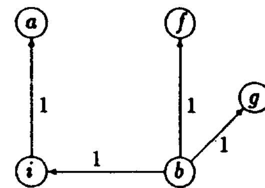
The (d, j) entry $P_i^k(j, d)$ of the predecessor table at node i after an iteration step k contains the predecessor of destination d on the SPT rooted at neighbor $j \in N(i)$. When an update message $[d, D_j^{k-1}(j, d), P_i^{k-1}(j, d)]$ arrives at node i from neighbor j, $P_j^{k-1}(j, d)$ in the update message replaces the (d, j) entry of the predecessor table ;

Destination	Predecessor	
	via *a*	via *b*
a	*i*	*i*
f	*a*	*b*
b	*i*	*i*
g	*b*	*b*

(a)



(b)



(c)

Fig 3. (a) Predecessor table at node i.
(b) SPT rooted at neighbor a.
(c) SPT rooted at neighbor b.

www.dbpia.co.kr

Predecessor Update(d) :
if (node i receives ad $[d, D_j^{k-1}(j, d), P_j^{k-1}(j, d)]$ from neighbor j)
  $P_i^k(j, d) \leftarrow P_j^{k-1}(j, d)$.

3.2.2. Eligibility Update

Denote the set of nodes on the shortest path from node i to destination d via neighbor j after an iteration step k by $SP_i^k(j, d)$. Each node i now constructs a shortest path $SP_i^k(j, d)$ by back-tracking, from destination d, the predecessors on the j-th column of predecessor table. Define a back-tracked successor to be the node which is last included in $SP_i^k(j, d)$ before the back-tracking process terminates. The back-tracked successor is denoted by $B_i^k(j, d)$ and should be distinguished from the route successor $S_i^k(j, d)$. The route successor is one of the eligible back-tracked successors whose path distance to destination d indicates a minimum. Under normal operation when no loops are formed, it is obvious that

$$B_i^k(j, d) = j, \ \forall j \in N(i). \tag{1}$$

Otherwise,

$$B_i^k(j, d) \neq j, \ \forall j \in N(i). \tag{2}$$

The following theorem shows that the equations (1) and (2) can be used for loop-detection.

Theorem 1 *Consider a shortest path $SP_i^k(j, d)$ from node i to destination d via neighbor j after an iteration step k. Then a loop is formed on the path $SP_i^k(j, d)$ if*

$$B_i^k(j, d) \neq j, \ \forall j \in N(i).$$

Proof. The shortest path $SP_i^k(j, d)$ is obtained by back-tracking the predecessors on the j-th column of predecessor table. If $B_i^k(j, d) \neq j$, the back-tracking process must have terminated at any other node between neighbor j and destination d by encountering node i. This implies that node i appears more than once and the loop has formed on the path $SP_i^k(j, d)$.

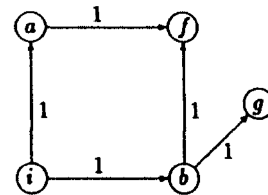
The eligibility, $B_i^k(j, d) = j$, $j \in N(i)$, is checked every time the predecessor table is updated by an update message. Suppose that an update message for destination d arrives at node i from neighbor j and replaces the (d, j) entry of the predecessor table. The procedure to check the eligibility of neighbor j is as follows :

```
  Eligibility Update(d) :
  if(O_i^k(j, d) ≠ P_i^{k-1}(j, d))
  p ← d;
while (P_i^k(j, p) ≠ i)/* back-track*/
p ← P_i^k(j, p) ;
if(j = p)        /* p: B_i^k(j, d) */
  E_i^k(j, d) = 1/* j is eligible */
else
E_i^k(j, d) = 0.  /* j is non-eligible */
```

To avoid repeated computations, the results are stored in the so-called *eligibility* table. The (d, j) entry of the eligibility table, denoted by $E_i^k(j, d)$, contains the eligibility of neighbor j as the route successor on the shortest path from node i to destination d. If neighbor j is eligible for the route successor, $E_i^k(j, d) = 1$, otherwise, $E_i^k(j, d) = 0$.

Destination	Eligibility	
	via a	via b
a	1	0
f	1	1
b	0	1
g	0	1

(a)



(b)

Fig 4. (a) Eligibility table at node i.
(b) Eligible paths from node i to all other nodes.

www.dbpia.co.kr

Figure 4 show the eligibility table at node i in the 5-node network. For this example, it is necessary to examine the SPTs shown in Figure 3. It is known that a loop may form due to any increases of link length since node i itself is found in both SPTs (Figure 3(a) and (b)). Since $B_i^k(a,b) = B_i^k(a,g) = b \neq a$, $E_i^k(a,b) = E_i^k(a,g) = 0$. Also since $B_i^k(b,a) = a$, $E_i^k(b,a) = 0$. That is, neighbor a is not eligible for the route successor of the shortest paths to node b and node g(Figure 3(a)). On the other hand, neighbor b should not be chosen as the route successor for the shortest path to node a. Figure 4(b) shows possible loop-free paths from node i to all other nodes. In this case, node i only needs to select a route successor (neighbor a or b) for destination f.

3.2.3 Route Update and Message Broadcast

The route successor $S_i^k(d)$ of node i for destination d should be the neighbor j whose entry $E_i^k(j,d)$ of the eligibility table indicates 1 while minimizing the path distance from node i to destination d. An update message is also sent out to all neighbors whenever $D_i^k(i,d)$ or $P_i^k(i,d)$ changes. Thus the proposed procedures for updating the route table and broadcasting update messages are modified as follows :

Route Update(d) :

$D_i^k(i,d) \leftarrow \min_{\{j:E_i^k(j,d)=1\}} \{D_i^k(j,d) + l_i^k(i,j)\}$;

With $p \in N(i)$ chosen,

$$S_i^k(d) \leftarrow p$$

Message Broadcast(d) :

if($D_i^k(i,d) \neq D_i^{k-1}(i,d)$ or $P_i^k(i,d) \neq P_i^{k-1}(i,d)$)

send $\{d, D_i^k(i,d), P_i^k(i,d)\}$ to each neighbor $j \in N(i)$;

3.3 Improvements

The proposed algorithm introduced above may cause a little problem : update messages from neighbors arrive at a node in arbitrary order and could cause some dealys in detecting a loop. The following theorm states that there exists a node that plays a key role in maintaining the correct SPTs at each node.

Theorem 2 *Consider a shortest path $SP_i^k(i,d)$ from node i to destination d after an iteration step k. A node $j \in SP_i^k(i,d)$ will change its route successor if and only if there is a node $c \in SP_i^k(i,d)$ which changes its predecessor on the path $SP_i^k(i,d)$.*

Proof. Suppose that no node on the path $SP_i^k(i,d)$ changes its route successor for destination d after an iteration step k. Then the shortest path should remain same as before : $SP_i^{k+1}(i,d) = SP_i^k(i,d)$. This implies that there exists no node on the path $SP_i^{k+1}(i,d)$ whose predecessor has been changed and contradicts the assumption. Similarly, the converse can be proved.

In Theorem 2, let us call the node c which changes its predecessor on the shortest path the *critical* node. The SPT changes its structure based only on the ciritical node. To maintain a correct SPT, updates for some destinations (downstream of the critical node) should be preceded by an update for the critical node. However, the proposed algorithm presented before does not guarantee an update message for the critical node to be processed first. Some of downstream destinations could be updated on the basis of the old SPT and loops may be formed. Those loops may not be detected even after updating the eligibility table for the critical node. The undetected loop will eventually be found when update messages for those destinations arrive later. Such delayed detection of looping can be avoided by grouping some related update messages, all the update messages related to a same link length change may be tied together. The Predecessor Update procedure is first performed for the destinations included in a group of update maessages. Then, the control of the routing algorithm proceeds to update the eligibility table for those destinations. By doing so, the predecessor table is properly updated for the correct decision of each neighbor's eligibility.

Another way is to include the identities of critical node and its predecessor in an update message

www.dbpia.co.kr

instead of a destination's predecessor. Changing a critical node directly affects the SPTs at other nodes and happens only when a node on the shortest path changes its route successor. (by Theorem 2). Thus a node which changes its route succcessor for a destination is responsible for broadcasting the information concerning the critical node and its predecessor throughout the network. An update message for destination d generated from node j after an iteration step k now takes a new form :

$$[d,\ D_j^k(j,d),\ c,\ P_j^k(j,c)]$$

where c is a critical node and $P_j^k(j,c)$ is the predecessor of critical node c on the shortest path $SP_j^k(j,c)$.

The critical node is the one having different predecessors, which is encountered first in backtracking, from destination d to node i itself, two columns (one for $S_i^k(d)$ and the other for $S_i^{k-1}(d)$) of the predecessor table. If node i does not change its route successor but only changes its minimum distance to destination d, node i must have received an update message from the current route successor on the shortest path to destination d or detected any change of the link length connected to that route successor. If the former is the case, node i copies the critical node and its predecessor from the update message which is sent by the route successor. Otherwise, node i does nothing. An update message does not necessarily contain the information concerning the critical node and its predecessor. The proecdures to find a critical node is as follows ;

Critical Node(d) :
if ($S_i^k(d) \neq S_i^{k-1}(d)$)
  $p \leftarrow d$;
  $q \leftarrow d$;
while ($P_i^k(S_i^k(d), p) = P_i^k(S_i^{k-1}(d), p)$)/*back-track*/
  $p \leftarrow P_i^k(S_i^k(d),\ p)$
  $q \leftarrow P_i^k(S_i^{k-1}(d),\ p)$
  $c \leftarrow p$;   /* *critical node* */
  $P_i^k(i,\ c) \leftarrow P_i^k(S_i^k(c),\ c)$; /* predecessor */
else if ($D_i^k(i,d) \neq D_i^{k-1}(j,d)$)
  if (node i receives an update message from $S_i^{k-1}(d)$)
    copy c and $P_i^k(i,c)$ from the update message
  else
    $c \leftarrow null$
    $P_i^k(i,c) \leftarrow null$;
else
    $c \leftarrow null$;
  $P_i^k(i,c) \leftarrow null$;

The rest of the proposed algorithm is modified to deal with the critical node. For destination d, the Predecessor Update procedure replace $(j,\ c)$ entry instead of $(j,\ d)$ entry in the predecessor table. The Eligibility Update procedure back-tracks starting from the critical node c instead of destination d and is followed by the Critical Node procedure. The Message Bradcast procedure also broadcasts the modified update message contatining the identities of ciritical node and its predecessor to all neighbors.

Ⅳ. Performance Analysis

The convergeence proof of the proposed algorithm refers to [12] since it differs from the Humblet algorithm only in regard to the way a loop is detected and resolved with the least amount of overhead, we focus on analyzing the performance of the proposed algorithm in terms of recovery speed from link/node failures in the following section. The recovery speed is defined as interval between the time a network runs into an unstable state due to link/node failures and the time the network returns to a normal states.

In the proposed algorithm, each node scans all the downstream nodes starting from a destination to determine the eligibility of its neighbor as a route successor and, optinally, to find out the critical node on the shortest path. In the worst case, a node performs back-tracking for all destinations via all neighbors. In other words, per-

www.dbpia.co.kr

forming the Eligibility Update and the optional Critical Node procedures in addition to updating the route table results in the increased computational overhead of $O(mNh)$ where m is the number of neighbors adjacent to a node, N is the total number of nodes in the network, and h is the height of the shortest path. It is also believed that the storage overhead to maintain additional tables(predecessor table and eigibility table) is negligible due to the dramatically reduced cost of memory. The size of update message is slightly increased to include predecessor (optionally, critical node) field and also does not significantly degrade the overall network performance.

4.1 Speed of Recovery

Before demonstrating the recovery speed of the proposed algorithm, we assume that it takes a unit time for an update message to be processed in a node and to traverse a link. We also assume that lower level (physical layer and link layer) protocols maintain an error free communication link between two nodes. Then the following theorem states that a loop can be always detected and resolved by the proposed algorithm within a finite time depending on the number of nodes involved in the loop

Theorem 3 *Consider an out-going link length change at an iteration step k causing loop-forming at another iteration step $k' \geq k$.*

(a) There exists at least a node i inside the loop such that

$$B_i^{k''}(S_i^k(d), d) \neq S_i^{k''}(d)$$

where $k'' \geq k'$

(b) Denote the number of nodes involved in the loop by l. Then

$$k'' - k' \leq l.$$

Proof. (a) Route successors of the nodes involved in a loop are inside the loop. While the loop exists, update messages containing finite path distance to destination d will be circulating the loop continuously. The existence of such update messages implies that there exists at least a path leading to destination d. Otherwise, the loop is broken automatically by the node receiving an update message containing infinite path distance. Thus, by back tracking the predecessors from destination d, we should be able to find a node i whose back-tacked successor $B_i^k(S_i^k(d), d)$ on the path $SP_i^k(i,d)$ is located outside of the loop. However, since the route successors of all nodes in the loop should be located inside the loop, $B_i^k(S_i^k(d), d) \neq S_i^k(d)$.

(b) The time needed to detect the loop is between k' and k'' (from (a)). At an iteration step k'', node i receives and update message from the route successor $S_i^k(d)$. The update message contains the information concerning the backtracked successor $B_i^k(S_i^k(d),d) \neq S_i^k(d)$. This update message is, in the worst case, generated from the predecessor of node i and received again by node i after circulating the whole loop. Then, it takes at most l time units for node i to recognize and escape from the loop.

This theorem directly applies to the proposed algorithm with the Critical Node procedure. For the theorem also to be valid for the proposed algorithm whithout the Critical Node procedure, update messages must be sent to neighbors in group so that delaying problem due to out-of-order update messages can be removed.

4.2 Compariosn to Other Algorithms

4.2.1 Loop-Free Algorithms

Among a series of loop free algorithms [8][9][10], Jaffe-Moss algorithm is the best in terms of revovery speed after link/node failures. The second version of Jaffe-Moss algorithm provides the recovery speed of $O(h)$ where h is the height of the shortest path tree. More precisely, it takes $3h$ unit times for the Jaffe-Moss algorithm to restart normal routing updates since failures occurred.

In general, it is difficult to compare the lengths of h in Jaffe-Moss algorithm and l in the proposed

www.dbpia.co.kr

algorithm since they change dynamically depending on the network topologies and traffic conditions etc. When link/node failures happened in the downstream of a node and a loop is formed, l may be less or greater than $3h$, that is, $l=O(h)$. It is apparent that the proposed algorithm is more efficient than the Jaffe-Moss algorithm when l is less than $3h$. In practice, there are many cases where $l \leq 3h$ provided that the network is not seriously unstable. The chances of l being less than $3h$ can be further reduced by carefully dimensioning the network.

4.2.2 Loop-Detection Algorithms

Most of loop-detection algorithms are effective only for preventing two-node loops. To cope with multi-node loops, they put too much burden on the network with update messages or processing overheads[13]. From the view-point of those overheads, Humblet algorithm[12] is superior to others and are considered here for comparison with the proposed algorithm.

In Humblet algorithm, loop-detection is performed by constructing the local STPs rooted at neighbors-the same concept used in the proposed algorithm. Thus the recovery speed from link/node failures in l time units. However, the Humblet algorithm uses a loop-detection technique similar to Dijkstra algorithm. The computational overhead to detect loops becomes $O(mN^2)$ which is larger than $O(mhN)$ of the proposed algorithm (usually $h \ll N$). A more careful observation shows that, in the Humblet algorithm, some destinations are updated regardless of their necessities once the loop-detection protocol is activated. To the extreme case, the whole STP is reconstructed even when an update message for a single destination arrives at a node. On the other hand, the proposed algorithm performs routing updates only for concerned destinations and processing overhead to detect loops can be saved. Therefore, it can be said that the proposed algorithm is at worst comparable to the Humblet algorithm in terms of computational overhead (when all destinations require to be updated and $h=N$).

V. Summary

After a brief discussion of the distriuted Bellman-Ford algorithm and the associated looping problems, a new distributed shortest path routing algorithm for loop detection and resolution has been proposed. The proposed algorithm gathers routing information (SPTs) from the local routing information (update messages) sent from neighbors. From the SPTs constructed locally at each node, a loop is detected by any node involved in the loop in a distributed manner. The proposed algorithm then immediately resloves the loop by changing the current route successor on the shortest path. Any kinds (two-node and multi-node) of loop can be eliminated and the recovery speed depends on the number of nodes involved in the loop. That is, with the proposed algorithm, loops are allowed to exist for a finite time and are inevitable under any distributed algorithms.

The proposed algorithm requires relatively small computation and memory overhead. No extra update messages are generated and communicated over the network. The size of an update message is slightly increased by including the predecessor field (optionally, the critical node). The proposed algorithm also maintains the simplicity and the dstributed nature of Bell-man-Ford algorithm. Compared to others, the proposed algorithm turns out to be more efficient unless the network is seriously unstable.

References

1. M.McQuillan, "Adaptive routing algorithms for distributed computer networks", Bolt Beranek and Newman Inc., BBN Rep. 2831, may 1974.
2. T. Cegrell, "A routing procedure for the TIDAS message swithing network", *IEEE Trans. Commun.*, vol. COM-23, pp.575-585, June 1975.

www.dbpia.co.kr

3. E. Sproule and F. Mellor, "Routing, flow, and congestion control in the Datapac network," *IEEE Trans. Commun.*, vol. COM-29, pp.386-391, April 1981.
4. DECnet, Digital Network Architecture, Routing Layer Functinal Specification, Version 2. 0. 0., Digital Equipment Corp., Maynard, MA, 1983.
5. M. McQuillan, I. Richer and E. C. Rosen, "Then new routing algorithm for the ARPANET", *IEEE Trans. Commun.*, vol. COM-28, pp. 711-719, May 1980.
6. E. W. Dijkstra, "A note on two problems in connection with graphs," *Numer. Math.* vol. 1, pp.269-271, 1959.
7. D.P. Bertsekas and R.G. Gallager, *Data Network*, Englewood Cliffs, NJ : Prentice Hall, 1987.
8. M. merlin and A. Segall, "A fail-safe distributed routing protocol", *IEEE Trans. Commun.*, vol. COM-27, pp. 1280-1287, May 1980.
9. A. Segall, "Advance in verfiable fail-safe routing procedures", *IEEE Trans. Commun.*, vol. COM-29, pp.491-497, April 1981.
10. M. Jaffe and F.H. Moss, "A responsive distributed routing algorithm for computer networks", *IEEE Trans. Commun.*, vol. COM-30, pp.1758-1762, July 1982.
11. M. Schwartz, *Telecommunication Networks : Protocols, Modeling and Analysis*, Addison Wesley, New York, 1987.
12. P. A. Humblet, "Another adaptive distributed shortest path algorithms", *IEEE Trans. Commun.*, vol. COM-39, pp.995-1003, June 1991.
13. K. G. Kang and M. S. Chen, "Performance analysis of distributed routing strategies free of ping-pong-type looping", *IEEE Trans. Commun.*, vol. C-36, No.2, pp.129-137, Feb. 1987.

朴 聖 宇(Sung Woo Park) 정회원

1962년 9월 13일생

1981년 3월 ~ 1985년 2월 : 연세대학교 전자공학과 졸업(공학사)

1985년 1월 ~ 1986년 3월 : 한국 데이타 통신(주) 연구원

1986년 6월 ~ 1989년 8월 : Texas A&M Univ. Dept. of Electrical Engineering(MS)

1989년 9월 ~ 1991년 12월 : Univ. of Califonia, Irvine Dept. of Electrical and Computer Engineering(Ph.D)

1992년 3월 ~ 현재 : 한남대학교 정보통신공학과 조교수

※주관심분야 : Computer Networks

金 永 川(Young Chon Kim) 정회원

1956년 12월 10일생

현재 : 전북대학교 컴퓨터공학과 부교수

제18권 제1호 참조

www.dbpia.co.kr