A Multi-path Routing Protocol For Unidirectional Networks

Wei Lou and Jie Wu
Department of Computer Science and Engineering
Florida Atlantic University
Boca Raton, FL 33431
{wlou, jie}@cse.fau.edu

Abstract *Most of routing protocols designed for the Internet today are for networks with bidirectional links. These protocols cannot be easily extended to networks with unidirectional links without increasing cost. In [1], Chen et al. presented a distance-vector-based routing protocol for unidirectional networks where a routing table is derived for each node in the network. In this paper, we extend the protocol in [1] by keeping one path for each outgoing link of a node to maintain multiple paths in the corresponding routing table. The path for a specific outgoing link is the shortest path from the node to the destination through that outgoing link. Thus, the protocol provides alternative paths that are vital in the dynamic network environment.*

Keywords: Multi-path routing protocols, routing algorithms, unidirectional networks

I. Introduction

Conventional routing protocols can be classified into link state and distance vector protocols. In link state routing protocols, each node maintains a view of the network topology with a cost for each link. In distance vector routing protocols, each node maintains a distance to each destination. Such information is kept in a routing table associated with each node. Routing protocols currently used for the Internet are based on the simple assumption that any two neighbors can bi-directionally exchange information. The traditional routing protocols like RIP [2] (a distance vector protocol) or OSPF [3] (a link state protocol) are both based on this assumption. As more and more mobile applications emerge, this assumption faces big challenges. However, for most mobile applications, distance vector routing protocols still use this assumption to generate routing tables. This will cause serious problems. Therefore, new algorithms based on the unidirectional networks topology need to be considered.

As a basic figure of the mobility patterns, the link between two neighboring nodes is temporarily connected and often unidirectional due to the different physical environment and the disparity of the transmission power levels of the two nodes. Figure 1 gives an example of ad hoc wireless networks (a special type of wireless network without infrastructure) where host A can receive messages from host B while B cannot receive messages from A. This could happen when B has a small transmission scope that cannot cover A while B is in A's transmission scope. B has to get A's message from host C. Therefore, the distance vector routing protocol mostly used in the Internet needs to consider the unidirectional links between two neighbors. The existence of unidirectional links may generate totally difference paths from A to B and from B to A even though they may be neighbors. In the above example, the path from A to B is (A, B) while the path from B to A is (B, C, A).

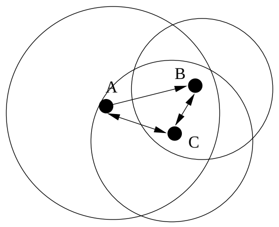

Fig. 1. Network with unidirectional links

In [4], Prakash pointed out that a distance vector routing protocol for unidirectional networks requires $O(n^2)$ information exchanges. This protocol uses an $n \times n$ D matrix to maintain the distance between two nodes and the routing table is constructed thereafter. The protocol proposed by Chen [1] reduced the complexity of information storage to $O(n)$, with carefully constructed FROM and TO tables of each node. This protocol, unfortunately, needs many rounds to reach the stable status in some cases and only one path is kept for each destination.

The protocol presented in the paper is an extension to the protocol in [1]. One path is kept for each outgoing link of the node in the table. Thus the routing table can provide multiple paths to each destination node. Each path for a specific outgoing link is the shortest path from the node to the destination through that outgoing link. The design of this protocol is based on a worst case situation that the network topology is frequently changed and that providing alternative paths are vital in this dynamic network environment.

The rest of the paper is organized as follows: Section 2 gives a brief description of some protocols for unidirectional networks. The extended protocol is given in Section 3. In Section 4, we consider some situations in dynamic networks and discuss the merits of this protocol. In the end, we draw some conclusions in Section 5.

This work was supported in part by NSF grant CCR 9900646 and grant ANI 0073736.

II. Related Works

Routing in unidirectional networks is much different from that in bidirectional networks because routing information can only be diffused in one direction. Based on the distance vector protocol in bidirectional networks, the complexity of information exchange and storage is $O(n)$, where n is the number of the nodes in the network. But in the network containing unidirectional links, $O(n)$, as the amount of information exchange, is not sufficient for each node to construct a complete routing table. Some modified protocols have been proposed recently with the consideration of the unidirectional links. All these protocols are in the category of distance vector algorithms, not in that of link state algorithms.

Prakash [4] proposed a protocol based on the protocols used in ad hoc wireless networks like DSDV [5] and AODV [6]: Each node maintains an $n \times n$ D matrix which indicates the distance between two nodes. A node's $From(To)$ $vector$ includes the distance and previous (next) hop from (to) all the other nodes. The node uses the D matrices received from its neighbor nodes to update its own D matrix and also to construct its $From$ and To vectors. Therefore, $O(n^2)$ as the amount of information exchange and storage is needed for this protocol. It is also proved that the routing paths determined by this protocol are loop-free.

Another protocol proposed by Chen et al. [1] reduces the complexity of information storage to $O(n)$: Each node first collects information from its predecessors' FROM tables to construct its FROM table. It then uses these FROM tables to generate its TO table. The algorithm used in this protocol is similar to the one in a regular distance vector protocol. Since the FROM table and TO table are of the size $O(n)$, the total complexity for this protocol is also $O(n)$. This algorithm requires the network be strongly connected so that a cycle exists that connects any two nodes. In some cases, a routing table needs many rounds to converge. In addition, only one possible path is kept in the routing table which might be insufficient in a changing environment, especially in the dynamic network environment.

III. Routing Protocol

Our approach is an extension of the protocol in [1]. We try to generate the routing table of each node by maintaining more information in each round. The TO table of a node allocates one entry to store the path information for each outgoing link from the node. The path for a specific outgoing link is the shortest path from the node to the destination through that outgoing link. The complexity of the protocol is $O(|E|)$, where $|E|$ is the number of the directed links of the network. Note that multiple paths generated from this protocol also provide alternate paths which are vital in dynamic networks.

A. Data structure

The network can be described as a directed graph $G = (V, E)$, where V is a vertex set presenting the hosts and E is an edge set presenting the links. We follow the notations used in [1]. (P, Q) represents the directed link from node P to node Q. P is called Q's $f\text{-}neighbor$ (predecessor) and Q is called P's $t\text{-}neighbor$ (successor). The cost of the link (P, Q) is denoted by $d(P, Q)$. The path from P to Q via nodes N_1, N_2, ..., N_k is represented as $(P, N_1, N_2, ..., N_k, Q)$. It is assumed that graph G is strongly connected, which implies that any pair of vertices in G can be reached by each other, and hence, a cycle exists that connects any two vertices. A cycle is a path with two end nodes being the same node.

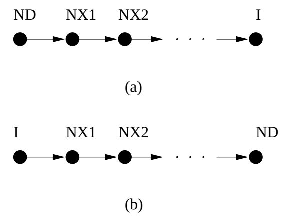

Fig. 2. (a) a FROM entry (b) a TO entry

Each node needs to maintain a FROM table (FT) and a TO table (TT). The format of entry e in FT is $(ND, DT, NX1, NX2, TTL)$, shown in Figure 2(a), which represents a path from source node ND to the node I that contains the FT table. The next node from node ND is $NX1$, and the node with two steps from ND is $NX2$ if such a node exists, else $NX2$ will set to ϕ. The total cost of the path from ND to I is DT. TTL is a timer associated with the entry. TTL is initialized to a constant value T and is decreased as time goes by. When TTL reaches zero, the entry will be considered expired and will be deleted from the FT. The format of the entry of a TO Table (TT) is represented as $(ND, DT, NX1, NX2, TTL)$, but ND here means the destination node, $NX1$ is the first hop from this node I, and $NX2$ is the second hop from node I, as shown in Figure 2(b). Also, field $NX2$ will be set to ϕ if such a node does not exist. When a field in the entry is not concerned in the discussion, we use "-" in that field.

We use the following notation to represent a node's tables, table's entries, and fields within the entries, such as $P.FT$ for the node P's FROM table, $FT.e_i$ for the i^{th} entry in the FROM table, and $e.NX1$ for the $NX1$ field of the entry e.

B. FROM table construction

The algorithm FROM in [1] is used to construct the FROM tables for the network. The FROM algorithm only maintains one path for each node. The format of a FROM table's entry is (ND, DT, NX, TTL) in which fields have the same meanings as indicated above. Algorithm FROM uses the IF statement ($e.NX$ = $e'.NX$ or $e.DT \leq e'.DT$) to maintain the shortest path information in the entry: $e.NX = e'.NX$ is used to refresh the existing entry. $e.DT \leq e'.DT$ is used to replace the entry if an entry with a shorter path has been found. The procedure uses the ELSE IF statement $e.DN \neq Q$ to keep the cycle (from Q to Q) away from the FROM table.

The following CFT algorithm is almost the same as the algorithm FROM. Each node constructs its FROM table based on its predecessors' FROM table information. A node periodically receives the updated FROM table from its predecessors. The node also sends its FROM table to its successors periodically. Since the FROM table needs to keep an entry for each outgoing link

Algorithm 1 FROM algorithm [1]

```
1. Initially, all FROM tables are empty.
2. Each node periodically sends to all its $t$-neighbors a FROM packet
   containing its FROM table.
3. When a node $Q$ receives a FROM packet $F$ from f-neighbor $P$,
   containing $P$'s FROM table:
   (a) For each $f \in F.FT$, $Q$ generates $e = \{f.ND, f.DT + d(P,Q), f.NX, T\}$ and executes the procedure:
   if $\exists e' \in Q.FT$ then
      if $e.NX = e'.NX \vee e.DT \leq e'.DT$ then
         replace $e'$ by $e$;
      end if
   else
      if $e.ND \neq Q$ then
         add $e$ to $Q.FT$;
      end if
   end if
   (b) $Q$ creates $e = \{P, d(P,Q), Q, T\}$ and executes the above
   procedure upon $e$.
4. The entry is deleted from the table when the timer of the entry
   expires.
```

of a node, fields ND and $NX1$ are used to identify this outgoing link. There is one more field $NX2$ in the entry because we need to use $(e.NX2 = e'.NX2)$ instead of $(e.NX = e'.NX)$ in the IF statement of the procedure in step 3(a). $NX2$ has the same function as the field NX in algorithm FROM. A specific case needs to be considered here: When a node Q receives its predecessor P's FT and finds an entry in which field $NX2$ is ϕ, Q will just put Q in the field $NX2$ to generate a new entry. In step 3(b), the entry $e = \{P, d(P,Q), Q, \phi, T\}$ indicates the direct link (P,Q) which has no $NX2$.

Algorithm 2 CFT algorithm (Construct a FROM Table)

```
1. Initially, all FROM tables are empty.
2. Each node periodically sends to all its $t$-neighbors a FROM packet
   containing its FROM table.
3. When a node $Q$ receives a FROM packet $F$ from f-neighbor $P$,
   containing $P$'s FROM table:
   (a) For each $f \in F.FT$, $Q$ generates $e = \{f.ND, f.DT + d(P,Q), f.NX1, NX2,\ T\}$, $NX2$ is either $f.NX2$ or $Q$ if
   $f.NX2$ is $\phi$, and executes the procedure:
   if $\exists e' \in Q.FT$ that $e'.ND = e.ND \wedge e'.NX1 = e.NX1$ then
      if $e.NX2 = e'.NX2 \vee e.DT \leq e'.DT$ then
         replace $e'$ by $e$;
      end if
   else
      if $e.ND \neq Q$ then
         add $e$ to $Q.FT$;
      end if
   end if
   (b) $Q$ creates $e = \{P, d(P,Q), Q, \phi, T\}$ and executes the above
   procedure upon $e$.
4. The entry is deleted from the table when the timer of the entry
   expires.
```

C. TO table construction

The TO table is the routing table used for sending information packages from source to destination. In [1], algorithm TO is triggered when a node Q receives a FROM table from its predecessor P and finds an entry in which ND field is Q. Then the algorithm uses this cycle to construct the entries from each node to node Q. The same procedure used in algorithm FROM is also used in algorithm TO to keep the most updated shortest path in the TO table.

Algorithm 3 TO algorithm [1]

```
1. Initially, all TO tables are empty.
2. When $Q$ receives a FROM packet F from an $f$-neighbor $P$ and
   $F.FT$ contains an entry whose $ND = Q$, then
   (a) $Q$ traces the path from $Q$ to $P$ in $F.FT$: ($Q = N_0, N_1, N_2, ..., N_m = P$), $m \leq 1$. Let $e_1, ..., e_m$ be the corresponding entries in $F.FT$ where $e_i = \{-, -, N_i, -\}$.
   (b) For $i = 1$ to $m$, $Q$ generates the entry $e = \{N_i, d_i, N_1, T\}$,
   where $d_i = d_{i-1} + [e_i.DT - e_{i+1}.DT]$ and $d_0 = d_{m+1} = 0$,
   and executes the procedure:
   if $\exists e' \in Q.FT$ then
      if $e.NX = e'.NX \vee e.DT \leq e'.DT$ then
         replace $e'$ by $e$;
      end if
   else
      if $e.ND \neq Q$ then
         add $e$ to $Q.FT$;
      end if
   end if
   (c) $Q$ sends a TO packet $T$ containing its TO table to $P$ using
   algorithm SOURCE ROUTE.
3  When $P$ receives $T$ from its child $Q$,
   (a) for each $t \in T.TT$, it generates $e = \{t.ND, t.DT + d(P,Q), Q, T\}$ and executes the above procedure.
   (b) $P$ generates $e = \{Q, d(P,Q), Q, T\}$ and executes the above
   procedure upon $e$.
4. The entry is deleted from the table when the timer of the entry
   expires.
```

The CTT algorithm follows the same idea. The $F.FT$ keeps the shortest paths from all other nodes to P, including node Q, then Q can find the shortest path from Q to P in $F.FT$. Thus, Q can construct its partial TO table with this path information. As used in steps 2a and 2b. Q uses the entry's DT field in P's FROM table to calculate the DT field of the entry in Q's TO table, from node N_1 to node N_m along this path. The procedure used in CFT algorithm is also used here in step 2b. P's TO table can be updated if a new shorter path is found. When P's TO table is constructed, this TO table is sent to its predecessor Q so that Q can construct Q's TO table by just appending link (Q,P) to P's TO table entries. This is what step 3 does. In step 4, node P will perform the loop-free check for the newly added entry since step 3 will introduce some entries that may cause a loop in its path to the destination. Algorithm TO differs significantly from algorithm CTT: algorithm TO just keeps one shortest path for each node and there will be no loop. The loop-free check procedure uses the node's FROM table to detect the loop in an entry because the FROM table keeps each node's shortest paths from all other nodes without any loop. Therefore, if an entry like $\{P, e.DT, e.NX1, e.NX2, -\}$ cannot be found in the FROM table of node $e.ND$, entry e in node P's TO table must contain a loop in its path. This type of entry is dangerous for the routing table and must be excluded from the TO table.

D. Case study

In this subsection, we first use the example in [1] to show the difference between the extended algorithm and the previous

Algorithm 4 CTT algorithm (Construct a TO Table)

```
1. Initially, all TO tables are empty.
2. When $Q$ receives a FROM packet F from an $f$-neighbor $P$ and $F.FT$ contains $e$ such that $e.ND = Q$, it does the following steps:
   (a) For the shortest path from $Q$ to $P$: $(Q = N_0, N_1, N_2, ..., N_m = P)$ with each outgoing link of $Q$ in $F.FT$, find the entries $e_1$, $e_2$, ..., $e_m$ in $F.FT$, where $e_i = \{N_{i-1}, -, N_i, -, -\}$.
   (b) For $i = 1$ to $m$, $Q$ generates $e = \{N_i, DT_i, N_1, NX2, T\}$, where $NX2$=$N_2$ if $i \neq 1$ or $phi$ if $i = 1$, $DT_i = DT_{i-1} + (e_i.DT - e_{i+1}.DT)$, $DT_0 = 0$, $e_{m+1}.DT = 0$, and executes the procedure:
   **if** $\exists e' \in Q.FT$ that $e'.ND = e.ND \wedge e'.NX1 = e.NX1$ **then**
      **if** $e.NX2 = e'.NX2 \vee e.DT \leq e'.DT$ **then**
         replace $e'$ by $e$;
      **end if**
   **else**
      **if** $e.ND \neq Q$ **then**
         add $e$ to $Q.FT$;
      **end if**
   **end if**
   (c) $Q$ sends $Q$'s TO table to $P$ using SOURCE ROUTE algorithm via the path $(Q, N_1, N_2, ..., P)$.
3  When a node $P$ receives a TO packet $T$ from its $t$-neighbor $Q$, it generates $e = \{t.ND, t.DT + d(P, Q), Q, t.NX1, T\}$ for each $t \in T.TT$, it also generates $e = \{Q, d(P, Q), Q, \phi, T\}$. $P$ executes the procedure in step 2(b) upon each $e$.
4  If $P$'s outgoing links are more than one, for a new $e$, $P$ will do the loop-free check procedure:
   (a) $P$ sends $(P, e)$ that combines $P$ and $e$ to $e.ND$ using the path found in the $P$.TO table.
   (b) When $e.ND$ receives $(P, e)$, it searches $e' = \{P, e.DT, e.NX1, e.NX2, -\}$ in its FROM table.
   (c) $e.ND$ sends an acknowledgement message back to $P$ indicating if $e'$ is found in $e.ND$'s FROM table or not.
   (d) When $P$ receives this acknowledgement that indicates $e'$ is not found, $P$ removes $e$ from its TO table.
5. The entry is deleted from the table when the timer of the entry expires.
```

one presented in [1]. The network topology of the example is shown in Figure 3(a). Since these tables need several rounds of information exchange to be fully filled, we show the tables of each round until they become stable.

We can use the CFT algorithm presented in the above section to get the FROM table in each round:

1) First round

A	B	C	D	E
-	A,1,B,ϕ,-	A,2,C,ϕ,-	-	-
-	-	B,2,C,ϕ,-	-	-
-	-	-	C,3,D,ϕ,-	-
-	-	-	-	D,2,E,ϕ,-
E,2,A,ϕ,-	-	-	-	-

2) Second round

A	B	C	D	E
-	A,1,B,ϕ,-	A,2,C,ϕ,- A,3,B,C,-	A,5,C,D,-	-
-	-	B,2,C,ϕ,-	B,5,C,D,-	-
-	-	-	C,3,D,ϕ,-	C,5,D,E,-
D,4,E,A,-	-	-	-	D,2,E,ϕ,-
E,2,A,ϕ,-	E,3,A,B,-	E,4,A,C,-	-	-

3) Third round

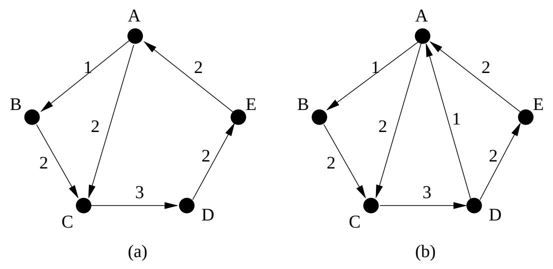

Fig. 3. (a) example 1 (b) example 2

A	B	C	D	E
-	A,1,B,ϕ,-	A,2,C,ϕ,- A,3,B,C,-	A,5,C,D,- A,6,B,C,-	A,7,C,D,-
-	-	B,2,C,ϕ,-	B,5,C,D,-	B,7,C,D,-
C,7,D,E,-	-	-	C,3,D,ϕ,-	C,5,D,E,-
D,4,E,A,-	D,5,E,A,-	D,6,E,A,-	-	D,2,E,ϕ,-
E,2,A,ϕ,-	E,3,A,B,-	E,4,A,C,-	E,7,A,C,-	-

4) Fourth round

A	B	C	D	E
-	A,1,B,ϕ,-	A,2,C,ϕ,- A,3,B,C,-	A,5,C,D,- A,6,B,C,-	A,7,C,D,- A,8,B,C,-
B,9,C,D,-	-	B,2,C,ϕ,-	B,5,C,D,-	B,7,C,D,-
C,7,D,E,-	C,8,D,E,-	-	C,3,D,ϕ,-	C,5,D,E,-
D,4,E,A,-	D,5,E,A,-	D,6,E,A,-	-	D,2,E,ϕ,-
E,2,A,ϕ,-	E,3,A,B,-	E,4,A,C,-	E,7,A,C,-	-

The TO table can be generated by the CTT algorithm with the same assumption described above:

1) First round

a) TO table after step 2 of CTT algorithm

A	B	C	D	E
-	A,9,C,D,-	A,7,D,E,-	A,4,E,A,-	A,2,A,ϕ,-
B,1,B,ϕ,-	-	B,8,D,E,-	-	-
C,2,C,ϕ,- C,3,B,C,-	C,2,C,ϕ,-	-	C,6,E,A,-	C,4,A,C,-
D,5,C,D,- D,6,B,C,-	D,5,C,D,-	D,3,D,ϕ,-	-	D,7,A,C,-
E,7,C,D,- E,8,B,C,-	E,7,C,D,-	E,5,D,E,-	E,2,E,ϕ,-	-

b) TO table after step 3 of CTT algorithm

A	B	C	D	E
-	A,9,C,D,-	A,7,D,E,-	A,4,E,A,-	A,2,A,ϕ,-
B,1,B,ϕ,- B,10,C,D,-	-	B,8,D,E,-	-	B,3,A,B,-
C,2,C,ϕ,- C,3,B,C,-	C,2,C,ϕ,-	-	C,6,E,A,-	C,4,A,C,-
D,5,C,D,- D,6,B,C,-	D,5,C,D,-	D,3,D,ϕ,-	-	D,7,A,C,-
E,7,C,D,- E,8,B,C,-	E,7,C,D,-	E,5,D,E,-	E,2,E,ϕ,-	-

c) TO table after step 4 of CTT algorithm

A	B	C	D	E
-	A,9,C,D,-	A,7,D,E,-	A,4,E,A,-	A,2,A,ϕ,-
B,1,B,ϕ,-	-	B,8,D,E,-	-	B,3,A,B,-
C,2,C,ϕ,- C,3,B,C,-	C,2,C,ϕ,-	-	C,6,E,A,-	C,4,A,C,-
D,5,C,D,- D,6,B,C,-	D,5,C,D,-	D,3,D,ϕ,-	-	D,7,A,C,-
E,7,C,D,- E,8,B,C,-	E,7,C,D,-	E,5,D,E,-	E,2,E,ϕ,-	-

2) Second round

a) TO table does not change its contents after step 2 of CTT algorithm

b) TO table after step 3 of CTT algorithm

TABLE I

THE FROM TABLE

A	B	C	D	E
-	A,1,B,-	A,2,C,-	A,5,C,-	A,7,C,-
B,9,C,-	-	B,2,C,-	B,5,C,-	B,7,C,-
C,7,D,-	C,8,D,-	-	C,3,D,-	C,5,D,-
D,4,E,-	D,5,E,-	D,6,E,-	-	D,2,E,-
E,2,A,-	E,3,A,-	E,4,A,-	E,7,A,-	-

TABLE II

THE TO TABLE

A	B	C	D	E
-	A,9,C,-	A,7,D,-	A,4,E,-	A,2,A,-
B,1,B,-	-	B,8,D,-	B,5,E,-	B,3,A,-
C,2,C,-	C,2,C,-	-	C,6,E,-	C,4,A,-
D,5,C,-	D,5,C,-	D,3,D,-	-	D,7,A,-
E,7,C,-	E,7,C,-	E,5,D,-	E,2,E,-	-

A	B	C	D	E
-	A,9,C,D,-	A,7,D,E,-	A,4,E,A,-	A,2,A,ϕ,-
B,1,B,ϕ,- B,10,C,D,-	-	B,8,D,E,-	B,5,E,A,-	B,3,A,B,-
C,2,C,ϕ,- C,3,B,C,-	C,2,C,ϕ,-	-	C,6,E,A,-	C,4,A,C,-
D,5,C,D,- D,6,B,C,-	D,5,C,D,-	D,3,D,ϕ,-	-	D,7,A,C,-
E,7,C,D,- E,8,B,C,-	E,7,C,D,-	E,5,D,E,-	E,2,E,ϕ,-	-

c) TO table after step 4 of CTT algorithm

A	B	C	D	E
-	A,9,C,D,-	A,7,D,E,-	A,4,E,A,-	A,2,A,ϕ,-
B,1,B,ϕ,-	-	B,8,D,E,-	B,5,E,A,-	B,3,A,B,-
C,2,C,ϕ,- C,3,B,C,-	C,2,C,ϕ,-	-	C,6,E,A,-	C,4,A,C,-
D,5,C,D,- D,6,B,C,-	D,5,C,D,-	D,3,D,ϕ,-	-	D,7,A,C,-
E,7,C,D,- E,8,B,C,-	E,7,C,D,-	E,5,D,E,-	E,2,E,ϕ,-	-

For the first example, the algorithms FROM and TO in [1] use four rounds and three rounds, respectively, to form the stable FROM table and TO table.

The stable FROM table getting from the FROM algorithm and the TO table from the TO algorithm are shown in Table I and II:

Although the CFT algorithm also uses four rounds to construct the stable FROM table, the FROM table maintains two paths for a node from node A which has two outgoing links. The TO table needs two rounds to be stable with the CTT. In step 3 of the CTT, a self-looped entry $e = (B, 10, C, D, -)$ will be introduced to node A's TO table. This entry indicates the path (A, C, D, E, A, B) which has a cycle from A to A. The loop-free check procedure in step 4 is executed to find this loop entry: Node A will send a package (A, e) to B, then B searches its FROM table for entry $(A, 10, C, D, -)$. Since such an entry cannot be found in B's FROM table, B will send a message to A indicating that e is a self cycled entry. When A receives this message, it will remove e from the table. After the TO table is stable, node A will keep two paths to all other nodes [1] in its TO table since A has two outgoing links.

As a second example, we add one more link (D, A) to the previous example, shown in Figure 3(b). In this example, node A has two incoming links and two outgoing links.

[1]There is only one path from A to B because another path contains a self cycle and is removed from A's TO table.

The FROM table generated by the FROM algorithm using the protocol in [1] will be stable after three rounds:

A	B	C	D	E
-	A,1,B,-	A,2,C,-	A,5,C,-	A,7,C,-
B,6,C,-	-	B,2,C,-	B,5,C,-	B,7,C,-
C,4,D,-	C,5,D,-	-	C,3,D,-	C,5,D,-
D,1,A,-	D,2,A,-	D,3,A,-	-	D,2,E,-
E,2,A,-	E,3,A,-	E,4,A,-	E,7,A,-	-

The TO table generated by the TO algorithm also needs after three rounds to be stable:

A	B	C	D	E
-	A,6,C,-	A,4,D,-	A,1,A,-	A,2,A,-
B,1,B,-	-	B,5,D,-	B,2,A,-	B,3,A-
C,2,C,-	C,2,C,-	-	C,3,A,-	C,4,A,-
D,5,C,-	D,5,C,-	D,3,D,-	-	D,7,A,-
E,7,C,-	E,7,C,-	E,5,D,-	E,2,E,-	-

The FROM table generated by the CFT algorithm needs four rounds to reach stable:

A	B	C	D	E
-	A,1,B,ϕ,-	A,2,C,ϕ,- A,3,B,C,-	A,5,C,D,- A,6,B,C,-	A,7,C,D,- A,8,B,C,-
B,6,C,D,-	-	B,2,C,ϕ,-	B,5,C,D,-	B,7,C,D,-
C,4,D,A,-	C,5,D,A,-	-	C,3,D,ϕ,-	C,5,D,E,-
D,1,A,ϕ,- D,4,E,A,-	D,2,A,B,- D,5,E,A,-	D,3,A,C,- D,6,E,A,-	-	D,2,E,ϕ,-
E,2,A,ϕ,-	E,3,A,B,-	E,4,A,C,-	E,7,A,C,-	-

The TO table generated from the CTT algorithm after four rounds

A	B	C	D	E
-	A,6,C,D,-	A,4,D,A,-	A,1,A,ϕ,- A,4,E,A,-	A,2,A,ϕ,-
B,1,B,ϕ,-	-	B,5,D,A,-	B,2,A,B,- B,5,E,A,-	B,3,A,B,-
C,2,C,ϕ,- C,3,B,C,-	C,2,C,ϕ,-	-	C,3,A,C,- C,6,E,A,-	C,4,A,C,-
D,5,C,D,- D,6,B,C,-	D,5,C,D,-	D,3,D,ϕ,-	-	D,7,A,C,-
E,7,C,D,- E,8,B,C,-	E,7,C,D,-	E,5,D,E,-	E,2,E,D,-	-

In this example, algorithms CFT and CTT need one more round to construct the stable FROM table and TO table compared with the algorithm used in [1] because the tables have to gather more information from every other node to construct the multiple paths. And two self-cycled entries will be generated by the CTT algorithm: $(B, 7, C, D, -)$ in node A's TO table and $(E, 8, A, C, -)$ in node D's TO table. These entries correspond to the path (A, C, D, A, B) and $(D, A, C, D, E,)$ respectively. These looped entries can also be removed by executing the loop-free check procedure. Note that when the number of nodes that have multiple outgoing links increases, the rounds for a stable table and the self-cycled entries will also increase.

IV. DISCUSSION

The FROM table generated will remain stable if the topology of the network does not change as time goes on. But for the dynamic network, the link between two nodes may be broken or the cost of the link may increase/decrease due to the dynamic physical environment. These changes will affect the cost of a link, and hence, the corresponding value in the table. We consider the following two cases:

Link failures: When a link is broken, the timer for this link will expire and the entry will be deleted from the table. In example 2 (Figure 3(b)), let us suppose that link (D, A) is broken. After some time, node A will delete the expired entries of its FROM table: $(B, 6, C, D, -)$, $(C, 4, D, A, -)$, $(D, 1, A, \phi, -)$ and then insert updated entries: $(B, 9, C, D, T)$, $(C, 7, D, E, T)$. Nodes B and C's FROM tables will also update their corresponding entries when node A updates its FROM table. If link (D, A) does not come back up again, the FROM table of example 2 will finally be the same as the table of example 1.

Link cost changes: Besides all the fields of FROM and TO tables in [1], there is a new added field $NX2$ in both FROM and TO tables. This field is used to deal with the problem of link-cost changes. In these algorithms, under the condition ($e'.DN = e.DN$ and $e'.XN1 = e.XN1$), the old entry will be replaced by the new one if $e'.XN2 = e.XN2$. For the example 2, suppose that the cost of link (D, A) changes its cost from 1 to 3, the entries of the FROM table in nodes A, B and C will be updated to the following:

A	B	C
-	A, 1, B, ϕ, -	A, 2, C, ϕ, - A, 3, B, C, -
B, 8, C, D, -	-	B, 2, C, ϕ, -
C, 6, D, A, -	C, 7, D, A, -	-
D, 3, A, ϕ, - D, 4, E, A, -	D, 4, A, B, - D, 5, E, A, -	D, 5, A, C, - D, 6, E, A, -
E, 2, A, ϕ, -	E, 3, A, B, -	E, 4, A, C, -

If the cost of link (D, A) increases its value to 5, the entry $(B, 6, C, D, -)$ of node A will be first updated to $(B, 10, C, D, -)$ when node A receives the FT packet from node D, and then this entry will update its value again to $(B, 9, C, E, -)$ when A receives another FT packet from node E and finds a shorter path (B, C, D, E, A) from B to A. Correspondingly, $(C, 4, D, A, -)$ and $(D, 1, A, \phi, -)$ in node A's FROM table will also be updated. The final FROM table of the graph is as follows:

A	B	C	D	E
-	A,1,B,ϕ,-	A,2,C,ϕ,- A,3,B,C,-	A,5,C,D,- A,6,B,C,-	A,7,C,D,- A,8,B,C,-
B,9,C,D,-	-	B,2,C,ϕ,-	B,5,C,D,-	B,7,C,D,-
C,7,D,E,-	C,8,D,E,-	-	C,3,D,ϕ,-	C,5,D,E,-
D,4,E,A,- D,5,A,ϕ,-	D,5,E,A,- D,6,A,B,-	D,6,E,A,- D,7,A,C,-	-	D,2,E,ϕ,-
E,2,A,ϕ,-	E,3,A,B,-	E,4,A,C,-	E,7,A,C,-	-

The proposed algorithm can keep multiple paths in both FROM and TO tables. The storage requirement for the table is $O(|E|)$, where $|E|$ is the number of links in the graph, since each outgoing link of a node will have a corresponding entry in the table. The complexity of this protocol will be much higher than the protocol in [1] for the dense network topology because in the dense network, $O(|E|) = O(|V|^2)$. But for the sparse network, $O(|E|) = O(|V|)$ and the protocol benefits from keeping multiple paths in a routing table with reasonable cost.

The merit of keeping multiple paths in a routing table is that it can greatly improve the tolerance of the network link failures without any package exchanges. When the primary shortest path in the routing table fails, an alternate path will be chosen without invoking the path finding process. In the dense network, each node may have many outgoing links to its neighbors, therefore, each pair of two nodes may normally have bidirectional links, or have a very short path with just one or two intermediate nodes. Keeping a path for every outgoing link of a node won't be an effective solution. In sparse networks, the path between two nodes will consist of more nodes, and thus, the path is more easily broken. If each node has just a few adjacent links, the cost of keeping multiple paths in the table will be reasonable. Multiple paths can also be used for a specific node to balance the data stream through different paths. Therefore, keeping more than one path in the routing table is more beneficial than just keeping the shortest path in the routing table.

V. Conclusions

In this paper, we have proposed a multi-path routing protocol for unidirectional networks. The protocol has extended the previous protocol by keeping one path for each outgoing link of the node in the table so as to providing multiple paths in a node's routing table. Each path for a specific outgoing link is the shortest path from that node to the destination through that outgoing link. The protocol is expected to converge more rapidly compared with the previous one in [1]. With the complexity of size $O(|E|)$, the protocol will have good performance in the sparse network.

References

[1] G.H. Chen, F. C.M. Lau, L. Xie, *A Distance-Vector Routing Protocol for Networks with Unidirectional Links*, Computer Communications, Vol. 23, No. 4, February 2000.

[2] C. Hedrick, *Routing Information Protocol*, Internet Request for Comments 1058, June 1988.

[3] J. Moy, *OSPF Version 2*, Internet Request for Comments 1247, 1991.

[4] R. Prakash, *Unidirectinal Links Prove Costly in Wireless Ad Hoc Networks*, Proc. of DIMACS Workshop on Mobile Networks and Computers, 1999.

[5] C. Perkins, P. Bhagwat, *Highly Dynamic Destination Sequenced Distance Vector Routing (DSDV) for Mobile Computers*, Proc. of ACM SIGCOMM Conf. on Communication Architectures, Protocols and Applications, pp 234-244, August 1994.

[6] C. Perkins, *Ad hoc On Demand Distance Vector(AODV) Routing*, internet draft, draft-ietf-manet-aodv-02.txt, November 1998.