

A Distance-Vector Routing Protocol for Networks with Unidirectional Links*

Hao Huang¹, Guihai Chen^{1,2}, Francis C.M. Lau², and Li Xie¹

¹Department of Computer Science, Nanjing University, China

²Department of Computer Science & Information Systems,
The University of Hong Kong, Hong Kong

March 1999

Abstract

We propose a simple distance-vector protocol for routing in networks having unidirectional links. The protocol can be seen as an adaptation for these networks of the strategy as used in the popular RIP protocol. The protocol comprises two main algorithms, one for collecting “from” information, and the other one for generating and propagating “to” information. Like the RIP protocol, this one can handle dynamic changes and tolerate node and link failures in the network.

Keywords: routing algorithm, distance vector, directed graph, unidirectional link, Routing Information Protocol.

1 Introduction

Current Internet routing protocols stand on the assumption that any links between two neighboring nodes are bidirectional. Unidirectional links (UDLs), however, are emerging and could be as ubiquitous as bidirectional links in future networks. Examples of systems having UDLs include direct broadcast systems (DBSs) using satellite [11] and mobile radio networks [10]. Impacts of unidirectional links on routing protocols are discussed in [3] and [10]. According to Ernst and Dabbous [3], adapting current protocols for bidirectional networks so that they also work for networks with UDLs (NUDLs) does not seem to lead to good

*Correspondence: F.C.M. Lau, Department of Computer Science & Information Systems, The University of Hong Kong, Pokfulam Road, Hong Kong / Email: fcmlau@csis.hku.hk / Fax: (+852) 2559 8447.

results. They discussed ways to adapt existing protocols for unidirectional networks. For distance-vector protocols, they gave an example of handling a simple unidirectional network. But a complete protocol for the general case is not given or discussed. New protocols need to be designed for networks having UDLs. We propose such a protocol in this paper.

The protocol we propose is new in the sense that it is not a direct modification of any existing protocol. Nevertheless, it uses a strategy similar to that of the distance-vector-based Routing Information Protocol (RIP) [4, 6], the well-known internet protocol that is widely deployed in internetworks as well as single networks. The RIP strategy is to have every node periodically send its routing table to all its immediate neighbors who would then use the received table to update or improve their own tables. We adopt the RIP strategy, but not the RIP protocol itself because it does not work for UDLs. Although newer protocols for interior gateway routing (IGPs) have been introduced, such as the Open Shortest Path First (OSPF) protocol [8, 9], RIP still maintains its edge in certain aspects. In small networks, RIP has very little overhead in terms of bandwidth used and configuration and management time, and is amenable to distributed, asynchronous operation. RIP is also very easy to implement, especially in relation to the new IGPs. RIP has also been widely adopted by many personal computer manufacturers for use in their networking products, including Xerox, Apple, Novel, 3Com, Ungermann-Bass, Banyan, and Cisco. It is expected that RIP will continue to enjoy its popularity for some period of time.

In adopting the RIP strategy, we realize that in an NUDL, exchanges of routing tables between neighboring nodes become difficult because information can only flow in one direction along a UDL. We also realize, however, that if the network is strongly connected, there must exist a path—a series of UDLs—that goes the other way connecting exactly the same two nodes. This latter path is sometimes referred to as a “back channel” between two adjacent nodes in the literature on NUDLs. Our protocol finds this back channel for a pair of adjacent nodes dynamically during runtime. In contrast, the modified RIP discussed in [1] and [2] depends on the existence of a pre-arranged back channel between a pair of adjacent nodes.

Clearly, in a strongly connected NUDL, between every pair of nodes there must exist at least one circuit—*i.e.*, a cycle of nodes. Each of the two nodes should be able to reach the other node by following this circuit. Any protocol that is going to work for NUDLs can very much be viewed as a circuit discovery protocol. Ernst and Dabbous have proposed such a circuit discovery protocol [3]. Using their protocol, each node maintains and manages a set of circuits; multiple circuits could exist for the same destination node. A simple procedure can

then be used to generate the necessary routing table from the circuits. They gave, however, only a high-level overview of the protocol. Considering the fact that even a small graph could have many cycles, it is important that the protocol would select only the best or appropriate circuits to keep in a node. It is not clear how this is exactly done using their protocol. We find it difficult therefore to give an evaluation of the protocol in terms of its correctness, and bandwidth and space requirements.

On the other hand, the protocol described in this paper attempts only to discover one circuit for every pair of adjacent nodes connected by a UDL. The circuit gives the receiving node a back channel for propagating its routing information to the feeding node. The circuit represents also the best routes from the feeding node to all the other nodes on the circuit.

The protocol recently proposed by Prakash and Singhal [10] in the context of mobile ad-hoc networks requires maintaining in every node a table of size $O(n^2)$, where n is the number of nodes in the network, and the table needs to be sent around from time to time. In contrast, the protocol we propose requires $O(n)$ space in a node, and the messages the nodes send out are of size $O(n)$.

We present the protocol in Sections 3 and 4. The discussion in these two sections assumes that the networks have only UDLs—that is, *unidirectional networks* or *UDNs*. Section 5 then shows that the protocol works also for networks having both UDLs and bidirectional links, also known as *archipelagos* where clusters of bidirectional links are like “islands” in a large space.

2 Preliminaries

Given a UDN of n nodes, the problem is to generate in each node a routing table containing an entry for each of the other $n - 1$ nodes in the network. We consider dynamic networks where nodes and links may go down occasionally, the topology may change, and *link costs* may vary from time to time. The protocol must strive to produce routing tables that represent a good approximation of the current situation of the network.

Every node is assumed to have a unique identity, represented by a capital letter. We denote a UDL that joins node P to node Q by (P, Q) ; P is Q 's *from-neighbor* or *f-neighbor* and Q is P 's *to-neighbor* or *t-neighbor*. Similarly, a path from A , to B , and so on, that ends at X is written as (A, B, \dots, X) .

Each link has a link cost which we assume is known at all times to the two nodes connected

by this link. The link cost of (P, Q) is denoted by $d(P, Q)$.

Each node maintains two tables, a *FROM table* and a *TO table*, abbreviated *FT* and *TT*, respectively. As this is a distance-vector protocol, the entries in both tables capture the distances between this node and the other nodes. *Distance* here actually refers to link cost. The distance between two adjacent nodes P and Q is $d(P, Q)$; the distance between A and B with respect to a certain path is the sum of the costs of those links making up the path. A node's FROM table represents the node's current view of the shortest paths from various nodes to this node, and its TO table the node's current view of the shortest paths from this node to various other nodes. Note that it is possible that none of these is a true shortest path because knowledge takes time to travel in a distributed system. Only the TO table is used in actual routing; the FROM table is for a node's internal use.

This is the format of a FROM (table) entry:

$$\{ND, DT, NX, TTL\}.$$

This entry represents a path from some node, ND , to this node; the distance is DT , and the second node following ND in this path is NX , shown graphically in Fig. 1(a). *TTL* (Time To Live) is the timer value associated with this entry. When *TTL* drops to zero, the entry would be deleted. An entry, when created or when “renewed,” is given a *TTL* value of T . The format of a TO (table) entry is similar, except where ND is the destination node, *i.e.*, the last node of the path, and NX is the next node following *this* node, as shown in Fig. 1(b). A field value “—” means that the field is not important in the discussion.



Figure 1: (a) A FROM entry. (b) A TO entry.

In the following, we use the object-oriented “dot” notation to represent a node's tables, table entries, and fields within an entry. For example, $P.TT$ is P 's TO table; $e.NX$ is the NX field of the entry e .

Periodically, every node packs its FROM table in a packet, a *FROM packet*, and sends it to every t-neighbor. Every so often, a certain event (to be discussed in the next two sections) in a node would trigger the node to send its TO table in a *TO packet* to a certain f-neighbor

through some back channel. Note that if P sends out a FROM packet F , it is not always true that $F.FT$ would be equal to $P.FT$ in contents in the next moment.

3 Construction and Maintenance of FROM Tables

A node's FROM table collects information about the paths that join other nodes in the network to this node. It is essential in the construction of the node's routing table, the TO table.

We say that two entries, p and q , *match*, if $p.ND = a.ND$.

Algorithm FROM:

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 every entry $f \in F.FT$, Q generates an entry, $e = \{f.ND, f.DT+d(P, Q), f.NX, T\}$ and executes the procedure:


```

          IF there exists a matching entry  $e' \in Q.FT$ 
            IF  $(e.NX = e'.NX)$  or  $(e.DT \leq e'.DT)$  replace  $e'$  by  $e$ ;
          ELIF  $e.ND \neq Q$ 
            add  $e$  to  $Q.FT$ ;
          
```
 - (b) Q then generates the entry $e = \{P, d(P, Q), Q, T\}$ and executes the above procedure for this entry.
4. When the timer of a FROM entry expires, the entry is deleted.

Note that the above algorithm ignores two kinds of entries (*i.e.*, the e above) that can be derived from the FROM packet: (1) entries representing alternative paths but whose DT is longer than the DT of existing matching entries, and (2) the entry that represents a self-cycle (from Q back to Q).

Instead of a formal proof, we use an example to demonstrate the correctness of the algorithm. The example is as shown in Fig. 2(a), where the number on a link is the link's cost. After several initial rounds of message exchanges using the above algorithm, the FROM tables of the various nodes have entries that are as shown in Table 1.

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
—	<i>A</i> , 1, <i>B</i> , —	<i>A</i> , 2, <i>C</i> , —	<i>A</i> , 5, <i>C</i> , —	<i>A</i> , 7, <i>C</i> , —
<i>B</i> , 9, <i>C</i> , —	—	<i>B</i> , 2, <i>C</i> , —	<i>B</i> , 5, <i>C</i> , —	<i>B</i> , 7, <i>C</i> , —
<i>C</i> , 7, <i>D</i> , —	<i>C</i> , 8, <i>D</i> , —	—	<i>C</i> , 3, <i>D</i> , —	<i>C</i> , 5, <i>D</i> , —
<i>D</i> , 4, <i>E</i> , —	<i>D</i> , 5, <i>E</i> , —	<i>D</i> , 6, <i>E</i> , —	—	<i>D</i> , 2, <i>E</i> , —
<i>E</i> , 2, <i>A</i> , —	<i>E</i> , 3, <i>A</i> , —	<i>E</i> , 4, <i>A</i> , —	<i>E</i> , 7, <i>A</i> , —	—

Table 1: FROM tables at stable state.

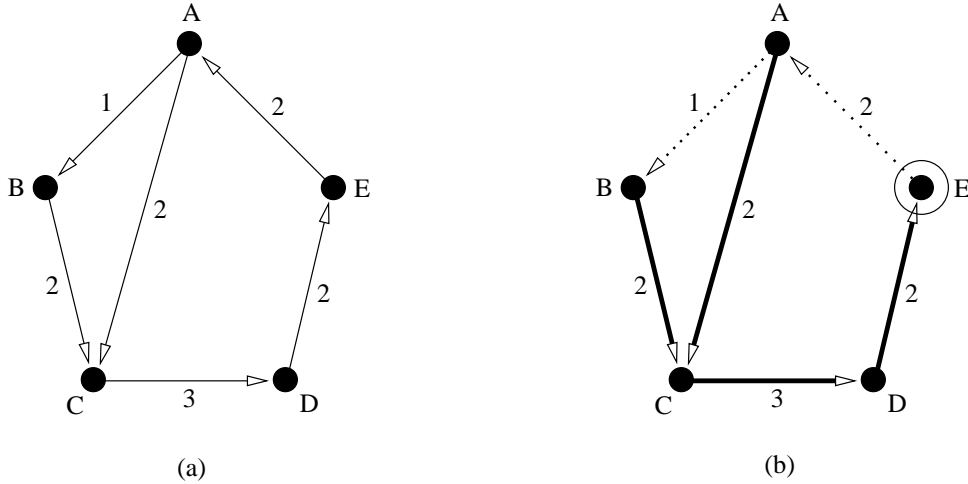


Figure 2: (a) A simple UDN. (b) The tree at *E*.

One can easily verify that these tables are in fact what comes out of executing the algorithm several times after startup. It can also be easily seen that all the entries in a node's FROM table represent shortest distances from all the other nodes to this node. This is due to the " \leq " condition in the "procedure" that prunes away the longer paths. For instance, suppose that node *D* first learned of node *A* through *B* and *C*, and so it has the entry $\{A, 6, B, -\}$ in its table; later on *D* receives the entry $\{A, 2, C, -\}$ in a FROM packet from *C*; then the condition applies, and the entry $\{A, 6, B, -\}$ is replaced by $\{A, 5, C, T\}$ in *D.FT*.

The tables given above represent a *stable state* of the system which will not change as time goes by as long as the network remains stationary. If in fact the network is a stationary

one, it is not necessary that entries be kept alive periodically. But if the network is a dynamic one so that links and nodes might be deleted or added and link costs might change over time, then we need those little devices, in particular the timers, in the algorithm to make sure that the table entries reflect or are a close approximation to the current situation at all times. Let's look at how the algorithm responds to the following possible situations, using again Fig. 2(a) as the example.

- **Node and link failures.** When a node fails, all its incident links become inoperational. Therefore, if the algorithm can handle (single) link failures, it can handle also node failures. Suppose the link (A, C) in Fig. 2(a) goes down. After a while, some of C 's timers expire, and the entry $\{A, 2, C, -\}$, as well as $\{D, 6, E, -\}$ and $\{E, 4, A, -\}$, in $C.FT$ are deleted, according to Step 4 of the algorithm. D and E , however, still think that A can reach them through (A, C) . But this knowledge will soon be proven wrong. D will learn of the fact that (A, C) is broken when its entry $\{A, 5, C, -\}$ expires and gets deleted. Subsequently, E 's entry $\{A, 7, C, -\}$ will also be deleted. The deleted entries, $\{A, 2, C, -\}$, $\{A, 5, C, -\}$, and $\{A, 7, C, -\}$, will soon be replaced by $\{A, 3, B, T\}$, $\{A, 6, B, T\}$, and $\{A, 8, B, T\}$, respectively, if (A, C) does not come back up soon enough.

We can see that the algorithm ignores all incoming entries that are worse in terms of distance than their matching entries in the local FROM table. Therefore, the entry $\{A, 6, B, -\}$ that comes to D via C would not stop the entry $\{A, 5, C, -\}$ in $D.FT$ from expiring. These two entries match, but the latter is obviously better than the former.

We should also point out the reason for the equality part of the " \leq " in Substep 3(a). Suppose for a moment that the cost of (B, C) is 1 instead of 2. Then, when the entry $\{A, 5, B, -\}$ from C shows up in D , it would immediately replace D 's entry $\{A, 5, C, -\}$ instead of having to wait for $\{A, 5, C, -\}$ to expire.

- **Changes of link costs.** A decrease in a link's cost presents no problem as the link will be treated like any other newly discovered cases and will be appropriately handled by Step 3 of the algorithm. An increase in a link cost's, on the other hand, is handled by the condition $(e.NX = e'.NX)$ in Step 3. For example, if the cost of (C, D) in Fig. 2(a) is changed from 3 to 4, D will update its entry $\{C, 3, D, -\}$ to $\{C, 4, D, -\}$ because of the above condition. When this new entry is propagated to E later on—*i.e.*, $e = \{C, 6, D, -\}$ —it replaces E 's $\{C, 5, D, -\}$.

It should now be clear from the examples just presented that timers are indeed essential in a dynamic environment, and why it is necessary to send the entire FROM table to every t-neighbor from time to time.

In Algorithm FROM, although the entry with $ND = Q$ is ignored as far as updating Q 's FROM table is concerned, it is used to trigger the execution of the other algorithm, Algorithm TO, which will be discussed in the next section.

Every entry in a node's FROM table contains a link (that highlighted in Fig. 1(a)), which is the first link in the path represented by the entry. Interestingly, if we connect all these links together, and resulting paths would form a tree.

Proposition 1 *The entries of a node's FROM table at any time form a tree.*

Proof: Given any node, P , the tree in question is rooted at P ; the branches are the different paths embedded in the table. All the tree edges (links) are pointing backwards towards the root.

Assume the contrary that $P.FT$ contains a non-tree. A non-tree has either (a) a cycle in which there is a node having more than one parent, or (b) an "orphan" branch which does not lead to P , or both. Either case is not possible:

- (a) A node, say X , having more than one parent would have more than one entry in $P.FT$ with $ND = X$, but clearly, by the procedure in Substep 3(a) of Algorithm FROM, there cannot exist two or more matching entries in a FROM table.
- (a) Consider an orphan branch that terminates at a node, say X , $X \neq P$ —that is, there is no entry in $P.FT$ with its $ND = X$, while all the other nodes in this branch have their entries in $P.FT$. For this to be possible, an entry with $ND = X$ must not have been propagated to P before. This could not have happened because by Algorithm FROM and by (a) above, entries for the other nodes in the branch must have been propagated to P through X , and X must have sent along all of its own entries including an entry with $ND = X$.

□

We refer to this tree we just discussed as a *FROM tree*. Fig. 2(b) shows E 's FROM tree at stable state.

When E sends its FROM table, *i.e.*, the tree in Fig. 2(b), to its t-neighbor A , we have a cycle (a closed circuit) that joins A to C , D , E , and then back to A . What A receives is

in fact E 's latest view of the shortest paths from A to all the other nodes (C , D , E) in the circuit. This is useful routing information that A should make use of. As we will see in the next section, A would incorporate this information in its TO table. On the other hand, A could treat the path to E as found in the circuit as its “virtual link” or “back channel” to E . A can use the source routing method to send packets to E via this back channel.

Note that the path (from A to E) just mentioned represents also E 's view of the shortest path from any node (other than A) in the path to any subsequent node in the path, for example, from C to E . A could take the trouble of passing this information to all these nodes in the path for incorporating in their TO tables. This is not necessary, however, as every one of these nodes would have received or will receive a similar packet from its f-neighbor like what has happened to A .

We assume that the nodes in the network are capable of performing source routing. The following algorithm will be useful in the construction and maintenance of TO tables.

Algorithm SOURCE_ROUTE:

1. Assume that node Q has just received a FROM packet F from its f-neighbor P and $F.FT$ contains an entry with $ND = Q$. By Proposition 1, $F.FT$ contains a (shortest) path from Q to P . To perform source routing from Q to P , Q encodes this path in the packet(s).

4 Construction and Maintenance of TO Tables

A node's FROM table is not a routing table because an entry there indicates how some other node may send packets to this node, but not how this node may do so to that node. The latter information is to be found in the TO table. The TO table is the routing table. The following constructs and maintains the TO table. Note that when a circuit is found in a node, to follow the RIP strategy, the node would send out its routing table to its f-neighbor; this is done in Substep 2(c) in the algorithm below.

Algorithm TO:

1. Initially, all TO tables are empty.
2. When a node Q receives a FROM packet F from a f-neighbor P , if $F.FT$ contains an

entry whose $ND = Q$, then

(a) Q traces the path from Q to P in $F.FT$: ($Q = N_0, N_1, N_2, \dots, N_m = P$), $m \geq 1$.

Let e_1, \dots, 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 there exists a matching entry $e' \in Q.TT$
 IF $(e.NX = e'.NX)$ or $(e.DT \leq e'.DT)$ replace e' by e ;
 ELIF $e.ND \neq Q$
 add e to $Q.TT$;

(c) Q sends a TO packet containing its TO table to P using Algorithm SOURCE_ROUTE.

3. When a node P receives a TO packet T from its child Q ,

(a) for each entry t in $T.TT$, it generates an entry, $e = \{t.ND, t.DT + d(P, Q), Q, T\}$ and executes the procedure given above.

(b) P then generates the entry $e = \{Q, d(P, Q), Q, T\}$ and executes the procedure given above.

4. When the timer of a TO entry expires, the entry is deleted.

Substeps 2(a) and 2(b) of the above take the circuit as the new shortest path from Q to any other in the circuit, and try to install that in the TO table. Let's refer to Fig. 2, and let the A there be the Q here. That is, A receives from E a packet containing E 's FROM table. Suppose E 's table is the one as shown in Table 1. A can easily trace the path to be $(N_0 = A, C, D, E = N_3)$, and therefore $e_1 = \{A, 7, C, -\}$, $e_2 = \{C, 5, D, -\}$, and $e_3 = \{D, 2, E, -\}$. Note that there is another entry in the table with $NX = N_1 = C$, $\{B, 7, C, -\}$, but that entry is not part of the traced path. By Substep 2(b), A then generates the following entries for e .

$$\{C, d_1, C, T\}, \quad d_1 = 0 + (7 - 5) = 2$$

$$\{D, d_2, C, T\}, \quad d_2 = 2 + (5 - 2) = 5$$

$$\{E, d_3, C, T\}, \quad d_3 = 5 + (2 - 0) = 7$$

The above two substeps, in fact, can be omitted and the algorithm would still work in coming up with the necessary TO table, but the process will take a much longer time because

entries like $\{X, -, X, -\}$, where X is Q 's t-neighbor in the circuit would have to come a full circle one link at a time via Substep 3(b) of the algorithm in order to finally come to Q .

Substep 3(b) is there in the algorithm because when P receives Q 's TO table, it knows that Q considers (P, Q) the shortest path from P to Q . Imagine if there were another path from P to Q (via other nodes) which has a smaller cost than $d(P, Q)$, Q would have sent its TO table to a different f-neighbor instead of P . So P takes the TO table from Q both as a confirmation of Q 's receipt of P 's FROM packet as well as an indication that (P, Q) (in Q 's view) is a shortest path from P to Q .

We need to argue for the necessity of sending TO tables around because FROM tables that are being sent around seem to already contain a fair amount of TO information, such as that embedded in the circuit. In fact, if the network is a ring, Algorithm TO can stop after Substep 2(b). There is no need for sending any TO table around in a ring. This is easy to see because the circuit we discussed above coincides with the physical ring, and therefore, given a circuit, any node would be able to generate all the necessary TO entries for itself. On the other hand, if the network is not a ring, the FROM tables coming from f-neighbors might not contain all the necessary TO information. Refer to the example in Fig. 2 again, where the network consists of two rings. The FROM table A receives from C does not contain the entry $\{A, -, B, -\}$ and so A cannot deduce from $C.FT$ its path to B . As a result, B must send its TO table to A , which takes place when A sends its FROM table to B and B finds a circuit. The following proposition does not need a proof.

Proposition 2 *The FROM table a node Q receives from its f-neighbor P contains at most one path that goes from Q to P .*

5 Archipelagos

We need to show that the algorithms work for bidirectional links. We can treat a bidirectional link as a pair of UDLs. Therefore, each bidirectional link is a cycle, a *simple cycle* involving only two nodes. Fig. 3 shows two views of an archipelago, where each bidirectional link appears as a simple cycle in the one on the right. In the previous section, we have shown that the algorithm works for cycles having more than two nodes. We give an example below to illustrate that the algorithms also work for simple cycles.

Consider the simplest UDN, as shown in Fig. 4. Initially, both FROM tables are empty. After the first exchange of FROM tables, P and Q each has an entry in their FROM tables:

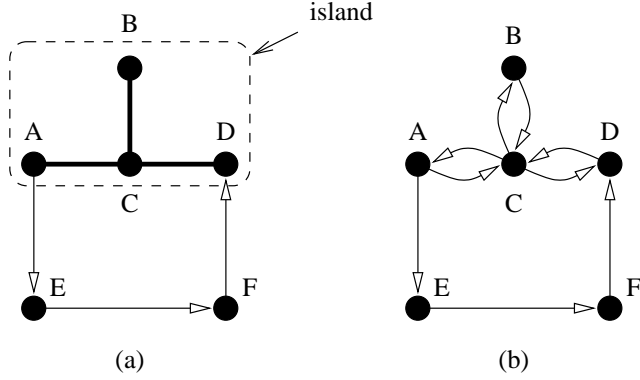


Figure 3: (a) An archipelago. (b) Another view.

$\{Q, -, P, -\}$ and $\{P, -, Q, -\}$, respectively. The next exchange would therefore “close a circuit” for both P and Q . P and Q would insert $\{Q, -, Q, -\}$ and $\{P, -, P, -\}$, respectively, into their TO tables, and then send these tables out to their respective t-neighbors. P , upon receiving Q ’s TO table, would not do anything to its own TO table except renewing the entry $\{Q, -, Q, -\}$; similarly for Q . P and Q will continue to exchange their tables periodically in order to keep their table entries alive.

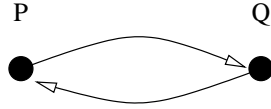


Figure 4: The simplest UDN.

It seems that in the above we are sending two many tables over a single bidirectional edge. Refer to Fig. 3(a), where A, B, C, D , form a bidirectional subnetwork, also called an *island*. For an all-bidirectional network, in fact, it suffices to execute only the FROM algorithm, because TO tables can be derived from FROM tables. The TO algorithm seems superfluous. Therefore, in an archipelago, one approach would be to apply only the FROM algorithm to the islands (thus saving some execution time and communication bandwidth), and apply both the FROM and the TO algorithm to the rest of the network. The difficulty with this approach, however, is that

- in a dynamic environment, links as well as nodes might come and go, and therefore it is not easy to discover and to keep track of the islands;
- the protocol for the boundary nodes, those at the perimeter of an island, could be complicated because they have to play two roles, both as an island node and as a

regular node.

Even if we can treat islands separately, the island nodes cannot get away with passing TO and FROM tables around just like their non-island counterparts. The reason is clear in Fig. 5 where several instances of shortest-path routing are shown. For this simple example of an archipelago having just one island, we see that routing cannot be easily broken into an intra-island component and an extra-island component. For instance, case (a) in the figure has to go through the island even though both the source and destination are outside of the island; case (b) has to go through some non-island edges even though the source and destination are island nodes. Furthermore, because of the shortest-path requirement, we cannot treat an island as one abstract node and allow a routing path to enter, exit, or pass through the island at any boundary node.

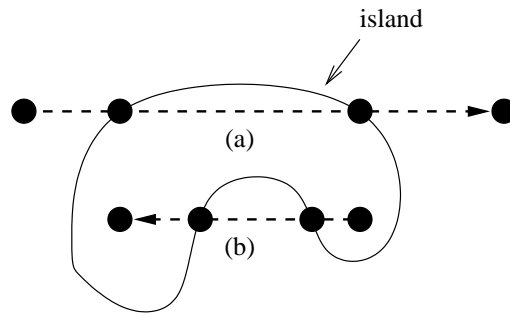


Figure 5: Routing through an island.

6 Concluding Remarks

The RIP protocol is widespread because of its simplicity. It suffers however from convergence problems such as the count-to-infinity problem. As our protocol is modeled after RIP, it could run into similar problems. This can be easily understood if we view the two connections (one UDL plus a back channel) between every two nodes in a UDN as a single bidirectional path. Like in traditional RIP-based networks, the problem can be overcome using techniques such as split horizons and poison reverse updates [5, 7].

The FROM and TO tables have size equal to $O(n)$, where n is the number of nodes in the network, which is also the size of messages that are being sent around periodically over all the edges. Our approach has been a “busy” one because of the need to keep entries alive even after the network has reached a stable state. An alternative would be to send

out messages only when there is a change to the topology or when a link changes cost, and only those affected entries need to be sent around. With such a message-thrifty variant, nodes still need to keep an eye on each other by exchanging control messages periodically and using watchdog timers. These messages, however, would be of a much smaller size than those messages carrying the entire table, and would be independent of the network size.

Another space and bandwidth saving possibility would be to send only one table, the FROM table, and to generate the TO table from the FROM table. This requires maintaining extra edges in a FROM table other than those of the FROM tree. Consider Fig. 2. At some point in time, C must have dropped the entry $\{A, 3, B, -\}$ because $\{A, 2, C, -\}$ was a better entry. If somehow C had kept this entry (marked special) which then got propagated to D , then E , and finally A , A would have more than just the circuit (A, C, D, E, A) but also the path (A, B) for computing new TO entries. We are now in the process of designing an algorithm incorporating both of the above possible improvements.

References

- [1] W. Dabbous, E. Duros, and T. Ernst, "Dynamic Routing in Networks with Unidirectional Links," *Proceedings of the Second International Workshop on Satellite-based Information Services*, Budapest, Hungary, October 1997.
- [2] E. Duros and C. Huitema, Handling of Unidirectional Links with RIP, Internet Draft, INRIA Sophia-Antipolis, March 1996. [<http://www.inria.fr/rodeo/udlr/documents/draft-udlr-rip-00.txt>]
- [3] T. Ernst and W. Dabbous, "A Circuit-based Approach for Routing in Unidirectional Links Networks," INRIA Research Report No. 3292, November 1997.
- [4] C. Hedrick, "Routing Information Protocol," Internet Request for Comments 1058, June 1988.
- [5] C. Huitema, *Routing in the Internet*, Prentice-Hall, 1995.
- [6] G. Malken, "RIP Version 2—Carrying Additional Information," Internet Request for Comments 1723, 1994.
- [7] G. Malken and M. Steenstrup, "Distance-Vector Routing," in *Routing in communications networks*, M. Steenstrup, ed., Prentice-Hall, 1995, 83–157.

- [8] J. Moy, "OSPF Version 2," Internet Request for Comments 1247, 1991.
- [9] J.T. Moy, *OSPF: Anatomy of an Internet Routing Protocol*, Addison-Wesley, 1998.
- [10] R. Prakash and M. Singhal, "Impact of Unidirectional Links in Wireless Ad-Hoc Networks," *Proceedings of DIMACS Workshop on Mobile Networks and Computing*, Rutgers University, NJ, March 1999.
- [11] Y.-G. Zhang and S. Dao, "Integrating Direct Broadcast Satellite with Wireless Local Access," *Proceedings of the First International Workshop on Satellite-based Information Services*, New York, November 1996, 24–29.