

Unidirectional Link-State Routing with Propagation Control

Lichun Bao † J.J. Garcia-Luna-Aceves ‡

† Computer Science Department

‡ Computer Engineering Department

Baskin School of Engineering

University of California, Santa Cruz, California 95064

E-mail: † baolc, ‡ jj@cse.ucsc.edu

Abstract— Unidirectional links can occur in wireless networks and mixed-media networks. However, the vast majority of routing algorithms proposed to date require bidirectional links to operate. We present an efficient link-state routing algorithm, which we call ULPC, that operates with unidirectional links. ULPC is based on the concept of “inclusive cycle” of a link, which is the distance that link-state updates about the link must propagate to ensure correct routing within the network. ULPC incrementally disseminates and selectively utilizes unidirectional link-state information to build correct routing tables. ULPC is verified to be correct. Simulations on a 20-node network with unidirectional links show that ULPC is superior over the traditional link-state routing algorithms relying on topology broadcast.

Keywords— Link-state routing, unidirectional link, inclusive cycle.

I. INTRODUCTION

Routing protocols are typically categorized into distance vector algorithms (DVAs) and link-state algorithms (LSAs). An LSA uses complete [14] or partial [2], [9], [10] network topology to build preferred routing paths. A DVA uses vectors of distances to destinations reported by neighbors to build routing tables [12], [15]. Although many DVAs and LSAs have been proposed, the vast majority assumes networks with bidirectional links, which is a problem, because networks with unidirectional links are found in many communication systems, such as wireless networks and mixed-media networks integrating heterogeneous transmission media. Unidirectional links may exist in a network due to differences in transmission power, code rate, terrain, antennas used, transmission media used among routers, and other reasons. A network with unidirectional links is called a *directed network*. In spite of the need to support unidirectional links in wireless and mixed-media networks, we found limited research on routing in directed networks [1], [3], [11], [16].

Link state algorithms based on topology flooding seemed a viable solution for unidirectional routing [13], [7]. Meanwhile, the IETF working group on unidirectional link routing (UDLR) [5] found another way, which disguises the unidirectional implication of the network through encapsulating and tunneling of IP packets at the link layer. These approaches are limited in that they assume communication networks to be strongly connected and the underlying transmission protocol to be reliable.

Dabbous et al proposed a circuit-based link-state approach for unidirectional routing [6], [8], [17]. To find a route to a destination, a circuit including both source and destination is first detected, and then is validated by sending a validation message along the circuit. If a validation successfully goes through the circuit, a bidirectional communication can thus be established between source and destination using paths in each direction on the circuit. However, when the network grows larger, the number of circuits maintained in the network becomes formidable and the algorithm has to resort to other tools, such as *on-demand routing* to nodes that are far away.

In this paper, we present, verify and simulate a new link-state routing protocol for directed networks, which we call ULPC (unidirectional link-state routing protocol with propagation control). Networks are assumed to have no hierarchical routing. The minimum requirement for a link to be used for packet routing is for it to have an *inclusive cycle* in the network. To discover the inclusive cycle, each link state includes a cycle size property. The link state is propagated in the network as far as the distance from the tail of the link to current node is less than the cycle size. In case the link has an inclusive cycle in the network, the head of the link is able to receive the link state and notice the existence of the downstream link. The cycle size of a link state is initialized to a maximal threshold which causes the link state propagated within a limited distance in the network. When the inclusive cycle is found, hopefully of smaller cycle size, the propagation mechanism further limits the distance that the link state propagates and network resources are saved. If the inclusive cycle breaks, the cycle size of the link state is reset to the threshold, which starts another search.

Section II specifies the network model as well as various concepts and notations used in this paper. Section III describes the basic operations of link vector algorithms used by ULPC. Section IV presents ULPC. Section V addresses its correctness. Section VI presents the results of simulation experiments on a network with 20 nodes, which show that ULPC is more attractive than the traditional link-state routing approach based on topology broadcast.

II. NETWORK MODEL

A network is modeled by a directed graph $G = (V, A)$, where V includes a set of nodes (routers) each with a

TABLE I
NOTATIONS

$u \cdot v$	Physical link $(u, v) \in A$.
$l_{u,v}^i$	Link state of $u \cdot v$ reported by i .
$i \Rightarrow j$	A path from i to j .
$i \mapsto j$	The shortest path from i to j .
$\partial_{u,v}^i$	Inclusive cycle of $u \cdot v$ found at i .
$dp_{u,v}^i$	Discovery path for $u \cdot v$ found at i .
TG_i	The network topology known by node i .
SG_i	Source graph (i.e., the shortest path tree) of node i based on TG_i .
DG_i	Discovery graph, which is part of transposed network topology at node i to find inclusive cycles.
U_i	The set of i 's upstream nodes.
D_i	The set of i 's downstream nodes.
$l \in i \Rightarrow j$	Path from i to j contains link l .
$ u \cdot v $	The cost of link $u \cdot v$.
$ i \Rightarrow j $	The cost of path $i \Rightarrow j$, i.e., $ i \Rightarrow j = \sum_{u \cdot v \in i \Rightarrow j} u \cdot v $.
$p_1 + p_2$	Concatenation of two paths p_1 and p_2 .

unique ID number, and $A \subseteq V \times V$ is the set of directed links. A bidirectional link between two nodes is represented by two unidirectional links. Link $(u, v) \in A$ if and only if v can receive information from u . Node u is called the *head* or *upstream node* of the link and v is the *tail* or *downstream node*. We assume that information propagates through a link with non-zero probability. We also use the notations shown in Table I to represent topologies, data structures, and the operation of ULPC.

The *inclusive cycle of a link* is the smallest cycle that includes the link on its path. The inclusive cycle for link $u \cdot v$ can be denoted as

$$\partial_{u,v} = u \cdot v + v \mapsto u = u \cdot v \mapsto u$$

In case of a bidirectional link, we have $\partial_{u,v} = u \cdot v + v \cdot u = u \cdot v \cdot u$. The *cycle size* of a link is the summation of link costs on the inclusive cycle.

The *discovery path* of a link is the shortest path from the tail of the link to current node. For example, the discovery path for link $u \cdot v$ at node i is the shortest path $v \mapsto i$.

A link state $l_{u,v}$ contains following properties:

$l_{u,v}.h$	ID of head u ;
$l_{u,v}.t$	ID of tail v ;
$l_{u,v}.c$	The cost of link $u \cdot v$;
$l_{u,v}.cs$	Inclusive cycle size;
$l_{u,v}.sn$	Sequence number of the link state;

III. LINK VECTOR ALGORITHMS

The link vector algorithm (LVA) introduced in [9] is a link-state algorithm in which each node maintains a *source graph* (e.g. the shortest path routing tree when using Dijkstra's algorithm) and adjacent links. Upon discovery of a new neighbor (and a new link), the node sends its complete source graph to the neighbor and reports incremental changes of its source graph thereafter. The collection of source graphs reported by the neighbors of a node comprises the topology information about the network. The parameters of a link state can be changed only

by its head node. For a given link, a node also records the set of neighbors that report the link in their source graphs. Link-state updates includes link parameters for the link, and an *operation code* on the link. The operation code is either ADDITION or DELETION, depending on whether a link state is more recent or is no longer used in the source routing tree at the reporting node. When a link state is received with DELETION code, the corresponding neighbor is deleted from the set of reporting neighbors for the link state. If the set of reporting neighbors is empty, the link state is deleted from the topology graph. LVA was shown to greatly reduce communication overheads found in link-state algorithms based on topology broadcast. LVA has been proven to converge within a finite amount of time after any sequence of topology changes.

ULPC adapts the mechanisms used in LVA to directed networks, in which neighbors on a unidirectional link cannot communicate directly. It is only after an inclusive cycle for a link is discovered at both head and tail of the link, that the downstream neighbor can send its complete source graph to its upstream neighbor via the path on the inclusive cycle. Furthermore, as shown in section IV-A.3, the inclusive cycle size is to be determined by the head of the unidirectional link.

IV. ULPC

ULPC consists of three parts: the *neighbor protocol (NBR)*, the *path computation protocol (NET)* and the *retransmission protocol (RET)*. *NBR* provides mechanisms for a node to detect upstream neighbors, update cycle sizes of downstream links, and propagate link states that satisfy Eq. (1) formulated in Section IV-A.2. *NET* computes the source graph (shortest-path tree) of the node based on Dijkstra's algorithm and exchanges routing information between neighbors. *RET* maintains a list of packets and keeps retransmitting upon timeouts, until the node receives acknowledgments for these packets from destinations or destinations stop being neighbors.

A. NEIGHBOR PROTOCOL

NBR maintains three data structures, D_i , U_i and DG_i . U_i contains data for detecting and maintaining incoming (upstream) links. Outgoing (downstream) links are saved in D_i , and can be used in routing as long as their inclusive cycles exist. DG_i is a transposed graph used to find inclusive cycles of upstream links.

A.1 Link Detection

A HELLO packet is periodically broadcasted by a node to inform neighbors of its existence. However, if other packets are sent out during the interval of HELLO packets, the next HELLO packet is suppressed. On periodic detection of packets from a neighboring node, say u , a node i creates a link state $l_{u,i}$ and u becomes one of U_i . Without loss of generality, the cost for an active link is set to 1, thus $l_{u,i}.c = 1$. If the link disappears, $l_{u,i}.c$ is set to ∞ by i .

A.2 Inclusive Cycle

Communicating between a source-destination pair of nodes using unidirectional links in the network, it mandates a directed circuit, which naturally implies an inclusive cycle for each link that is utilized in routing that communication data.

An inclusive cycle is the only way for a link state to propagate back to the upstream node efficiently, so that the upstream node can be informed of whatever happens to the link. Without considering an inclusive cycle in routing decision at the upstream node, the upstream node might keep using an outdated downstream unidirectional link when the inclusive cycle is gone and the new link state can not be propagated back to the upstream node. To indicate the existence of an inclusive cycle for a link, a *cycle size* property is maintained for the link.

ULPC handles the discovery of inclusive cycles in *NBR* using the discovery graph DG_i at node i . Node i propagates a link state to its downstream nodes if the link state satisfies the following formula:

$$|dp_{u,v}^i| < |l_{u,v}.cs| \quad (1)$$

where $dp_{u,v}^i = v \mapsto i$ is the *discovery path* for link $u \cdot v$ at node i , and $l_{u,v}.cs$ is the cycle size property of link $u \cdot v$.

DG_i is represented by:

$$DG_i = \{l \mid \exists u \cdot v \in A(l \in (u \cdot v \mapsto i) \wedge |v \mapsto i| < |l_{u,v}.cs|)\} \quad (2)$$

Therefore, DG_i maintains link states satisfying Eq. (1) and others residing on discovery paths. Whenever there is a change in DG_i , i propagates the change to its downstream nodes.

To find the shortest path term $v \mapsto i$ in DG_i , links are transposed in DG_i , i.e., link $u \cdot v$ is stored as link $v \cdot u$. Then $v \mapsto i$ in (2) is discovered in the shortest path tree by running Dijkstra's algorithm on DG_i .

ULPC sets the cycle size of a newly discovered upstream link to a negative number $-P_{lim}$, which stands for *propagation limit*. According to Eq. (2), the link state with $-P_{lim}$ will only be propagated within a radius of $P_{lim} + 1$ from the tail node. This is important, because we require that a link state have a positive cycle size to be used in routing; therefore, by initializing $l.cs$ to $-P_{lim}$, a link state will not be used in routing until it is propagated back to its head if it has an inclusive cycle of size less than or equal to $P_{lim} + 1$ and the head sets the cycle size property of the link to the actual cycle size, positive.

We set a limit on the cycle size of a link state because of two reasons. First, most links in a network topology have small inclusive cycles even for satellite links. Second, we do not want to utilize a link with large inclusive cycle because the cost of coordinating routing information between neighbors on a unidirectional link is too much.

With *NBR*, the head u of link $u \cdot v$ receives the state of $u \cdot v$ because every node on inclusive cycle $\partial_{u,v}$ will keep propagating $l_{u,v}$ according to (1) until it reaches u .

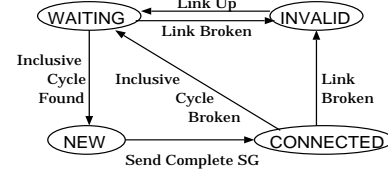


Fig. 1. State Transitions of Upstream Node

A.3 Cycle Size

The cycle size of a link state is decided by the head of the link. Though it seems natural to let the tail of a link solely determine the link state, this may cause problems. As an example, suppose a link $u \cdot v$ comes into existence and its inclusive cycle is discovered by the tail v . Now the link state $l_{u,v}$ has to propagate via the inclusive cycle again to get to the head u . Not until u knows $l_{u,v}.cs > 0$, can u use the link in routing. Accordingly, two iterations through the inclusive cycle are needed for a link to be utilized in routing. Furthermore, suppose the inclusive cycle is broken, and the link also disappears subsequently; it happens that the upstream node still assumes the existence of the link and has no way of recovering from that fault.

The upstream of a link node decides its cycle size for the sake of both efficiency and safety of routing. The scenario of the previous example is: when u finds out the inclusive cycle of its downstream link $u \cdot v$ in DG_u , u updates the link state and propagates according to Eq. (1).

The timeliness of a link state is determined with a sequence number. A problem arises if two sources, head and tail of a link, can change the sequence number of the link state. We solve this problem by letting the downstream node increase sequence number whenever it observes a newer upstream link state from an upstream node.

The following algorithms resolve the sequence number for a link state. Node u detects the inclusive cycle and determines $l_{u,v}.cs$, while node v accepts any change on cycle size made by u for $l_{u,v}$ and increments the sequence number.

Procedure at head u :

```

/* Found  $\partial_{u,v}$  in  $DG_u$ . */
if (  $|\partial_{u,v}^u| \neq l_{u,v}.cs$  ) {
   $l_{u,v}.cs \leftarrow |\partial_{u,v}^u|$ ;
   $l_{u,v}.sn \leftarrow l_{u,v}.sn + 1$ ;
  propagate  $l_{u,v}^u$ ;
}

```

(a)

Procedure at tail v :

```

/* Received  $l_{u,v}^u$  from  $u$ . */
if (  $l_{u,v}.cs \neq l_{u,v}^u.cs$  ) {
   $l_{u,v}.cs \leftarrow l_{u,v}^u.cs$ ;
   $l_{u,v}.sn \leftarrow$ 
   $\max(l_{u,v}.sn, l_{u,v}^u.sn) + 1$ ;
  propagate  $l_{u,v}^v$ ;
}

```

(b)

A.4 Neighbor States

The upstream node table U_i at node i is used to monitor incoming links. An upstream neighbor may be in one of four states as illustrated in Fig. 1. For example, an upstream node $u \in U_i$ in WAITING state indicates the link $u \cdot i$ has been detected by i but is still unnoticed

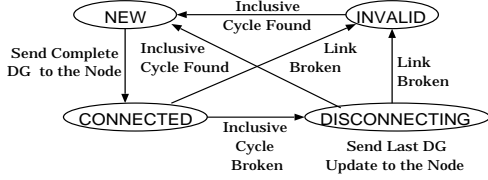


Fig. 2. State Transitions of Downstream Node

by u . The state of u moves from WAITING to NEW once i gets to know that $l_{u,i}.cs$ is set to positive number by u , when i sends its complete source graph SG_i to u . Then u switches from NEW state to CONNECTED state and stays there as long as the inclusive cycle remains ($l_{u,i}.cs > 0$). Changes in SG_i are sent to u reliably if u is in CONNECTED state. If the inclusive cycle is broken, as implied by $l_{u,i}.cs$ set to $-P_{lim}$, u changes from CONNECTED state back to WAITING state. The INVALID state of u means that the link $u \cdot i$ disappears (i.e., $l_{u,i}.c = \infty$).

A downstream node table D_i is also maintained for outgoing links (Fig. 2). A downstream node is INVALID in the beginning, and enters NEW state if the inclusive cycle is found at i , when i sends its complete discovery graph DG_i to that downstream node. Afterwards, the downstream node becomes CONNECTED and incremental changes in DG_i are sent to that downstream node. Outgoing links in CONNECTED state are given to NET for routing. A CONNECTED downstream node becomes DISCONNECTING if the inclusive cycle is broken until another inclusive cycle is found to change it back to NEW state. An outgoing link in state INVALID and DISCONNECTING cannot be used in routing due to uncertainty of the downstream node. Downstream neighbors in CONNECTED state receive discovery graph updates reliably.

Although the head and tail of a link are separated by a possibly large inclusive cycle to maintain their neighbor relationship, we prove that they have an *equivalent* view regarding the validity of the link for routing purposes (Lemma 1), that is, one of two conditions holds for a link $u \cdot v$:

1. u is CONNECTED in U_v and v is CONNECTED in D_u , so that $u \cdot v$ is eligible for routing at u , and v sends SG_v updates to u .
2. Neither u nor v is in CONNECTED state in the neighbor table of the other side, and they do not coordinate topology information.

A.5 Discovery Path Propagation

NBR requires reliable propagation of discovery graph updates to keep correct information about inclusive cycles. We run Dijkstra's algorithm on DG_i to compute the shortest path tree and discover links that satisfy Eq. (1) as well as inclusive cycles of outgoing links from i . If link states that satisfy Eq. (1) or link states that are on discovery paths of these links change, a link-state update packet is generated containing those changes and sent reliably to

its CONNECTED downstream nodes.

Some unknown downstream nodes may also receive such an update. Since incremental updates to DG_i will cause a fragmented view of discovery paths at new downstream nodes, an updated link that satisfies (1) and its discovery path should be packed into the same packet to keep the integrity of a link with its discovery path.

B. PATH COMPUTATION PROTOCOL

A *topological graph* (TG_i) is maintained at each node by NET to compute the source graph SG_i , from which the *routing table* is derived using Dijkstra's algorithm. TG_i of node i is composed of source graphs from its downstream neighbors and the discovery graph (3).

$$TG_i = \left(\bigcup_{k \in D_i} SG_k \right) \cup DG_i \quad (3)$$

DG_i is used for routing by i to send its complete source graph to a NEW upstream node because link states on the inclusive cycle may not be reported by i 's downstream nodes. $SG_k, k \in D_i$ is used for routing to farther nodes.

SG_i of node i is not represented separately from TG_i actually, but indicated by a tag of each link-state structure in TG_i . SG_i is represented in Eq. (4) based on the concept of Dijkstra's algorithm:

$$SG_i = \{ l \mid l = u \cdot v \wedge \exists k \in D_i (i \mapsto v = i \cdot k \mapsto u \cdot v) \wedge (\forall l' \in k \mapsto v, l' \in SG_k \cup DG_i) \} \quad (4)$$

Link-state additions, updates and deletions in source graph will be sent reliably to CONNECTED upstream neighbors. Each link-state entry in an update packet is affixed with an operation code to indicate UPDATE or DELETE operation on that link. The operation code UPDATE means the link is added or updated in the source graph, while DELETE means the link state is deleted from the source graph.

V. CORRECTNESS OF ULPC

Lemma 1: The knowledge about the link at its head and tail is equivalent.

Proof: In ULPC, we consider two attributes of a link state $l_{u,v}, u \cdot v \in A$: link cost $l_{u,v}.c$ and cycle size $l_{u,v}.cs$. We have to decide the equivalence of the link state at both u and v in four cases for the algorithms (a) and (b):

1. $l_{u,v}.c < \infty$ and $\exists \partial_{u,v} \in G$, which means an inclusive cycle exists for the link. Changes to $l_{u,v}$ at u or v can get to each other by *NBR*. Since v accepts $l_{u,v}.cs$ regardless $l_{u,v}.sn$ and always increase $l_{u,v}.sn$, u will accept the new $l_{u,v}^v$. Therefore, u and v will keep the same link state about $u \cdot v$.
2. $l_{u,v}.c < \infty$ and $\partial_{u,v} \notin G$, which means the link exists but the inclusive cycle is broken. Since the broken point will propagate down the path on the inclusive cycle and get to u , u resets the cycle size to

– P_{lim} and propagates to v . *RET* at u will keep sending the new link state to v for limited times (not reliable), so that the new cycle size of the link can get to v .

3. $l_{u,v}.c = \infty$ and $\exists \partial_{u,v} \in G$, which means the link is broken. The link state can be propagated down the inclusive cycle by *NBR* reliably because the link causes *DG* changes. In this case, u gets to know $l_{u,v}^v$ within finite time after disappearance of the link.
4. $l_{u,v}.c = \infty$ and $\partial_{u,v} \notin G$, worst of which is that the link disappears and the inclusive cycle also breaks before $l_{u,v}^v$ is propagated to u . In this case, since u knows exact information about $l_{u,v}.cs$ by *NBR*, v knows about $l_{u,v}.c$, both u and v consider each other invalid neighbor.

It thus follows that equivalence about the link state is maintained. ■

Lemma 2: An upstream node can reliably get the source graph of its downstream nodes in CONNECTED state.

Theorem 3: Any link-state update is either propagated to or discarded by all nodes that use the link for routing.

Proof: Suppose link state $l_{u,v}$ becomes $l'_{u,v}$ at time T , and i uses link $u \cdot v$ in routing before T . We prove that i either updates or discards the link in routing within finite amount of time after T .

A link is used for routing only if the link has finite positive cycle size. Since i uses link $u \cdot v$ for routing, for any link $s \cdot t$ on the shortest path $i \mapsto v = i \mapsto u \cdot v$, $l_{s,t}.cs > 0$ and t reports the remaining path $t \mapsto v$ to s . Because u and v hold equivalent link state of $u \cdot v$ according to *Lemma 1*, we only consider the propagation of $l'_{u,v}$ from u in following two cases after T :

1. $\forall l \in i \mapsto u$, inclusive cycle ∂_l remains. Then $l'_{u,v}$ can be propagated backward by *NET* to i along the path on $\partial_l, l \in i \mapsto u$, and $l'_{u,v}$ is updated at i .
2. Suppose $\exists s \cdot t \in i \mapsto u$, $s \cdot t$ or $\partial_{s,t}$ breaks before $l'_{u,v}$ is propagated back along $\partial_{s,t}$, then i will not receive update $l'_{u,v}$ by *NET*. But according to (4), s stops using t as next-hop to v , and stops using any link state reported by t . Therefore, s sends $l_{u,v}$ with DELETE operation code upwards to i if $u \cdot v$ is unusable by *NET* at s . That is, $\forall k \in i \mapsto s$, the link state of $u \cdot v$ with DELETE operation keeps propagating upward if $u \cdot v$ is unusable by k . Finally i discards $u \cdot v$ in routing.

In any case, the propagation distance is finite, so is the time. So the theorem stands. ■

Theorem 4: Updates on network topology changes stabilize in finite time.

Theorem 5: When ULPC stabilizes, there is no loop in network routing.

VI. SIMULATIONS

ULPC was simulated using the C++ Protocol Toolkit (CPT) [4]. *NBR*, *NET* and *RET* use UDP packets to exchange neighbor packet (NbrPkt), routing information

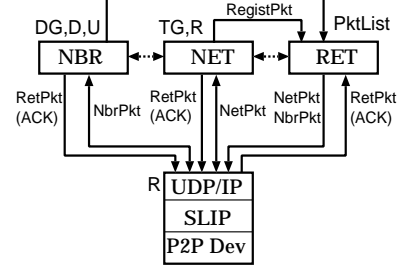


Fig. 3. Program Structure of ULPC

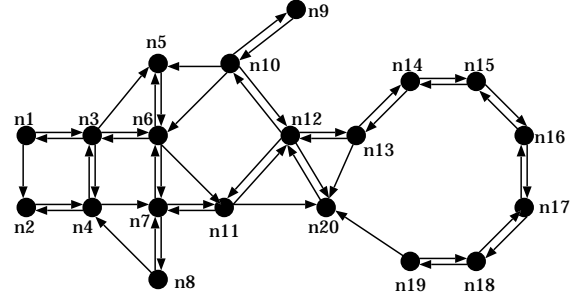


Fig. 4. Topology Graph of a Sample 20-Node Network

packet (NetPkt) and acknowledgment packet (RetPkt) of NbrPkt and NetPkt (see Fig. 3). The physical connections are point-to-point (P2P) links in order to eliminate considerations regarding channel access. The data structures used in each module are also denoted beside each module box in Fig. 3, namely, topology graph (*TG*), routing table (*R*), discovery graph (*DG*), upstream and downstream node tables (*D*, *U*) and the retransmission packet list (PktList), some of which are cross-referenced by different modules.

We also simulated an ideal link-state algorithm based on OSPF [14], called OSPFC (OSPF with propagation control), to compare with ULPC. OSPFC uses the same mechanisms in *NBR* to discover usable link states for routing. Furthermore, OSPFC sends any *reachable* link-state update to its upstream neighbors in CONNECTED state. A link is reachable if the head of the link is reachable through a path that is composed of links with positive cycle size. Changes in reachability of a link state will also generate a link-state update to all CONNECTED upstream neighbors. However, we don't use operation codes in OSPFC link-state update packets because any link-state change is flooded in the network.

Fig. 4 is a sample network with unidirectional links. We tried plot the topology such that a tree connects all nodes with bidirectional links, and traditional routing algorithms can also emulated by ULPC if we set $P_{lim} = 1$. However, if any of these bidirectional links breaks, traditional algorithms no longer works.

Setting $P_{lim} = 3$, we collected statistics about ULPC and OSPFC in three scenarios, namely, link deletions, link cost increases by 1 and node deletions. In each scenario, the number of reachable nodes, update packets in *NET* and

update packets in *NBR* sent by all nodes were summarized at each event, respectively, and the results are plotted in Fig. 5, 6 and 7.

We see that OSPFC sometimes found more reachable nodes (Fig. 5) than ULPC did. This is because discovery graphs in OSPFC take advantage of knowing the complete topology of the network thus gets more inclusive cycles of unidirectional links. In general, ULPC saves one third of the network resources used by OSPFC in *NET* computations (Fig. 7), while they consume similar resources in *NBR* computations (Fig. 6).

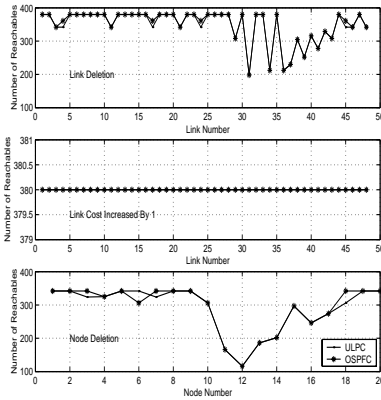


Fig. 5. Number of Reachable Nodes

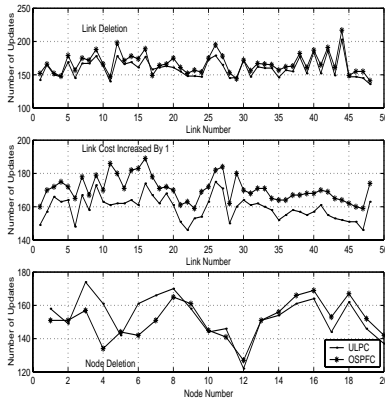


Fig. 6. Number of *NBR* Update Packets

VII. CONCLUSIONS

We have presented a new routing protocol, ULPC, which adapts link vector algorithms to directed networks. ULPC is scalable and provides a promising technique to extend traditional routing algorithms into the regime of arbitrary network topologies. We proved the correctness of ULPC and compared its performance against the distributed topology broadcast approach, OSPFC. The concept of inclusive cycle used in ULPC permits our generalization of link-state algorithms to network with unidirectional links.

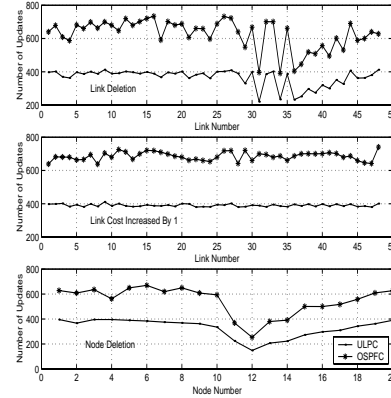


Fig. 7. Number of *NET* Update Packets

REFERENCES

- [1] Yehuda Afek and Eli Gafni. Distributed Algorithm for Unidirectional Networks. *SIAM J. Comput.*, 23(6):1152–1178, Dec. 1994.
- [2] J. Behrens and J. J. Garcia-Luna-Aceves. Hierarchical Routing Using Link Vectors. In *Proc. IEEE INFOCOM 98*, San Francisco, California, March 29-April 2 1998.
- [3] Pomalaza-Raez C.A. A distributed routing algorithm for multihop packet radio networks with uni- and bi-directional links. *IEEE Transactions on Vehicular Technology*, 44(3):579–85, Aug. 1995.
- [4] Rooftop Communications Corp. C++ Protocol Toolkit (CPT), 1997. info@rooftop.com.
- [5] Walid Dabbous, Yongguang Zhang, David Oran, and Rob Coltun. A Link Layer Tunneling Mechanism for Unidirectional Links, April 2000. Internet-Draft, Network Working Group.
- [6] E. Duros and W.Dabbous. Supporting Unidirectional Links in the Internet. In *Proceedings of the First International Workshop on Satellite-based Information Services*, Rye, New York, U.S.A., Nov 96.
- [7] Emmanuel Duros. Handling of unidirectional links with OSPF. Technical report, Network Working Group, Mar. 1996.
- [8] Thierry Ernst and Walid Dabbous. A Circuit-based Approach for Routing in Unidirectional Links Networks. Technical Report ercim.inria.publications/RR-3292, Inria, Institut National de Recherche en Informatique et en Automatique, Oct. 1997.
- [9] J.J. Garcia-Luna-Aceves and J. Behrens. Distributed, Scalable Routing Based on Vectors of Link States. *IEEE Journal on Selected Areas in Communications*, Oct. 1995.
- [10] J.J. Garcia-Luna-Aceves and M. Spohn. Scalable Link-State Internet Routing. In *Proc. IEEE International Conference on Network Protocols (ICNP 98)*. Austin, Texas, Oct. 14–16 1998.
- [11] M. Gerla, L. Kleinrock, and Y. Afek. A distributed routing algorithm for unidirectional networks. In *GLOBECOM '83. IEEE Global Telecommunications Conference*, volume 2(3), pages 654–8, San Diego, CA, USA, 28 Nov.-1 Dec. 1983.
- [12] C. Hedrick. Routing Information Protocol. RFC 1058, Network Working Group, Jun. 1988.
- [13] Robert McCurley and Fred B. Schneider. Derivation of a Distributed Algorithm for Finding Paths in Directed Networks. *Science of Computer Programming*, 6(1):1–9, 1986.
- [14] J. Moy. OSPF Version 2. RFC 2178, Network Working Group, Jul. 1997.
- [15] S. Murthy and J.J. Garcia-Luna-Aceves. An Efficient Routing Protocol for Wireless Networks. *ACM Mobile Networks and Applications Journal, Special issue on Routing in Mobile Communication Networks*, 1996.
- [16] M. Nishida, H. Kusumoto, and J. Murai. Network architecture using network address translation mechanism for network with unidirectional links. *Transactions of the Institute of Electronics, Information and Communication Engineers B-II*, J81B-II(5):458–67, May 1998.
- [17] W.Dabbous, E. Duros, and T. Ernst. Dynamic Routing in Networks with Unidirectional Links. In *Proceedings of the Second International Workshop on Satellite-based Information Services*, Budapest, Hungary, Oct. 1997.