An Extended Fault-Tolerant Link-State Routing Protocol in the Internet

Jie Wu, *Senior Member*, *IEEE*, Fei Dai, *Student Member*, *IEEE*,
Xiaola Lin, *Member*, *IEEE*, Jiannong Cao, *Member*, *IEEE*, and Weijia Jia, *Member*, *IEEE*

Abstract—Link-state routing protocols, such as OSPF and IS-IS, are widely used in the Internet today. In link-state routing protocols, global network topology information is first collected at each node. A shortest path tree (SPT) is then constructed by applying Dijkstra's shortest path algorithm at each node. Link-state protocols usually require the flooding of new information to the entire (sub)network after changes in any link state (including link faults). Narvaez et al. proposed a fault-tolerant link-state routing protocol without flooding. The idea is to construct a shortest restoration path for each unidirectional link fault. Faulty link information is distributed only to the nodes in the restoration path and only one restoration path is constructed. It is shown that this approach is loop-free. However, the Narvaez et al. approach is inefficient when a link failure is bidirectional because a restoration path is unidirectional and routing tables of nodes in the path are partially updated. In addition, two restoration paths may be generated for each bidirectional link fault. In this paper, we extend the Narvaez et al. protocol to efficiently handle a bidirectional link fault by making the restoration path bidirectional. Several desirable properties of the proposed extended routing protocol are also explored. A simulation study is conducted to compare the traditional link-state protocol, the source-tree protocol, the Narvaez et al. unidirectional restoration path protocol, and the proposed bidirectional restoration path protocol.

Index Terms—Fault tolerance, link-state, Internet, loop-free, routing.

1 Introduction

Link-state routing protocols, such as OSPF [7], [8], [10], [13] and IS-IS [1], [12], are the dominant routing protocols in the Internet [2]. There are two major phases in such protocols: 1) Each IP router first collects the complete topological information of the underlying (sub)-network; 2) each router then computes the routes according to the collected topological information. The first phase is performed distributively by all the routers in the network through exchanging link state information with its neighboring routers. In the second phase, each router can construct a routing table based on the *shortest path tree* (SPT) built using the topological information. Any SPT algorithm such as Dijkstra's shortest path algorithm [3] can be used in building the SPT.

Compared to other routing protocols such as distance-vector protocols, one of the major advantages of link-state protocols is that each router computes the routes independently using the same link-state information; it does not depend on the computation done in other routers in the network. When link states are changed in the network, new information need only be sent once to each router for updating the routing table. Huitema [7] listed four good reasons why most network specialists favor link state protocols over the distance vector approach:

1. fast, loopless convergency;
2. support of precise metrics and, if needed, multiple metrics;
3. support of multiple paths to a destination; and
4. separate representation of external routes.

However, link-state protocols usually require flooding the network when any change occurs in the link states in the network. Flooding may be prohibitively expensive, especially when the link states change too frequently or when the number of links in the network is too large. Limiting the frequency of such updates can partially solve the problem when the effect of the change of the cost metric is minor in terms of transmission delay. However, this approach is inefficient in covering a link fault—because certain paths may be disconnected as a result of the link fault, delay in information update will lead to undeliverable packets.

In [11], Narvaez et al. presented a routing algorithm based on the link state method to limit routing information that needs to be delivered in a link-state protocol when a single link fails. Instead of using the flooding method, the proposed scheme restores all the paths traversing the failed link by performing only local updates on the affected routers. Specifically, a *shortest restoration path* is constructed that connects u to v for a faulty link uv (see Fig. 1). (Note that a shortest restoration path does not guarantee a shortest path from source to destination.) Their method can restore loop-free routing after a link fault while propagating information about that failure to as few routers as possible and only to the ones along the shortest restoration path. This approach is also useful to divert traffic from a congested link. However, the Narvaez et al.

- *J. Wu and F. Dai are with the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431. E-mail: {jie, fdai}@cse.fau.edu.*
- *X. Lin is with the Department of Computer Science, City University of Hong Kong, Kowloon, Hong Kong. E-mail: csxlin@ciyu.edu.hk.*
- *J. Cao is with the Department of Computing, Hong Kong Polytechnic University, Hung Hom, Kowloon, Hong Kong. E-mail: csjcao@comp.polyu.edu.hk.*
- *W. Jia is with the Department of Computer Engineering and Information Technology, City University of Hong Kong, 83 Tat Chee Ave., Kowloon, Hong Kong, SAR China. E-mail: itjia@cityu.edu.hk.*

Manuscript received 22 May 2001; revised 3 Oct. 2002; accepted 26 Dec. 2002.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number 114181.

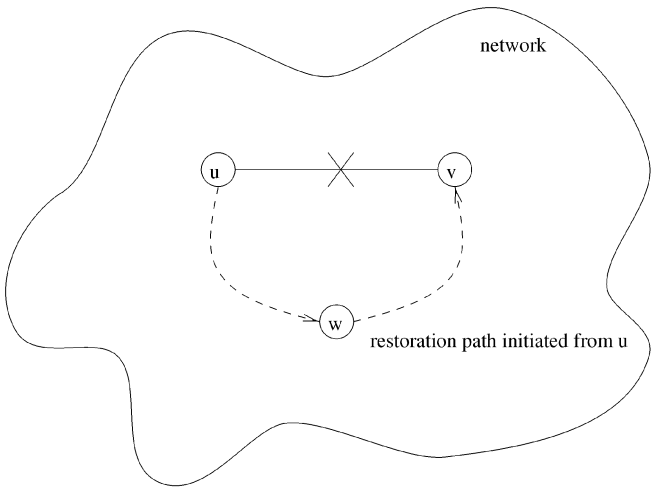

Fig. 1. A restoration path initiated from u.

approach is inefficient when a link fault is bidirectional because a restoration path is unidirectional and routing tables of nodes in the path are partially updated. In addition, two restoration paths may be generated for each bidirectional link fault.

In this paper, we modify the Narvaez et al. protocol to efficiently handle the link fault by making the shortest restoration path bidirectional so that nodes on the restoration path are completely updated and the construction of the restoration path is initiated at both end nodes of the faulty link. Only one shortest restoration path is constructed if the shortest restoration path is unique. When the shortest restoration path is not unique, one path is still constructed by either restricting the initiation process to only one end node or extending Dijkstra's algorithm. Our model is based on the bidirectional faulty model, which is commonly used in most routing protocols, including OSPF [7]. In the subsequent discussion, all link faults are assumed to be bidirectional. Note that, without specific hardware support, the detection of a unidirectional link fault is harder than a bidirectional one since each end node has to distinguish a fault that appears in the incoming link from the one in the outcoming link. We also point out a flaw in the Narvaez et al. protocol that may cause a routing loop. Several desirable properties of the proposed extended routing protocol are also explored. A performance study through simulation is conducted to compare the traditional link-state protocol, the source-tree protocol, the Narvaez et al. unidirectional restoration path protocol, and the proposed bidirectional restoration path protocol. In the subsequent discussion, we use nodes and routers interchangeably.

The rest of the paper is organized as follows: Section 2 reviews the basic ideas used in link-state routing protocols and some related works and briefly describes the Narvaez et al. fault-tolerant link-state routing protocol without flooding. Section 3 proposes an extension of the Narvaez et al. protocol for handling a bidirectional link fault. An example is given in Section 4. Section 5 discusses properties of the proposed extended routing protocol. Section 6 presents simulation results. Section 7 is the discussion and Section 8 concludes the paper and discusses possible future work.

2 Preliminaries

2.1 Related Works

Most Internet routing protocols fall in two categories: distance-vector and link-state. The distance-vector protocol is based on an iterative message exchanging process among neighbors to construct routing tables. The protocol often takes too long to converge because of the count-to-infinity problem. This problem exists even with the help of the split horizon mechanism. Distance-vector routing was used in the ARPANET until 1979, when it was replaced by link-state routing. Link-state protocols are free of routing loops, but the overhead is high because the link-state information is flooded all over the network. The details of the link-state protocol will be discussed in the next section.

A hybrid approach of distance-vector and link-state was proposed by Garcia-Luna-Aceves et al. [5], [6] to achieve both communication efficiency and loop freedom. In this approach, each node maintains a *source tree* which is an SPT, instead of global link-state in link-state protocols or routing tables in distance-vector protocols. In the source-tree approach, each node has only partial link-state information, including its adjacent links, links in its source tree, and links in its neighbors' source trees. When a link in the source tree of a node fails, the node recomputes its source tree using Dijkstra's algorithm. Note that the resultant source tree may not be optimal after the reconstruction process since it is based on partial link-state information. To ensure loop freedom and optimality, any changes in the source tree of a node are further propagated to its neighbors, which in turn recompute their source trees and feed back shorter paths (embedded in the SPTs) if any. After this process converges, each node has the optimal paths to all destinations in its source tree.

Information propagation of the source-tree protocol is similar to the one used in the distance-vector protocol and may take more steps to converge than the link-state protocol. However, the source-tree protocol avoids the count-to-infinity problem by using link-state information instead of distance information to compute shortest paths. Compared with the link-state protocol, the source-tree protocol has less storage overhead and fewer link-state update messages because each node only propagates link changes in its source tree. Two Internet protocols have been proposed using the source-tree model: the *link-vector algorithm* (LVA) [5] and the *adaptive link-state protocol* (ALP) [6]. ALP has lower message overhead than LVA when the cost of a link decreases and both ALP and LVA have similar overhead when a link fails.

2.2 Link-State Protocols

A typical link-state protocol uses the following steps:

1. Topological information of the network (link state) is first collected at each node by exchanging and accumulating adjacent link information among neighbors.
2. A shortest path tree (SPT) is constructed at each node by applying an SPT algorithm, such as Dijkstra's shortest path algorithm, on the graph representing the network topology.
3. For any routing with a given destination, a shortest path is selected from the SPT at the source node if the *source routing* approach is used; otherwise, a *routing table* is constructed from the SPT if the *distributed routing* approach is applied. The routing

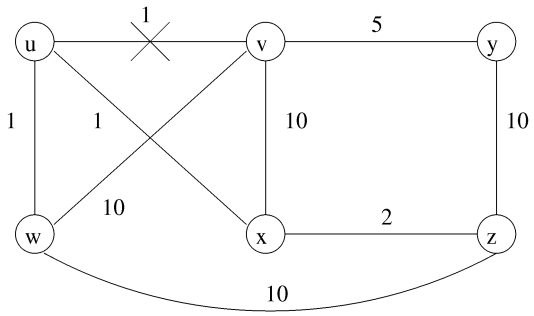

Fig. 2. A sample example.

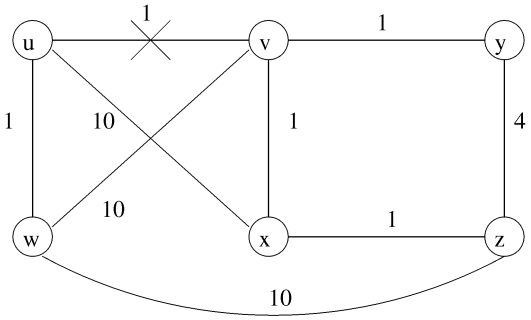

Fig. 3. Another sample example.

table includes next hop information for each destination.

Note that, in source routing, the source node decides the complete path, while, in distributed routing, only the next hop is decided at each node along the routing path. In this paper, we use the distributed routing approach where a routing table is constructed directly at each node based on the associated SPT. A (sub)network that uses link-state routing protocols can be viewed as an undirected graph $G = (V, E)$, where V is a vertex (node) set and E is an edge (link) set. $(u, v) \in E$ is a bidirectional link where u and v are two vertices in V. uv represents a directed link from u to v. Each (u, v) is associated with a cost (a positive number) representing the cost of traveling from u to v (and from v to u). Clearly, (u, v) can be viewed as two directed links, uv and vu, and nodes u and v keep the link state of uv and vu, respectively. Both end nodes u and v of a faulty link (u, v) can detect the fault. Let $u(= w_0) - w_1 - w_2 - \ldots - v(= w_n)$ denote a path which is a sequence of directed links from u to v, where vertices w_i are distinct. $u - v$ represents a directed link among two neighbors while $u -^* v$ represents a path connecting u to v through a sequence of directed links. $P(u, w_1, w_2, \ldots, v)$ represents a path consisting of undirected links.

The *tunneling* scheme [14] is a possible solution to handle link faults. Basically, a new path from u to v is constructed when link uv is broken (link vu is also broken). Any path containing uv will be replaced by a new path from u to v. Once a packet arrives at u, it will be encapsulated in another packet with destination v and forwarded along the new path until reaching v. Then, the packet is decapsulated. The remaining routing process follows a regular link-state routing protocol. However, encapsulation/decapsulation limits the efficiency of high-speed routers since every single routing packet that goes through the new path has to be encapsulated at node u. This method is not suitable to be used in high-speed networks.

Narvaez et al. [11] proposed a fault-tolerant link-state routing in the Internet without flooding. This approach can handle one unidirectional link fault at a time. The basic idea is to restore all the paths traversing the faulty link by performing updates only in the neighborhood. First, a *shortest restoration path* (a path with the minimum cost) is constructed that connects u to v (assuming uv fails, as in Fig. 1). Then, only nodes along the shortest restoration path need to update their routing tables. Specifically, *we only need to update next-hop information for those destinations that are descendants of the faulty link in the SPT of each node in the restoration path.*

The way these routing tables should be updated remains a challenge. Suppose a restoration path has been constructed, a simple update that recomputes routing tables of nodes based on new link-state information along the restoration path does not work because packets might leave the restoration path too soon. For the example illustrated in Fig. 2, suppose that only the routing tables associated with nodes u and x are updated and $u - x - v$ forms a shortest restoration path initiated from u for faulty link (u, v). A packet from u to y will exit at x to z. Because the routing table associated with z is not updated, path $z - x - u - v - y$ is still considered the shortest. Therefore, x is selected as the next hop and, consequently, a routing loop between x and z is formed. Forcing all the packets that would have had to traverse the faulty link to travel through the entire restoration path would not work either. For the example of Fig. 3, assuming that a packet needs to be forwarded from u to z, once the packet reaches v via $u - x - v$, the next hop will be x since $v - x - z$ is the shortest path to z. Again a routing loop occurs between v and x. In this situation, a packet exits the restoration path too late.

2.3 Branch Update Algorithm

The *branch update algorithm* proposed by Narvaez et al. was designed in such a way that a packet exits a restoration path at a right time. This protocol constructs a shortest restoration path $u(= w_0) - w_1 - w_2 - \ldots - v(= w_n)$ initiated from u for faulty link uv.

Branch Update Algorithm

At node $u = w_0$ upon detecting a faulty link uv or at node w_i (where $i \neq 0 \neq n$) upon receiving a special packet indicating the failure of uv.

1. The set D_i is defined as all the nodes that are descendants of uv in the shortest path tree SPT rooted at w_i (see Fig. 5).
2. The link-state database (that includes global network topology) is modified to incorporate the change of state of link uv (the link is down).
3. Dijkstra's shortest path algorithm is applied to recompute the next hop for reaching node v only. The new next hop for v is now some other node w_{i+1}.
4. The next hop for all the destination nodes in D_i is set to w_{i+1}.
5. If w_{i+1} is not equal to v, send a special packet to w_{i+1} indicating the failure of uv.

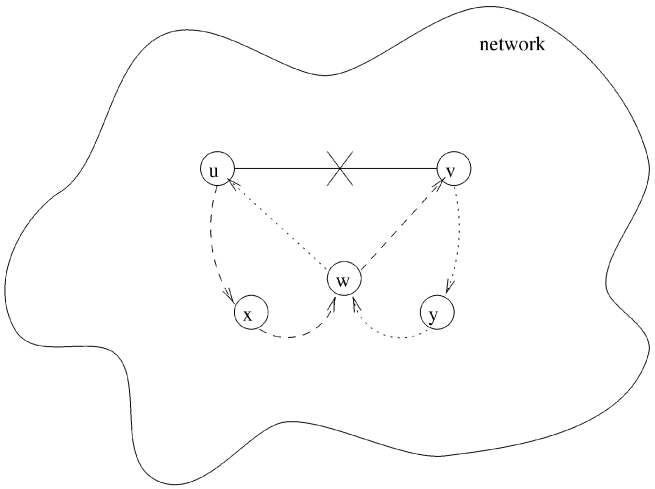

Fig. 4. Two restoration paths for faulty link (u, v).

Note that any path connecting u to v, not necessarily the shortest one, can be used as a restoration path. The Narvaez et al. protocol has two desirable properties:

- It guarantees loop-free routing after the link fault.
- If a *minimal restoration path* (in terms of hop count), rather than a shortest restoration path (in terms of cost), is used, it guarantees the minimum number of nodes that need to be informed of the link fault.

However, the branch update algorithm is proposed to handle a unidirectional link fault. Using the algorithm, two restoration paths are needed for each link fault that is bidirectional. For the example of Fig. 1, an outdated SPT rooted at w can reach a destination via either $w -^* u - v$ or $w -^* v - u$ (other destinations whose shortest paths do not include uv or vu are of no interest here). In the branch update algorithm where u is the initiator, it only updates the path of type $w -^* u - v$ to a destination and the successor of w in the restoration path is selected as the next hop to reach the destination. When the path is of type $w -^* v - u$, it is taken care of by another restoration path initiated from v. When these two paths share the same node set, each node in the set is visited twice, one for each packet initiated from each end node of the faulty link.

When two restoration paths do not share the same node set, the situation is more complex. Considering the example of Fig. 4 where link (u, v) fails, the restoration path initiated from u is $u -^* x -^* w -^* v$ and the one initiated from v is $v -^* y -^* w -^* u$. (The case for vu can be treated in a similar way.) Suppose x does not appear in $v -^* y -^* w -^* u$, then the routing table of x is *partially updated*, i.e., x knows the failure of uv but not that of vu. In fact, any shortest path of type $x -^* v - u$ to a destination is not updated at x, thus routing proceeds based on the outdated routing table without new information about the faulty link (u, v) until the packet reaches a node on the restoration path $v -^* y -^* w -^* u$ (initiated from v). Then, node u is reached by following the restoration path $v -^* y -^* w -^* u$.

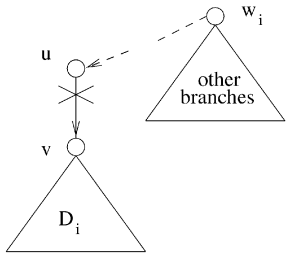

Fig. 5. An SPT rooted at w_i: Triangle D_i contains descendants of uv which belong to a branch of w_i and triangle "other branches" contains other branches of w_i.

3 Extended Branch Update Algorithm

In the extended routing protocol proposed in this paper, only one restoration path needs to be constructed for a bidirectional link fault as long as the shortest path is unique. This is done by making the path bidirectional. Initially, two restoration paths are still initiated, one from each end node of a faulty link. When two restoration paths for the same link fault meet at an intermediate node, both processes stop. In the examples of Figs. 1 and 4, when two restoration paths meet at node w, both processes stop. A bidirectional restoration path $P(u, w, v)$ is constructed for Fig. 1 and $P(u, x, w, y, v)$ for Fig. 4. To make the restoration path bidirectional, we distinguish the orientation of the path to a destination that goes through the faulty link (u, v). Suppose w is a node in the restoration path initiated from u, if a path derived from the SPT that is initiated from w to a destination contains link uv (i.e., the path is of type $w -^* u - v$), the corresponding destination is kept in set D. If the path is of type $w -^* v - u$, the corresponding destination is kept in D'. The next hop of a destination in D (D') is the successor (predecessor) of w in the restoration path. To relate two restoration paths that are intended for the same link fault, a special *marker* is used for each faulty link. A node in the restoration path is marked once visited. Note that the shortest path tree (SPT) property implies that at least one of D and D' is empty.

Extended Branch Update Algorithm

At node $u = w_0$, upon detecting a faulty link (u, v), or at node w_i (where $i \neq 0 \neq n$), upon receiving a special packet indicating the failure of $(u, v(= w_n))$:

1. The set D_i (and D'_i) is defined as all the nodes that are descendants of uv (and vu) in the shortest path tree (SPT) rooted at w_i.
2. If w_i has been marked for (u, v), exit; otherwise, w_i is marked.
3. The link-state database is modified to incorporate the change of state of link (u, v) (the link is down).
4. The next hop for all the destination nodes in D'_i is set to w_{i-1}.
5. If w_i is v, exit; otherwise, Dijkstra's shortest path algorithm is applied to recompute the next-hop for node v only. The new next hop for v is now some other node w_{i+1}.
6. The next hop for all the destination nodes in D_i is set to w_{i+1}.

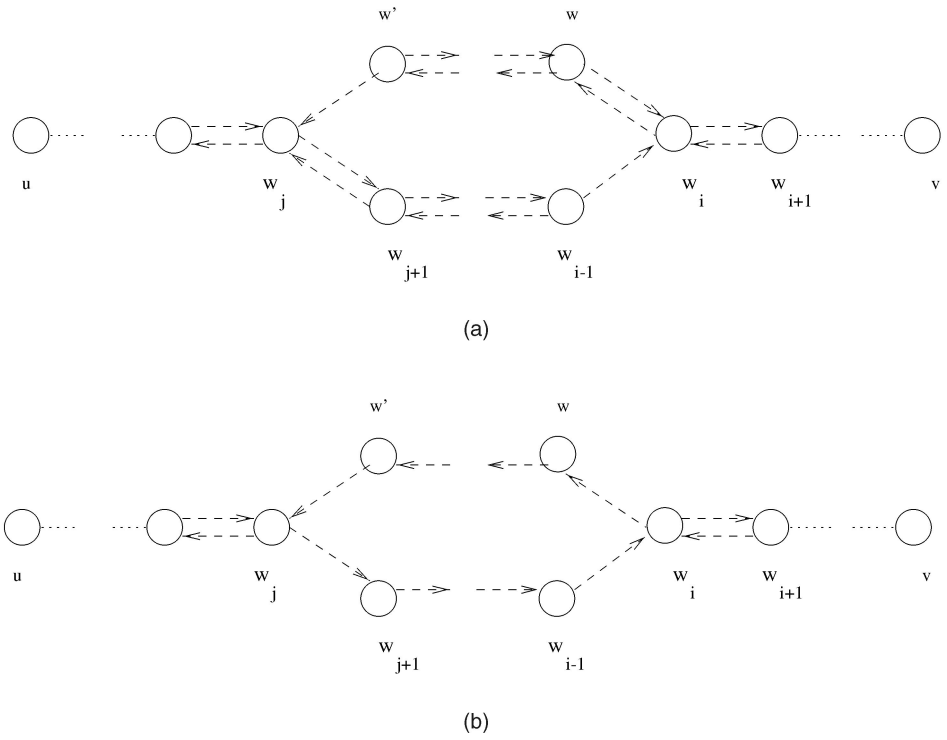

Fig. 6. Overlapped restoration paths: (a) Extended Branch Update Algorithm and (b) Branch Update Algorithm.

7. Send a special packet to w_{i+1} indicating the failure of (u, v).

The extended branch update algorithm is also applied to the other end node v of the faulty link (u, v) by exchanging the role of u and v. The extended routing protocol guarantees a shortest restoration path (see Theorem 3). In addition, it guarantees that only a minimal number of nodes need to be informed when the proposed routing protocol tries to search for a minimal restoration path (see Theorem 4).

Note that the restoration path initiated from u may or may not be the reverse of the one initiated from v because several shortest paths may exist in a given network. Two bidirectional restoration paths will be constructed for a bidirectional link fault (u, v).

The situation where a marked node is encountered deserves more discussion. Suppose the restoration path initiated from u encounters a marked node, say w_i, as shown in Fig. 6a. That means the restoration path initiated from v has selected w_i in its restoration path, that is, the restoration path initiated from v has passed through node w_i and a signal has been sent to either w_{i-1} or a node other than w_{i-1}, say w. In either case, the process initiated from u simply stops at w_i, as shown in Fig. 6a. Eventually, the path initiated from v will reach a marked node w_j (w_j could be u). Again, the process simply stops at w_j. The restoration paths constructed in the above situation are called *overlapped paths*. In the special case where $w_j = u$ and $w_i = v$, the resultant paths are called *node-disjoint paths*. Two restoration paths exist for each direction, one complete (that connects u and v) and one incomplete. In Fig. 6a, $u -^* w_j - w_{j+1} -^* w_i -^* v$ and $w' -^* w - w_i -^* v$ are for D while $v -^* w_i - w -^* w' - w_j -^* u$ and $w_{i-1} -^* w_{j+1} - w_j -^* u$ are for D'. Although two restoration paths are constructed for each bidirectional link fault, each node in a restoration path is completely updated rather than partially updated (see Fig. 6b) as in the original branch update algorithm. That is, with the same number of informed nodes, the extended routing protocol provides routing information more accurately than that of the branch update algorithm. The net effect is that the proposed routing protocol provides shorter routes for some destinations.

Considering again the example of Fig. 2, suppose that node z intends to forward a packet to y. The packet is forwarded to x because z does not have new information about the link fault (u, v) and $z - x - u - v - y$ is still considered the shortest path from z to y. If x belongs to a restoration path $v - x - u$ initiated from v, the packet is forwarded to v based on the extended routing protocol (the predecessor of x in the restoration path). Using the original branch update algorithm, we assume that two separate restoration paths are constructed: $u - w - v$ (initiated from u) and $v - x - u$ (initiated from v). Because path $x - u - v - y$ is still considered the shortest to y at x, the packet will be sent to u and then forwarded to v along the restoration path $u - w - v$. Finally, the packet is sent to y via v. The resultant routing path is $z - x - u - w - v - y$. Note that the path $v - x - u$ is just the optimal replacement for faulty link (u, v), thus the resultant routing path $z - x - v - y$ generated from the extended routing protocol is not the shortest. In fact, the shortest path from z to y is path $z - y$ with a cost of 10. However, if the shortest path does not contain the faulty link, it will remain the shortest.

If the requirement is to generate a single restoration path for each link fault, the extended routing protocol can be easily modified to ensure that one and only one end node of each link fault initiates the construction of a restoration path. This can be accomplished by comparing the IDs of two end nodes and letting the one with the larger ID initiate the process. Still another approach exists which initiates the

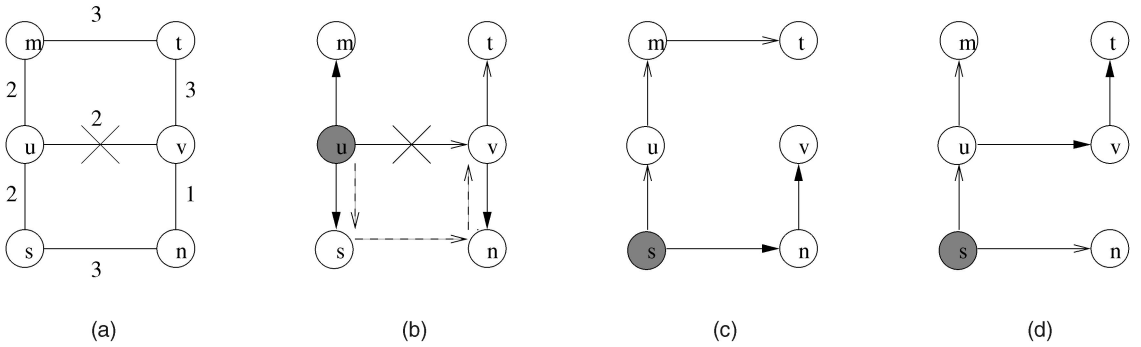

Fig. 7. In network (a), multiple shortest paths exist from u and s to t. In node u's view (b), its shortest path to t is affected by the faulty link and should be adjusted; while, from s's view (c), its shortest path is not affected. (d) The SPT rooted at s generated from the extended Dijsktra algorithm.

construction from both end nodes, but guarantees a single restoration path. It is based on a simple modification of Dijkstra's algorithm which allows us to detect all the "equal length" paths [7]. Here, we further modify the *extended Dijkstra algorithm* by *keeping the ID of the last hop of each path* during the formation of SPT. During the formation of SPT, when two equal length paths are detected, the one with a larger ID of the last hop survives. In addition, both u and v try to find a restoration path from u to v and the restoration path from v is just the reverse of the one from u. This approach is called asymmetric simple restoration path construction.

It should be stressed that the above extended Dijkstra algorithm should be used at each node to construct its SPT. Otherwise, a routing loop may occur during a restoration process (a flaw in the original Narvaez et al. protocol). For example, in Fig. 7a, there are two shortest paths between nodes u and t. However, only one path is used in node u's shortest path tree (SPT). If there is no consistent rule for each node to select a shortest path, node u may select the path $u - v - t$. Similarly, there are three shortest paths between nodes s and t and node s may select the path $s - u - m - t$. After link (u, v) fails, a restoration path $u - s - n - v$ is constructed at u to bypass the faulty link. Node u changes its next hop for v to the next node in the restoration path, which is node s. On the other hand, since the SPT rooted at node s does not use link (u, v) in its shortest path to t, its next hop for t remains the same, which is u. When a routing packet sent to t reaches node u, it will be circulated between nodes u and s and can never reach its destination. Using the extended Dijkstra algorithm, the following property is ensured: *If $u -^* v -^* t$ is a path in the SPT rooted at node u, then the same subpath $v -^* t$ appears in the SPT rooted at node v.* In this case, node s should have selected path $s - u - v - t$, as shown in Fig. 7d (since u has a larger id than n and v has a larger id than m).

Like any link-state-based protocols (including the Narvaez et al. protocol), the extended routing protocol may generate short-term loops because of the delay in link-state propagation along the restoration path. Referring to Fig. 1, consider a shortest path from s to t (not shown in the figure) that goes through link (u, v). Suppose the routing packet reaches node w before the restoration process starts. If the restoration path that includes node w is constructed when the packet reaches node u, clearly node w will be visited again since it is along the restoration path for (u, v). However, short-term is temporary and will not cause serious problems.

4 Example

In this section, we illustrate the proposed scheme using an example. Fig. 8 shows a sample network with eight nodes. Table 1 shows routing tables for all nodes in Fig. 8, where "Dest." stands for destination and "NH(*id*)" represents the next hop information in the routing table associated with node *id*. Distance information is not included in Table 1.

Suppose link (u, v) fails, we can easily determine the corresponding shortest restoration path as $u - x - y - v$, that is, $w_0 = u$, $w_1 = x$, $w_2 = y$, and $w_3 = v$. Fig. 9 shows the SPT (before link (u, v) fails) rooted at each w_i for $i =$ 0, 1, 2, and 3. Note that, in Fig. 9a, there are two shortest paths from u to y: $u - x - y$ and $u - v - y$. Path $u - x - y$ is selected because x has a larger id than v. D_i and D'_i can be easily derived from the corresponding SPT:

- At node u: $D_0 = \{v, w\}$ and $D'_0 = \phi$.
- At node x: $D_1 = \{v, w\}$ and $D'_1 = \phi$.
- At node y: $D_2 = \phi$ and $D'_2 = \phi$.
- At node v: $D_3 = \phi$ and $D'_3 = \{t, u, x, z\}$.

There are two views of the network: Nodes s, t, w, and z have the old view (link states before the link fault), while nodes u, v, x, and y have the new view (link states after the link fault). Based on the proposed scheme, only the routing tables of nodes u, v, x, and y that are along the shortest restoration path are affected by the link fault (as shown in Table 2). In Table 2, $id \rightarrow id'$ in the next hop column represents the following: id is the next hop before link (u, v) fails and id' is the

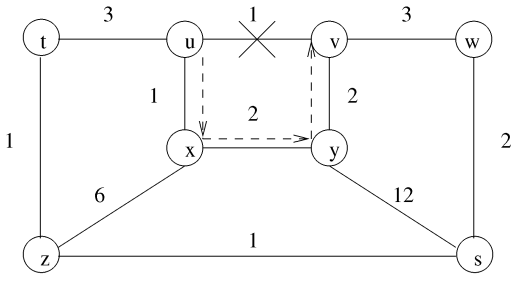

Fig. 8. A sample network with eight nodes. The dotted line corresponds to the restoration path for faulty link (u, v).

TABLE 1
Routing Tables of Fig. 8 before Link (u, v) Fails

Dest.	NH(s)	NH(t)	NH(u)	NH(v)	NH(w)	NH(x)	NH(y)	NH(z)
s	–	z	t	w	s	u	v	s
t	z	–	t	u	s	u	x	t
u	z	u	–	u	v	u	x	t
v	w	u	v	–	v	u	v	t
w	w	z	v	w	–	u	v	s
x	z	u	x	u	v	–	x	t
y	w	u	x	y	v	y	–	t
z	z	z	t	u	s	u	x	–

next hop after link (u, v) fails. All other entries in the next hop columns in Table 2 remain unchanged, that is, the same as ones in Table 1. Again, after the faulty link is replaced by a restoration path, a shortest path that includes the faulty link may or may not be the shortest path after the replacement. In the example of Fig. 8, any shortest path using the restoration path is still the shortest. In the example of Fig. 7, the shortest path $u - v - t$ is replaced by $u - s - n - v - t$ after the failure of link (u, v) and it is no longer the shortest one from u to t. The shortest one is $u - m - t$.

5 Properties

In this section, we study several desirable properties of the extended branch update algorithm. Let G and G' be graphs (representing network topology) before and after link fault (u, v). $d_G(u, v)$ and $d_{G'}(u, v)$ are the distances between u and v in G and G', respectively. Since u and v are directly connected in G, $d_G(u, v)$ is the cost of link (u, v) in G. Unless otherwise specified, the restoration path here refers to either a single restoration path or overlapped restoration paths (including node-disjoint restoration paths). Also, it is assumed that a restoration path is constructed when a routing process starts.

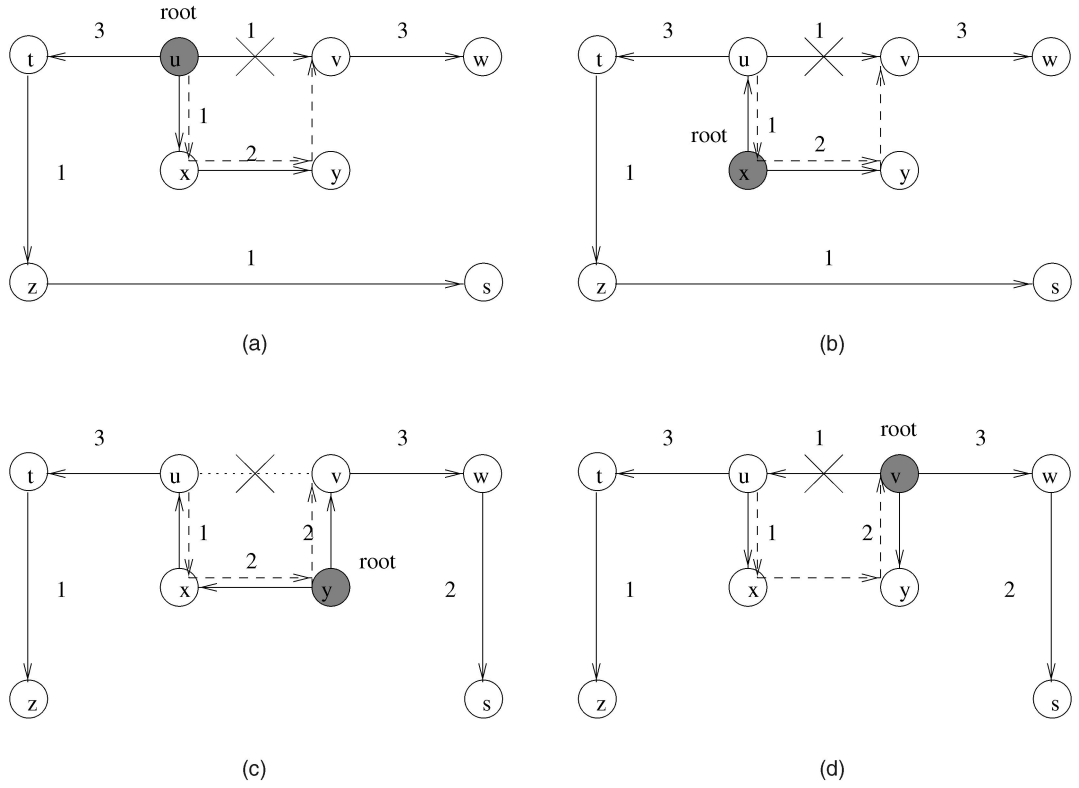

Fig. 9. SPT rooted at (a) node u, (b) node x, (c) node y, and (d) node v.

TABLE 2
Routing Tables of Nodes along the Restoration Path after Link (u, v) Fails

Dest.	NH(u)	NH(x)	NH(y)	NH(v)
s	t	u	v	w
t	t	u	x	$u \rightarrow y$
u	$-$	u	x	$u \rightarrow y$
v	$v \rightarrow x$	$u \rightarrow y$	v	$-$
w	$v \rightarrow x$	$u \rightarrow y$	v	w
x	x	$-$	x	$u \rightarrow y$
y	x	y	$-$	y
z	t	u	x	$u \rightarrow y$

The following result shows that shortest paths in G remain the shortest in G' if they are not affected by the faulty link; otherwise, the lengths of these paths increase by a predictable value.

Theorem 1. *The extended branch update algorithm ensures routing optimality as long as the path constructed before the link fault does not contain the faulty link; otherwise, the increase in the length of the path is upper bounded by* $d_{G'}(u, v)$ - $d_G(u, v)$.

Proof. It is clear that a link fault (u, v) will not decrease the distance between two nodes. Therefore, a shortest path in G remains the shortest in G' if the path does not contain the faulty link. On the other hand, if a shortest path in G contains (u, v), the faulty link will be replaced by a shortest restoration path between u and v in G'. Consider a destination which is a descendant of uv in the SPT. Suppose the packet to be routed reaches node u without reaching any node along the restoration path, then link uv is replaced by the restoration path and the length of the path increases by $d_{G'}(u, v)$ - $d_G(u, v)$. Note that the packet may exit the restoration path because a shorter path is found to the destination. In this case, the increase in the length of the path will be less than $d_{G'}(u, v)$ - $d_G(u, v)$. Suppose the packet reaches node w, which is along the restoration path, before it reaches node u. In this case, the restoration path can be simply expressed as $u -^* w -^* v$. The packet directly follows the restoration path at node w to reach node v. Following a similar argument as in the first case, the increase in the length of the path will be upper bounded by $d_{G'}(u, v)$ - $d_G(u, v)$ - $d_{G'}(u, w) < d_{G'}(u, v)$ - $d_G(u, v)$. □

Because information about the faulty link is distributed to nodes along the restoration path, link state information associated with different nodes is different, i.e., link state information is either *updated* including the location of the faulty link or *outdated* without including the location of the faulty link. The following result shows that the routing process is still loop-free even with inconsistent views of link states among nodes in a network. Again, we assume that routing tables are stable during the routing process, that is, each table of a node along the restoration path has a new view (knowing the link fault) and each table of a node outside has the old view.

Theorem 2. *The extended routing protocol ensures that loop-free routing will continue after the link fault.*[1]

Proof. We first review a concept used in [11]. A packet at node s is *affected* by a faulty link if its intended destination d is a descendent of the faulty link in the SPT rooted at s; otherwise, it is *unaffected*. If there is only one restoration path for faulty link (u, v), without loss of generality, we assume that a packet is affected because of uv (not vu), as shown in Fig. 5. The following three cases are considered (see Fig. 10):

1. If a packet at node s is unaffected, it will remain unaffected and reach the destination d based on the outdated SPT that is loop-free. Suppose node u on the path $s -^* d$ is affected, then the path $u -^* d$ in the SPT rooted at u is different from the path $u -^* d$ in the SPT rooted at s. This is a contradiction to the property of the extended Dijkstra algorithm.
2. If a packet at node s is affected and s is on the restoration path, then, as long as the packet is affected, the packet will stay on the restoration path until it becomes unaffected at w_j (w_j could be node v). Path $s -^* w_j$ is loop-free. Since the packet is unaffected at w_j, it remains unaffected and eventually reaches its intended destination d. Again, $w_j -^* d$ is loop-free. In addition, paths $s -^* w_j$ and $w_j -^* d$ do not share any intermediate node. Therefore, path $s -^* w_j -^* d$ is loop-free.
3. If a packet at node s is affected but s is not on the restoration path, then the packet is routed based on the outdated SPT until reaching node w_i (including node u) which is on the restoration path for (u, v). Path $s -^* w_i$ is clearly loop-free. As long as the packet is affected, the packet will stay along the restoration path until it becomes unaffected at w_j ($i < j$ and w_j could be node v). Path $w_i -^* w_j$ is loop-free. Since the packet is unaffected at w_j, it remains unaffected and eventually reaches its intended destination d. Again, path $w_j -^* d$ is loop-free. It is obvious that $w_i -^* w_j$ does not share any intermediate node with either $s -^* w_i$ or $w_j -^* d$ since intermediate nodes in $w_i -^* w_j$ belong to the restoration path. We use proof by contradiction to show that $s -^* w_i$ and $w_j -^* d$ do not share any intermediate node. Assume that these two paths share node w which has an outdated SPT (see Fig. 10). Node w in $s -^* w_i$ is affected by the faulty link while node w in $w_j -^* d$ is unaffected. This is a contradiction. Therefore, $s -^* w_i -^* w_j -^* d$ is loop-free.

Two overlapped or node-disjoint restoration paths may be constructed for a link fault (u, v). This occurs in the extended branch update algorithm when bidirectional restoration paths initiated from u and v do not select the same set of intermediate nodes. After overlapped or node-disjoint restoration paths are constructed, there are two restoration paths for each direction. Since each packet will use at most one

1. A similar result is given in [11]; however, the proof in [11] is flawed.

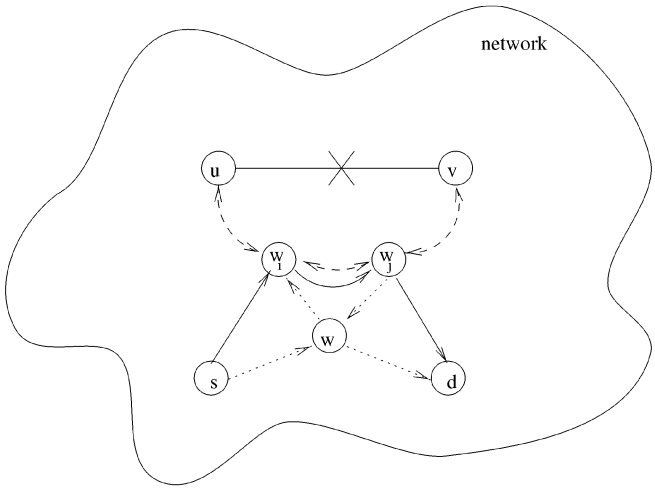

Fig. 10. Loop-free routing.

restoration path, the same argument used for the single restoration path case still applies. □

The next two theorems show properties related to restoration path(s) constructed from the extended routing protocol.

Theorem 3. *The extended routing protocol ensures shortest restoration path(s) from u to v (from v to u).*

Proof. The result applies to complete restoration paths (which connect u and v). Based on the extended branch update algorithm, each node w_i has updated link state information when it selects node w_{i+1} on the restoration path. When two restoration paths are constructed, they are done independently from two end nodes of the faulty link. Therefore, each path is a shortest one. If one restoration path from u to v is constructed from combining two (sub)paths, one from each end node of the faulty link (u, v), without loss of generality, we assume that these two paths are merged at node w. Clearly $u -^* w$ and $w -^* v$ are the shortest paths between u and w and between w and v, respectively. $u -^* w -^* v$ is the bidirectional path formed by combining $u -^* w$ and $w -^* v$ and it is a shortest restoration path between u and v with w being an intermediate node. In addition, there will be no shorter restoration path between u and v without using w as an intermediate node; otherwise, w will not be selected during the construction of $u -^* w$. The case for the restoration path from v to u can be proven in a similar way. □

Theorem 4. *If a minimal restoration path is used and a single restoration path is constructed, the extended routing protocol guarantees such a minimal path and the number of nodes along the path corresponds to the minimum number of nodes that need to be informed of the link fault.*

Proof. In [11], it has been proven that the number of nodes in a minimal restoration path corresponds to the minimum number of nodes that need to be informed of the link fault without causing the looping problem, i.e., if the number of nodes to be informed is less than the minimum number, there is always a set of metrics for the links of the network that will cause any scheme to create routing loops after the link failure. Let $P(u, w)$ and $P(w, v)$ be the sections of bidirectional paths between u and w and between w and v, respectively. We only need to prove that, when the resultant restoration path is constructed by combining $P(u, w)$ and $P(w, v)$ at node w, path $P(u, w, v)$ has the minimum number of nodes. Clearly, both $P(u, w)$ and $P(w, v)$ contain minimum numbers of nodes between u and w and between w and v, respectively. $P(u, w, v)$ is the path formed by combining $P(u, w)$ and $P(w, v)$ and it is a minimum restoration path between u and v with w being an intermediate node. In addition, there will be no restoration path between u and v without using w as an intermediate node that has a fewer number of nodes; otherwise, w will not be selected during the construction of $P(u, w)$. □

6 Performance Study

We conducted a performance study through simulation to compare the overhead and performances of four routing protocols: the traditional link-state protocol (LS), the source-tree-based protocol (ST), the traditional link-state protocol using unidirectional restoration paths (URP), and bidirectional restoration paths (BRP). These protocols are simulated in a custom discrete event simulator based on the following time-slot model: The simulation time is divided into unit-length slots (also called steps). During each step, each node receives control messages (i.e., link-state information) that are sent by its neighbors in the previous step, updates its routing table, and sends messages if necessary. A simulation starts when a bidirectional link fault is detected and stops after the simulated protocol converges, that is, when the fault has been dealt with and no more control messages need to be sent.

Networks used in the simulation are generated by BRITE [9], a general-purpose random topology generator. BRITE is specially designed to generate Internet-like network topologies that exhibit the *power-law* [4] for the distribution of node degrees. That is, the number of nodes with a given degree d is proportional to d^α, where α is a constant. Such a network can be generated in two steps. First, n nodes are randomly placed in a rectangular area (this process is called *node placement*). Then, bidirectional links are added according to the *RouterWaxman* model, that is, nodes are incrementally added to the network. For each newly added node, m existing nodes are selected as its connecting nodes, where the geographically closer nodes have the higher probabilities to be selected. Each link is assigned a cost, which is an integer between 1 and 10 and is proportional to the geographical distance between the two end nodes.

All four routing protocols are evaluated upon two categories of random networks: sparse and dense ones. In relatively sparse networks ($m = 2$), nodes are randomly placed in the rectangular area. A faulty link usually affects many nodes and needs a relatively long restoration path in such networks. In relatively dense networks ($m = 8$), the node placement follows the *heavy-tailed* distribution to generate Internet-like network topologies. A variable X follows a heavy-tailed distribution if $P[X > x] = k^\alpha x^\alpha L(x)$, where k and α are constants and $L(x)$ is a slowly varying

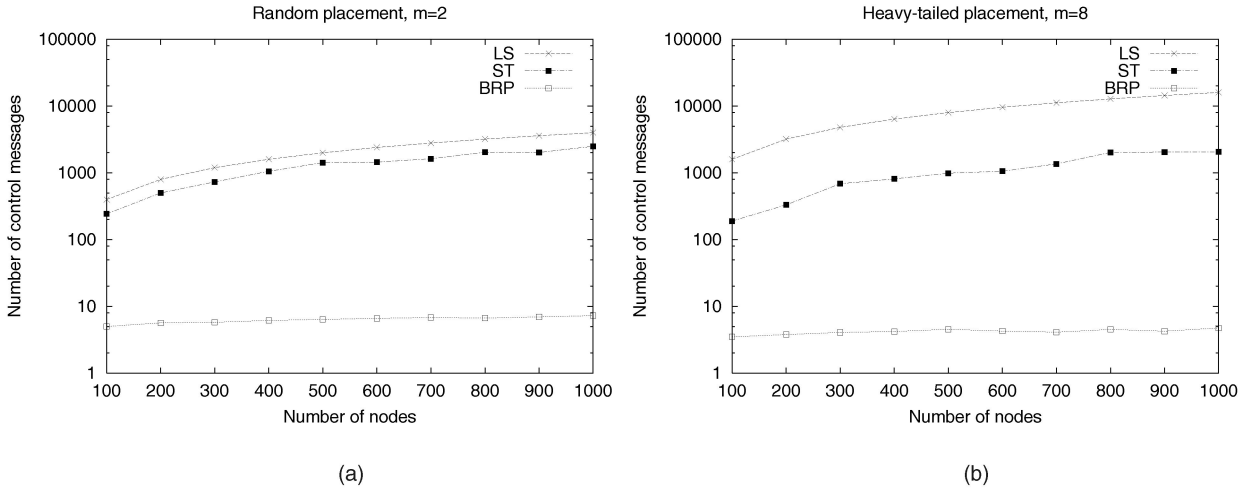

Fig. 11. Communication overhead of different routing protocols in (a) relatively sparse networks and (b) relatively dense networks.

function. In such networks, a faulty link can usually be replaced by a relatively short restoration path in the neighborhood and affect much fewer nodes than in sparse networks. For each category, we generate random networks with sizes (n) ranging from 100 to 1,000. For each n, we generate 200 networks and compute the following three metrics of each routing protocol:

1. *Message overhead*: The average number of control messages per faulty link. Each control message carries updated link-state information.
2. *Converging speed*: The average number of steps it takes to deal with a faulty link.
3. *Routing performance*: The average length increase of a routing path compared with the optimal path after a faulty link.

In LS and ST, a control message is sent to all neighbors. Each message is counted as multiple messages in a point-to-point network (such as switch-(router-)based networks) and as a single message in a shared-medium network (such as Ethernet). In URP and BRP, a control message is sent only to its next node in the restoration path and is counted as a single message. The simulation results are presented in two groups:

1. BRP versus the two nonrestoration path protocols (i.e., SL and ST).
2. BRP versus URP.

For the first group, only message overhead (as shown in Fig. 11) and converging speed (as shown in Fig. 12) are compared. Both nonrestoration path protocols generate optimal paths. The average path length increase of BRP is presented in the second group. Fig. 11 shows the magnitude of difference in message overhead between the restoration path and the two nonrestoration path protocols. Both SL and ST generate thousands of control messages, while the average number of control messages generated by BRP is less than 10. In BRP, all control messages are sent along restoration paths and, along the restoration path, each node forwards a control message at most once. Therefore, the number of control messages is no more than the number of nodes in restoration paths. In SL, the control message containing the updated link-state information is flooded throughout the network. The number of control messages is equal to the total degree of nodes in the network (i.e., twice the number of links in the network) in a point-to-point

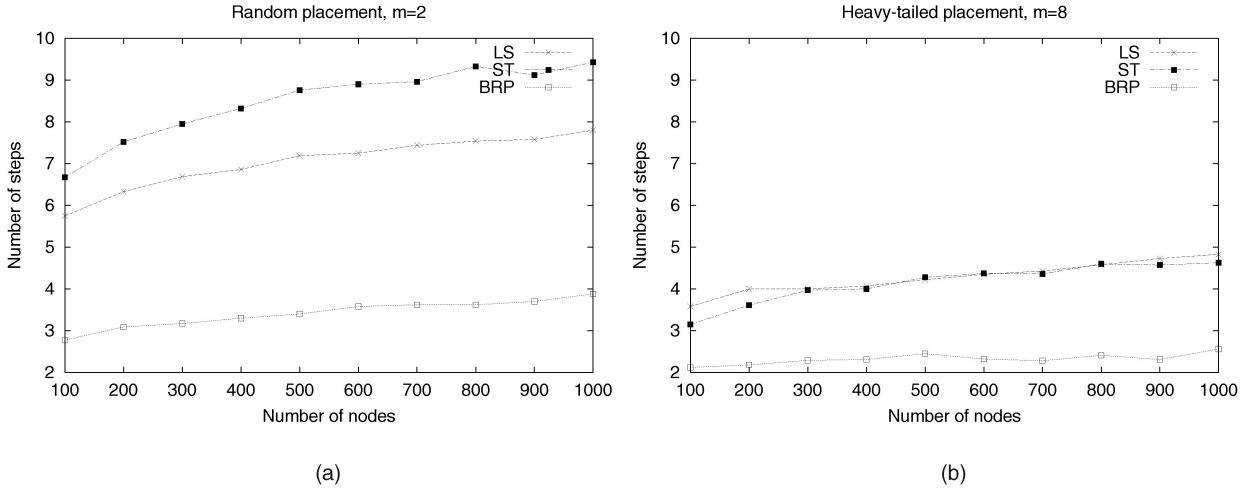

Fig. 12. Converging speed of different routing protocols in (a) relatively sparse networks and (b) relatively dense networks.

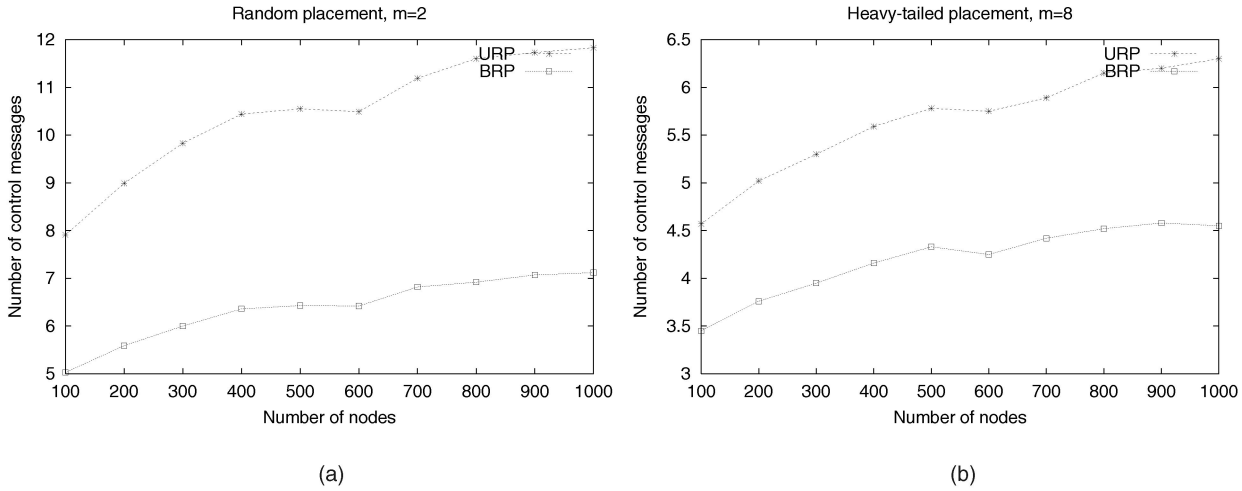

Fig. 13. Communication overhead of two restoration path protocols in (a) relatively sparse networks and (b) relatively dense networks.

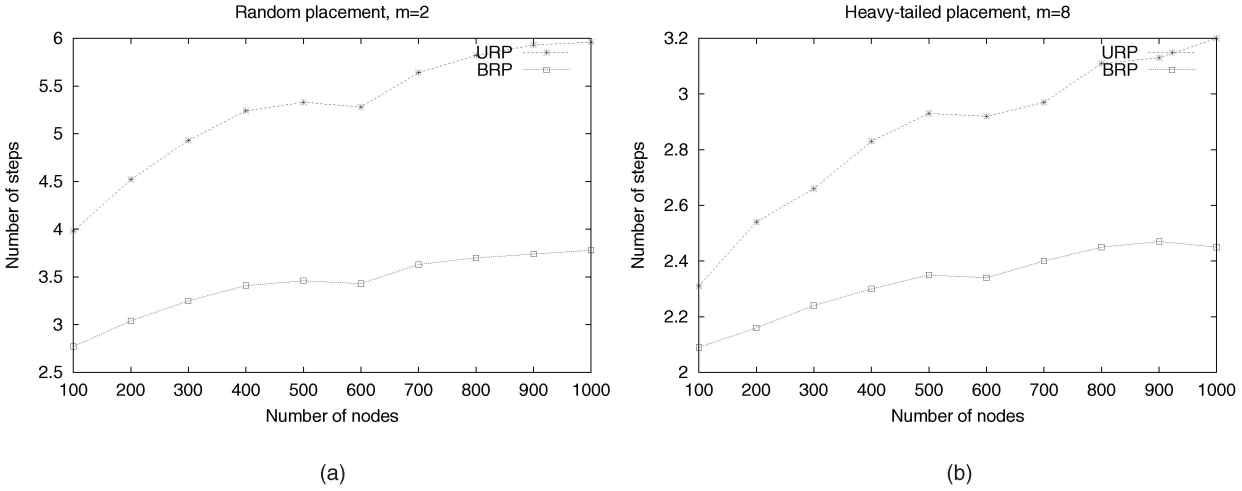

Fig. 14. Converging speed of two restoration path protocols in (a) relatively sparse networks and (b) relatively dense networks.

network and is equal to the number of nodes in the network in a shared-medium network (not shown in Fig. 11). In ST, only affected nodes (i.e., those nodes with their source trees adjusted) send control messages containing the added or deleted links in their source trees. The number of control messages is usually more than the total degree of the affected nodes because some affected nodes may send control messages more than once. Fig. 12 shows that the restoration path protocol also has faster converging speed than nonrestoration path protocols. In relatively sparse networks, ST takes more steps than SL because control messages may propagate back and forth to rebuild an optimal path in ST. In relatively dense networks, ST and SL need a similar number of steps. In these networks, ST is likely to rebuild optimal paths in the neighborhood of the faulty link and converges more quickly than SL. In both cases, the number of steps used by BRP is at most half of those used by SL or ST.

For the second group, all three metrics are compared and BRP outperforms URP under all these metrics. Note that, when two restoration paths, unidirectional or bidirectional, are constructed simultaneously from both ends of the faulty link, their relationship can be one of the following three cases: 1) totally distinct (i.e., node-disjoint), 2) totally overlapped (i.e., single), or 3) partially overlapped. In case 1), URP requires the same number of steps and control messages as BRP, but may have a higher average length increase of affected paths. In case 2), URP has the same average length increase as BRP, but requires more steps and control messages to construct the restoration path. In case 3), URP requires more steps and control messages than BRP, as in case 2), and has a higher average length increase of affected paths, as in case 1). Simulation results show that, on average, the number of control messages generated by BRP is about 60 percent of that generated by URP (as shown in Fig. 13) and the number of steps used by BRP is also about 60 percent of that used by URP (as shown in Fig. 14) in both relatively dense and relatively sparse networks.

The average length increase is computed as:

$$\frac{\sum_{u,v \in V}(d(u,v) - d_{opt}(u,v))}{NP_{affected}},$$

where $d(u,v)$ is the traveling distance from node u to node v in a restoration path protocol, $d_{opt}(u,v)$ is the length of the shortest path between node u and node v, and $NP_{affected}$ is the number of paths affected by the faulty link. A shortest path between a pair of source and destination nodes is said to be affected by a link fault if it contains the faulty link prior to the link fault. Note that the average length increase is a measure for affected paths only. The average length

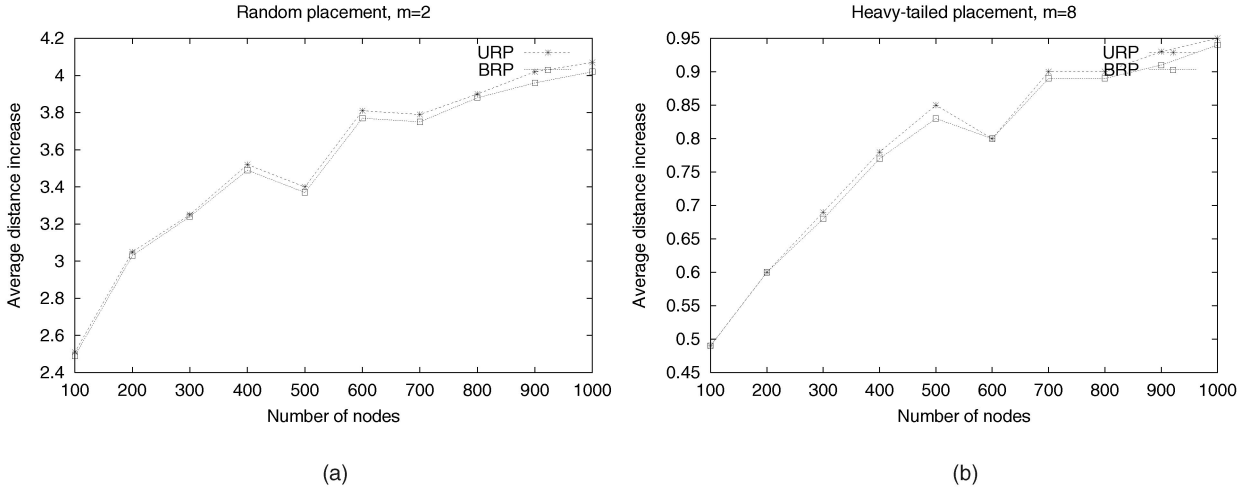

Fig. 15. Average path length increase of two restoration path protocols in (a) relatively sparse networks and (b) relatively dense networks.

increase in percentage is defined as the ratio of the average length increase over all nodes (affected and unaffected) to the average length of all paths (affected and unaffected). Note that the quality of routes deteriorates after a sequence of faults. Average length increase may accumulate for each route. Therefore, after a predefined period, the network needs to be "refreshed" by collecting global link-state (see details in the next section). Here, we simulate only the first link failure in each network to compute the length increase. As expected, the average length increase of affected paths is larger in relatively sparse networks than in relatively dense networks (as shown in Fig. 15). The average length increase of BRP is smaller than URP, but the difference is insignificant. The percentage of affected paths is also higher in relatively sparse networks (as shown in Fig. 16). In fact, the percentage of affected paths is very low for both relatively dense ($< 0.3\%$) and relatively sparse ($< 1.6\%$) networks. That explains why the average length increase in percentage is extremely small for very large networks. When the network size is 1,000, the average length increase in relatively sparse networks is about 0.04 percent, and, in relative dense networks, the average length increase is about 0.005 percent (as shown in Fig. 17).

Overall, compared with SL and ST, BRP has the least message overhead and fastest converging speed. There is a performance penalty in the average traveling distance of a routing packet. However, the average length increase is relatively small per link fault. Compared with URP, BRP has significant improvements in message overhead and converging speed and a slight improvement in routing performance.

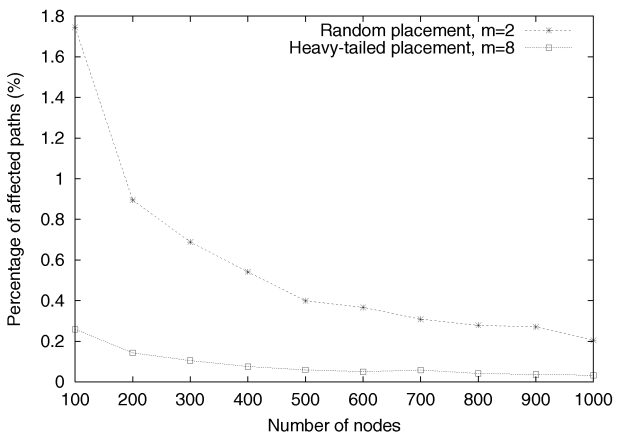

Fig. 16. Average percentage of paths affected by a link fault.

7 Discussion

In Fig. 8, suppose there are two faults $f_1 : (u, v)$ and $f_2 : (t, z)$. The shortest restoration path for f_1 is $u - x - y - v$, that is, $w_0 = u$, $w_1 = x$, $w_2 = y$, and $w_3 = v$, as discussed in Section 4. The shortest restoration path for f_2 is $z - x - u - t$, that is, $w_0 = z$, $w_1 = x$, $w_2 = u$, and $w_3 = t$. Note that nodes x and u appear in the restoration paths of both faults. D_i and D'_i of f_1 and f_2 can be derived from the corresponding SPT (see Figs. 9 and 18):

- At node u: $D_0(f_1) = \{v, w\}$ and $D'_0(f_1) = \phi$.
- At node x: $D_1(f_1) = \{v, w\}$ and $D'_1(f_1) = \phi$.
- At node y: $D_2(f_1) = \phi$ and $D'_2(f_1) = \phi$.
- At node v: $D_3(f_1) = \phi$ and $D'_3(f_1) = \{t, u, x, z\}$.

and

- At node z: $D_0(f_2) = \{t, u, v, x, y\}$ and $D'_0(f_2) = \phi$.
- At node x: $D_1(f_2) = \phi$ and $D'_1(f_2) = \{s, z\}$.
- At node u: $D_2(f_2) = \phi$ and $D'_2(f_2) = \{s, z\}$.
- At node t: $D_3(f_2) = \phi$ and $D'_3(f_2) = \{s, w, z\}$.

There exist four views: Nodes u and x know the failures of f_1 and f_2, nodes t and z know only f_2, nodes y and v know only f_1, and the rest do not know the existence of either f_1 or f_2. The routing table after the occurrences of f_1 and f_2 is shown in Table 3. Consider a routing example from z to v, the message is first routed along the restoration path of f_2 and exits at node x, which is on the restoration path of f_1. The resultant routing path is $z - x - y - v$. Note that the shortest path before the occurrences of f_1 and f_2 is $z - t - u - v$.

However, as in the Narvaez et al. protocol, a routing loop may occur in some extreme cases, even when restoration paths are disjointed, as shown in Fig. 19. In this example, there are two faults (s, t) (with restoration path $s - z - y - t$

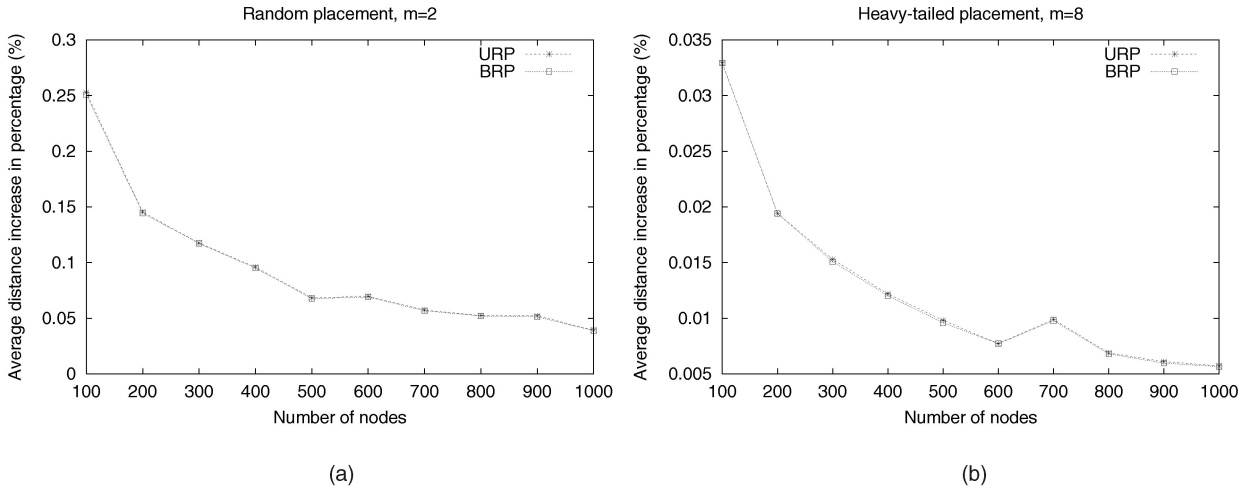

Fig. 17. Average length increase in percentage of two restoration path protocols in (a) relatively sparse networks and (b) relatively dense networks.

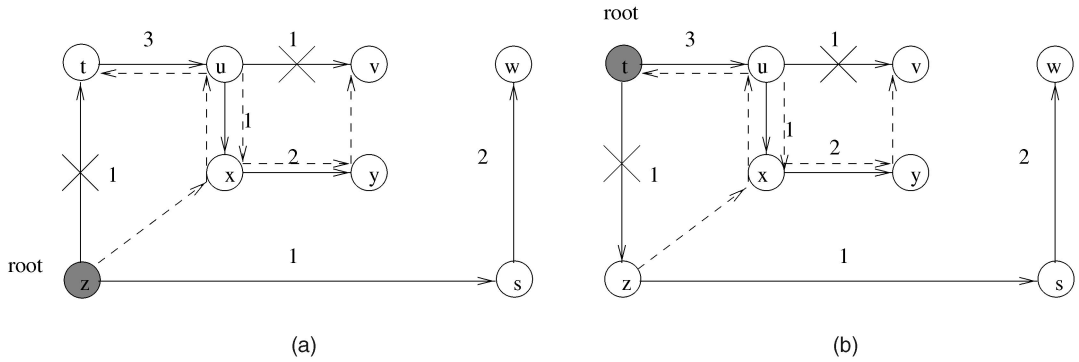

Fig. 18. SPT rooted at (a) node z and (b) node t.

initiated from s) and (u, v) (with restoration path $v - w - x - u$ initiated from v). Consider a routing from s to d, the message is routed around fault (s, t) through its restoration path and exits the path at node y to enter the restoration path of (u, v) at w. The message is then routed around the restoration path of (u, v) and exits the path at node x and enters the restoration path of (s, t) again at node z. In this case, a routing loop is formed among nodes z, y, w, and x.

TABLE 3
Routing Tables of Nodes along the Restoration Path after Links (u, v) and (t, z) Fail

Dest.	NH(t)	NH(u)	NH(x)	NH(y)	NH(v)	NH(z)
s	$z \rightarrow u$	$t \rightarrow x$	$u \rightarrow z$	v	w	s
t	$-$	t	u	x	$u \rightarrow y$	$t \rightarrow x$
u	u	$-$	u	x	$u \rightarrow y$	$t \rightarrow x$
v	u	$v \rightarrow x$	$u \rightarrow y$	v	$-$	$t \rightarrow x$
w	$z \rightarrow u$	$v \rightarrow x$	$u \rightarrow y$	v	w	s
x	u	x	$-$	x	$u \rightarrow y$	$t \rightarrow x$
y	u	x	y	$-$	y	$t \rightarrow x$
z	$z \rightarrow u$	$t \rightarrow x$	$u \rightarrow z$	x	$u \rightarrow y$	$-$

One way to handle such a routing loop is to keep a *routing history* of restoration paths visited to detect a routing loop. Suppose each restoration path has a distinct label (say, the corresponding faulty link), whenever the routing message uses a new restoration path (i.e., using at least one link of the restoration path), it is recorded in its routing history. A repeat appearance of a restoration path corre-

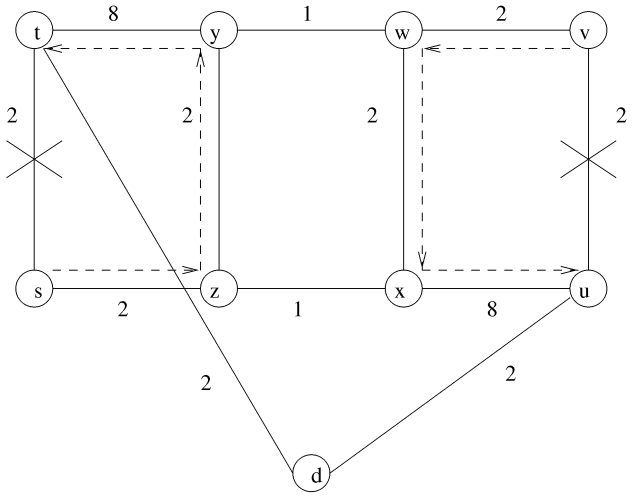

Fig. 19. A routing loop between two disjointed restoration paths for routing from s to d.

sponds to an unsafe state (that might lead to a deadlock situation). In this case, a global flooding is initiated. Note that the length of routing history is bounded by the number of faults in the network. Therefore, routing history will not incur significant overhead. Also, the length of routing history can be used as a lower bound for the number of faults in the network. Another option is to predefine a threshold. Once the length of the routing history exceeds the threshold, a global flooding starts.

8 Conclusions

In this paper, we have extended a fault-tolerant link-state routing protocol in the Internet. This approach trades optimality for low overhead. A shortest path is maintained as long as it does not contain a faulty link; otherwise, the faulty link is replaced by a shortest restoration path and the cost increase of the resultant path is upper bounded by the cost difference between the shortest restoration path and the faulty link. Our approach is based on the premise that link faults are bidirectional. Therefore, instead of constructing two unidirectional restoration paths (one from each end node of a faulty link), one bidirectional restoration path is constructed. The simulation results show that the proposed approach has much less message overhead and faster converging speed compared with the existing ones, including the Narvaez et al. unidirectional restoration path protocol, the Garcia-Luna-Aceves and Behrens source-tree protocol, and the traditional link-state protocol.

Our future work includes investigating other possible trade offs between optimality and low overhead. We also plan to develop efficient and fast algorithms (especially symmetric algorithms) for constructing a single restoration path from both end nodes of a faulty link. The efficient way of handling multiple faults, especially the prevention method that avoids routing loop among restoration paths, still remains an open problem.

Acknowledgments

This work was supported in part by US National Science Foundation grants CCR 9900646, ANI 0073736, and EIA 0130806.

References

[1] R. Callon, "Use of OSI IS-IS for Routing in TCP/IP and Dual Environments," RFC-1195, 19 Dec. 1990.
[2] D.E. Comer, *Internetworking with TCP/IP,* vol. 1, second ed. Prentice Hall, 1991.
[3] E.W. Dijkstra, "A Note on Two Problems in Conexion with Graphs," *Numerische Math.,* vol. 1, pp. 269-271, 1959.
[4] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *Proc. SIGCOMM,* pp. 251-262, 1999.
[5] J.J. Garcia-Luna-Aceves and J. Behrens, "Distributed, Scalable Routing Based on Vectors of Link States," *IEEE J. Selected Areas in Comm.,* vol. 13, no. 8, pp. 1383-1395, Oct. 1995.
[6] J.J. Garcia-Luna-Aceves and M. Spohn, "Scalable Link-State Internet Routing," *Proc. IEEE Sixth Int'l Conf. Network Protocols (ICNP '98),* Oct. 1998.
[7] C. Huitema, *Routing in the Internet,* second ed. Prentice Hall PTR, 2000.
[8] J. McQuillan, I. Richer, and E. Rosen, "The New Routing Algorithm for the ARPANET," *IEEE Trans. Comm.,* vol. 28, no. 5, pp. 711-719, May 1980.
[9] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Boston University Representative Internet Topology Generator," http://cs-pub.bu.edu/brite/index.html, Mar. 2001.
[10] J. Moy, "OSPF version 2," Internet Draft RFC-2178, July 1997.
[11] P. Narvaez, K.-Y. Siu, and H.-Y. Tzeng, "Fault-Tolerant Routing in the Internet without Flooding," *Dependable Network Computing,* D. Aversky, ed., pp. 193-206, Kluwer Academic, 2000.
[12] R. Perlman, "A Comparison between two Routing Protocols: OSPF and IS-IS," *IEEE Networks,* vol. 5, pp. 18-24, Sept. 1991.
[13] M. Steenstrup, *Routing in Communications Networks.* Prentice Hall, 1995.
[14] A.S. Tenenbaum, *Computer Networks,* fourth ed. Prentice Hall, 2003.

Jie Wu received the BS and MS degrees from Shanghai University of Science and Technology (now Shanghai University), China, in 1982 and 1985, respectively, and the PhD degree from Florida Atlantic University in 1989. He is now a professor in the Department of Computer Science and Engineering, Florida Atlantic University. He has published more than 200 papers in various journals and conference proceedings. His research interests are in the areas of wireless networks and mobile computing, routing protocols, fault-tolerant computing, and interconnection networks. He is the author of the text *Distributed System Design* (CRC Press). He is a member of the ACM and the IEEE Computer Society and a senior member of the IEEE.

Fei Dai received the BS and MS degrees in computer science in 1990 and 1993, respectively, from Nanjing University, China. He worked as a senior programmer in the China Greatwall Computer Group from 1993 to 1996 and as a software architect and team leader in the IT department of J&A Securities Co., Ltd. from 1996 to 2001. He is currently a PhD student in the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton. He is a student member of the IEEE and the IEEE Computer Society.

Xiaola Lin received the BS and MS degrees in computer science from Peking University, Beijing, China, and the PhD degree in computer science from Michigan State University, East Lansing. He is currently an associate professor in the Department of Computer Science, City University of Hong Kong. His research interests include parallel and distributed computing, design and analysis of algorithms, and computer network. He is a member of the IEEE and the IEEE Computer Society.

Jiannong Cao received the BSc degree in computer science from Nanjing University, China, in 1982, and the MSc and PhD degrees from Washington State University in 1986 and 1990, all in computer science. He joined the Hong Kong Polytechnic University in 1997, where he is currently an associate professor. His research interests include parallel and distributed computing, networking, mobile computing, fault tolerance, and distributed programming environments. He is a member of the IEEE Computer Society, IEEE Communication Society, IEEE, ACM, and American Association for the Advancement of Science (AAAS).

Weijia Jia received BSc, MSc, and PhD degrees, all in computer science in 1982, 1984, and 1993, respectively. In 1995, he joined the City University (City U) of Hong Kong, where he is currently an (adjunct) associate professor in the Department of Computer Science and Department of Computer Engineering and Information Technology. His research interests include computer network, distributed systems, multicast and anycast QoS routing protocols for the Internet, and wireless communications. He is a member of the IEEE and IEEE Computer Society.