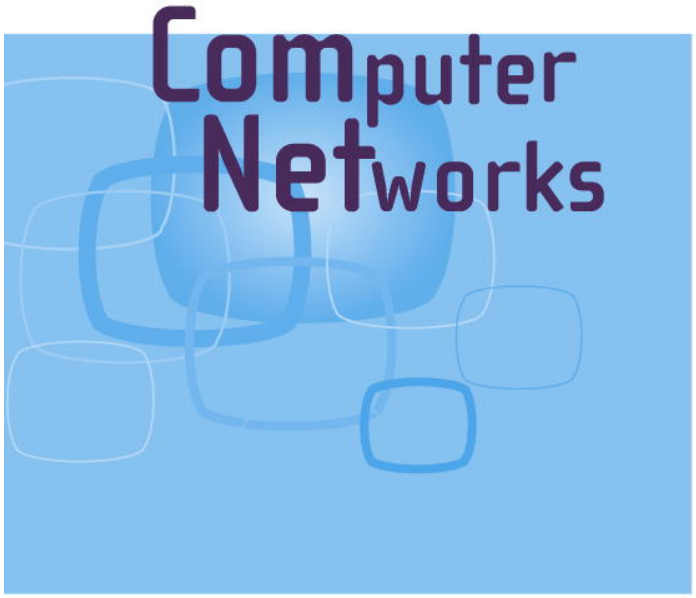
Volume 52 Issue 3 22 February 2008 ISSN 1389-1286



Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 52 (2008) 593–609
Computer Networks
www.elsevier.com/locate/comnet

Mitigating transient loops through interface-specific forwarding

Srihari Nelakuditi [a,*], Zifei Zhong [a], Junling Wang [a], Ram Keralapura [b], Chen-Nee Chuah [b]

[a] Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208, United States
[b] Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA 95616, United States

Received 20 February 2007; received in revised form 1 August 2007; accepted 24 October 2007
Available online 7 November 2007

Responsible Editor: I.F. Akyildiz

Abstract

Under link-state routing protocols such as OSPF and IS–IS, when there is a change in the topology, propagation of link-state advertisements, path recomputation, and updating of forwarding tables (FIBs) will all incur some delay before traffic forwarding can resume on alternate paths. During this convergence period, routers may have inconsistent views of the network, resulting in transient forwarding loops. Previous remedies proposed to address this issue enforce a certain order among the nodes in which they update their FIBs. While such approaches succeed in avoiding transient loops, they incur additional message overhead and/or increased convergence delay. We explore an alternative approach, *loopless interface-specific forwarding* (LISF), that mitigates transient loops by forwarding a packet based on both its incoming interface and destination address. LISF needs to compute and update interface-specific instead of interface-independent forwarding tables. But it requires neither the synchronization of FIB updates at different nodes nor the modification of the existing link-state routing mechanisms. LISF is easily deployable with current routers if they already maintain a FIB at each interface for lookup efficiency. This paper presents the LISF approach, illustrates its strengths and limitations, discusses four alternative implementations of it and evaluates their performance.

Keywords: Link state routing; Transient forwarding loops; Interface-specific forwarding

1. Introduction

The widely used link-state routing protocols such as OSPF and IS–IS distribute link states so that each router has a complete description of the network topology. When a link fails due to a faulty interface or a fiber cut [1], the nodes adjacent to the failure detect it and flood this change in link state to the rest of the network so that all the routers can recompute their routing tables. These routing table entries are then pushed onto Forwarding Information Base (FIB) at all line cards. Each of these steps – failure detection, link-state propagation, routing table recomputation and FIB updates – incur some delay. Only after these steps are complete, packets, for which the shortest paths to their destinations are affected by the failed link, are guaranteed to be forwarded correctly along the new alternate paths. The interval between the failure

* Corresponding author.
E-mail address: srihari@cse.sc.edu (S. Nelakuditi).

1389-1286/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2007.10.012

detection and the FIB updates at all the routers, is referred to as the *convergence delay*. During this period, routers may have inconsistent views of the network and therefore can cause forwarding loops [2]. While these loops last for only a short time and their effect is mitigated by the TTL field in IP datagrams, they can still overwhelm high capacity links and render them unusable. A looping packet consumes bandwidth, increasing the traffic over a link by as much as 128 times, causing the link to congest, and thereby impacting other traffic that is not supposed to be affected by the link state change [3]. Therefore, it is desirable to avoid any forwarding loops even if they are only transient.

Several remedies for the transient looping problem have been suggested in the literature [3–6] and an IETF working group has been addressing this issue [7]. Path locking with safe neighbors approach [4] categorizes routes into three types A, B, or C, and installs new routes for B and C types after a fixed configurable delay such that delay for type B is greater than delay for type C routes. While this approach decreases the likelihood of loops, it does not completely eliminate them. Moreover, it introduces additional delays in the installation of new routes compounding the convergence delay. A loop-free path-finding algorithm proposed in [5] blocks a potential loop when it detects that a loop can be formed. To achieve this, a router first reports to all its neighbors that its distance to reach the destination is infinity, and then waits for those neighbors to acknowledge its message with their own distances and predecessor information before updating its successor in the forwarding table. A similar method has been proposed in [6,8], where the forwarding table updates in the network are ordered such that a node updates its forwarding table only after all its neighbors that use the node to reach different destinations through the failed link update their forwarding tables. Recently, another technique was proposed in [9] that progressively changes the metric associated with the affected link while ensuring loop-freedom at each step of progression, without requiring modifications to OSPF or IS–IS, making it readily deployable. Although these mechanisms completely eliminate transient forwarding loops, they are designed to handle only managed link state changes or unplanned changes in the state of protected links.

We explore an alternate approach [10] – *loopless interface-specific forwarding* (LISF) – that exploits the existence of one forwarding table per interface to mitigate transient loops even in case of unplanned changes to unprotected links. When all the routers in a network have the same view of the network, there would not be a forwarding loop. Only in the presence of discrepancies in the views of different routers, a packet might get caught in a loop. In such a case, the packet would have arrived through an *unusual* interface of at least one of the routers involved in the loop. Therefore, a forwarding loop can be avoided if the packet were to be discarded in such a scenario rather than forwarded to the usual next hop. LISF does precisely that by selectively discarding packets that arrive through unusual interfaces. The key advantages of LISF are that it mitigates transient loops without increasing the convergence delay and without employing any additional mechanisms to synchronize the forwarding table updates at different nodes in the network.

The contributions of this paper are as follows. It proposes LISF approach for mitigating transient loops and illustrates its strengths and limitations. It presents four implementation choices of the LISF approach and discusses their relative merits. It proves that all the LISF methods guarantee loop-freedom in a network with symmetric link weights in case of a change in the status of a single link or node. It also shows that even the most aggressive LISF method, that discards all packets that arrive through unusual interfaces, cannot completely eliminate all possible loops in a network with asymmetric links even in case of single link state change. Similarly, it demonstrates that loops can happen under all LISF methods in case of multiple failures even in a network with symmetric links only. However, results of the evaluation of LISF methods on real networks indicate that the proposed approach works well in practice. Note that the key contribution of this paper lies more in its exploration of the solution space for avoiding transient forwarding loops than its presentation of a better alternative to the existing schemes.

The rest of this paper is structured as follows. In Section 2, we illustrate the problem of transient loops. Our LISF approach for mitigating forwarding loops and four possible implementations of it are described in Section 3. In Section 4, we prove that the proposed LISF methods prevent loops when the status of a single link or node changes in networks with symmetric links. The possibility of loops in case of a change in the status of a single link in networks with asymmetric links or multiple links

in networks with symmetric links is demonstrated in Section 5. The results of our simulations evaluating the LISF methods are presented in Section 6. We finally conclude the paper in Section 7.

2. Transient looping problem and existing approaches

We now illustrate the occurrence of transient loops using an example. Consider the topology shown in Fig. 1(a), where each directed link is labelled with its weight. For the purpose of illustration, let us assume that all the nodes have similar characteristics with a *failure detection time* of 50 ms, a *failure notification time* between neighboring nodes of 100 ms, and *route computation and update time* of 400 ms at a node (100 ms for nodes that are not affected by the failure for a given destination to reflect the relation between the number of affected prefixes and the corresponding FIB update time).

Consider a scenario where link E–D fails at time 0 s. We examine how this failure impacts the forwarding of packets from source node A to destination node D. Table 1 summarizes the routing events under the traditional OSPF and the corresponding changes in the packet's forwarding path from node A to node D. The resulting convergence delay (i.e., the total time for all the nodes in the network to converge after the failure) is 0.65 s, and the service disruption time (i.e., the total time for which the service between A and D is disrupted due to the failure) is 0.55 s. During the interval between the forwarding table updates in nodes E and F (i.e., between 0.45 s and 0.55 s), both the nodes have a different view of the network, resulting in a forwarding loop.

To avoid transient loops during the convergence after a planned link failure or an unplanned failure of a protected link, a method referred to as oFIB was proposed in [6,8] that ensures ordered installation of forwarding table entries by exchanging messages between neighboring nodes. For the same purpose, another technique proposed in [9], which we refer to as incU, that progressively reconfigures a link metric while ensuring loop-free forwarding at each step of the sequence. Our objective is to develop a scheme that combines the best features of OSPF, oFIB, and incU while handling unplanned link state changes of unprotected links. It should: (i) allow each node to update its FIB asynchronously like OSPF without any additional communication overhead and like incU without any coordination overhead; (ii) like oFIB and incU, enable the network to converge without any transient forwarding loops; and (iii) like oFIB and OSPF, update the FIB exactly once per link state change. Such a scheme would ideally respond to the failure of E–D as shown in Table 1. Its behavior would be effectively similar to OSPF except that a packet is dropped if it would loop otherwise (as in the case of packets destined to D from F or E during the interval from 0.45 s to 0.55 s). Consequently, the ideal scheme would have the convergence delay of 0.65 s and service disruption time of 0.55 s while also avoiding forwarding loops. In the following sections, we present and evaluate a scheme that closely approximates this ideal behavior.

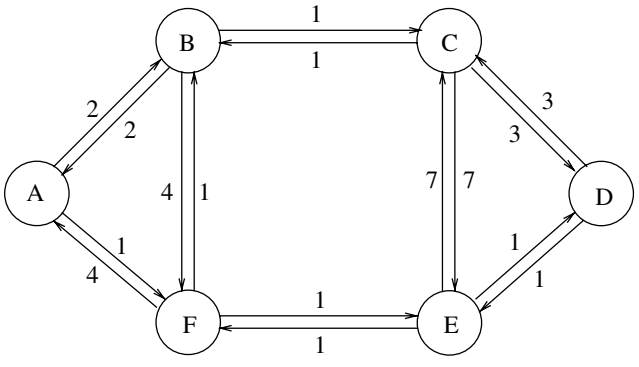


Fig. 1. Topology used for illustration.

3. Our approach

Our approach for avoiding forwarding loops is based on the notion of *interface-specific forwarding*, where a packet's forwarding depends on the incoming interface in addition to the destination address.

Table 1
Summary of routing events under OSPF and LISF

Time (s)	Event	Packets from A to D	
		OSPF	LISF
0	Failure of link E–D	A–F–E-drop	A–F–E-drop
0.05	D, E: failure detected	A–F–E-drop	A–F–E-drop
0.15	C, F: failure notified	A–F–E-drop	A–F–E-drop
0.25	A, B: failure notified	A–F–E-drop	A–F–E-drop
0.35	B: route update	A–F–E-drop	A–F–E-drop
0.45	D, E: route update	A–F–E–F-···**LOOP**	A–F–E–F-**DROP**
0.55	C, F: route update	A–F–B–C–D	A–F–B–C–D
0.65	A: route update	A–B–C–D	A–B–C–D

Table 2
Interface-independent forwarding tables at node B

		destination				
		A	C	D	E	F
interface	A→B	A	C	C	A	A
	C→B	A	C	C	A	A
	F→B	A	C	C	A	A

In this section, we first briefly explain interface-specific forwarding and argue how it can be exploited to avoid loops. We then present four methods of computing interface-specific forwarding table entries and illustrate the differences between these methods in terms of loop-avoidance and computational complexity.

3.1. Interface-specific forwarding

A packet in an IP network is traditionally routed based on its destination address alone regardless of its source address or the incoming interface. Therefore, a single forwarding table that maps a destination address to a next hop and an outgoing interface is sufficient for current routers to perform IP datagram forwarding. Nevertheless, routers nowadays maintain a forwarding table at each line card of an interface for lookup efficiency [11]. However, all these forwarding tables at each interface are identical, i.e., these forwarding tables are interface-independent. For example, interface-independent forwarding tables at node B of Fig. 1 are as given in Table 2.

Instead of maintaining the same forwarding table at each interface, it is possible to avoid forwarding loops by making the entries of these tables *interface-specific*. Table 3 gives the possible set of interface-specific forwarding table entries at node B of Fig. 1. Each entry is marked with '–', **X**, or a next-hop node. The entries marked '–' are obviously never used. The entries marked **X** are not referenced when there is no failure and all nodes in the network have the same consistent view. For example, in Fig. 1, a packet with destination D should not arrive at B from any of its neighbors since B is not the next hop for them. Similarly, B should not receive from A, a packet destined for F, since A is along the path from B to F. However, in the presence of link failures and inconsistent forwarding tables at different nodes (during the convergence period), a packet may arrive at a node through an unusual interface. Interface-specific forwarding enables special treatment of such packets that arrive through unusual interfaces without introducing any changes to the forwarding plane of the current network infrastructure. Here, we study how interface-specific forwarding can be exploited for the purpose of avoiding loops during the convergence period after a link state change in the network.

Table 3
Interface-specific forwarding tables at node B

		destination				
		A	C	D	E	F
interface	A→B	–	C	**X**	**X**	**X**
	C→B	A	–	**X**	**X**	A
	F→B	A	C	**X**	**X**	–

3.2. Loopless interface-specific forwarding

It is clear that under link-state routing, when all the routers in a network have the same view of the network, there would not be a forwarding loop. Only in the presence of discrepancies in the views of different routers, a packet might get caught in a loop. However, in such a case, under interface-specific forwarding, the packet would have arrived through an unusual interface of at least one of the routers involved in the loop. So a forwarding loop can be avoided if the packet were to be discarded in such a scenario rather than forwarded to the

Table 4
Notation used in this paper

$\mathscr{V}$	Set of nodes in the network
$\mathscr{E}$	Set of links in the network
$\mathscr{R}_i^d$	Set of next-hops from node i to destination d
$\mathscr{F}_{j\to i}^d$	Set of next hops to d for packets arriving through the interface $j \to i$
$\mathscr{P}_i^d$	Set of nodes along any of the shortest paths from i to d
$\mathscr{C}_i^d$	The cost of the shortest path
ℓ	The link with state change
$\mathscr{T}^D$	Shortest path tree rooted at node D
$S(i,\mathscr{T})$	Subtree of $\mathscr{T}$ below node i
$S^\ell(i,\mathscr{T})$	$S(i,\mathscr{T})$ containing only nodes affected by state change of ℓ
A_i, A'_i	Nodes that are aware of state change of ℓ
B_i, B'_i	Nodes that are unaware of state change of ℓ

Table 5
Differences in LISF methods in discarding a packet to d arriving at i from j

Method	Discard condition	Discard criterion
PIng-POng	$j \in \mathscr{R}_i^d$	In and out interfaces are same
CYCLe	$j \in \mathscr{P}_i^d$	Previous node along the path
NO Forward Progress	$\mathscr{C}_{\mathscr{R}_i^d}^d \geqslant \mathscr{C}_j^d$	No forward progress
UNusal INterface	$i \notin \mathscr{R}_j^d$	Unusual incoming interface

usual next hop. We refer to this approach of avoiding forwarding loops by selectively discarding packets that arrive through unusual interfaces as *loopless interface-specific forwarding* (LISF).

Ideally, a packet should be discarded by a router only if its forwarding would definitely result in a loop. However, with only its own local view of the network, a router cannot always determine the actual forwarding path of a packet with certainty. Therefore, the design challenge of LISF is to ensure loop-freedom without unnecessarily discarding packets. In this paper, we study several implementation choices of LISF, ranging from conservative discarding of packets only if there would certainly be a loop otherwise but forwarding even if there could be a loop, to aggressively discarding of packets whenever there could be a loop even if there may not actually be a loop.

We now present four different LISF methods. The difference between them lies in which of the entries marked **X** in Table 3 are set to $\ominus$, meaning *discard*. These methods are named according to the criterion they use to discard a packet. The notation used here is listed in Table 4. The operation of these methods, when a packet for destination d arrives at node i from neighbor j, is summarized in Table 5 and elaborated in detail below. It should be noted that, under LISF, *a node i makes packet forwarding/discarding decisions based solely on its own view of the network*.

Before we proceed to describe various LISF methods, we note that earlier, we proposed an IP fast reroute scheme based on interface-specific forwarding, referred to as failure inferencing based fast rerouting (FIFR) [12]. The difference between FIFR and LISF is in how entries marked **X** are overwritten. FIFR replaces some of these entries with alternate next-hops for rerouting packets, whereas LISF resets them to $\ominus$ for discarding packets. Compared to FIFR, LISF is less complex and much simpler to implement as it involves invalidating some forwarding entries instead of identifying suitable alternate entries. It does not appear feasible to simultaneously achieve both loop-free local rerouting with FIFR and loop-less global convergence with LISF. However, LISF is compatible with other IP fast reroute schemes such as not-VIA [13] and MRC [14] since LISF forwards in consistence with the topology associated with the destination address and the routing configuration of the packet. For example, consider a packet destined to d being rerouted by a node x, adjacent to a failed link x–y, to a not-VIA address z of the next-hop node y. Whereas the packet might be discarded for avoiding a loop if it were addressed to either y or d, due to the semantics of address z, it's arrival would not be deemed unusual at any node along the path from x to z. Therefore, under LISF, a packet with not-VIA address is forwarded as expected by the not-VIA scheme.

3.2.1. Discard if PIng-POng (PIPO)

Discard a packet if its incoming and outgoing interfaces are the same, i.e., $\mathscr{F}_{j\to i}^d = \ominus$ if $j \in \mathscr{R}_i^d$.

PIPO discards a packet only when it arrives at a node from its next hop node, i.e., along a reverse shortest path to the destination. It is the most conservative of all the methods listed here as it discards a packet only when there is indeed a loop. Otherwise, without PIPO, in such a scenario, packets will ping-pong between two neighboring nodes. For example, in Table 6, a packet to destination D arriving at B from C is discarded by PIPO since C is the next hop to D from B. PIPO is also the simplest since it incurs no additional overhead for computing interface-specific forwarding table entries beyond the currently used Dijkstra's algorithm for computing interface-independent forwarding tables. However, PIPO can ensure loop-freedom only when two nodes are involved in a loop, which is the case when links are *symmetric* (bidirectional with equal weights in both directions) and inconsistency in the views among routers is limited to a single link's state.

3.2.2. Discard if CYCLe (CYCL)

Discard a packet if the previous node appears along a shortest path from this node to the destination, i.e., $\mathscr{F}_{j\to i}^d = \ominus$ if $j \in \mathscr{P}_i^d$.

CYCL discards a packet when it arrives from a node which falls along a shortest path from this

Table 6
Interface-specific forwarding tables at B under different LISF methods

interface	destination A	C	D	E	F
A→B	–	C	**C**	⊖	⊖
C→B	A	–	⊖	**A**	A
F→B	A	C	**C**	**A**	–

(**PIPO**)

interface	destination A	C	D	E	F
A→B	–	C	**C**	⊖	⊖
C→B	A	–	⊖	**A**	A
F→B	A	C	**C**	⊖	–

(**CYCL**)

interface	destination A	C	D	E	F
A→B	–	C	⊖	⊖	⊖
C→B	A	–	⊖	**A**	A
F→B	A	C	⊖	⊖	–

(**NOFP**)

interface	destination A	C	D	E	F
A→B	–	C	⊖	⊖	⊖
C→B	A	–	⊖	⊖	A
F→B	A	C	⊖	⊖	–

(**UNIN**)

node to the destination. When the links are symmetric, CYCL behaves just like PIPO. Only when links are asymmetric and the resulting paths are asymmetric, the operation of CYCL could be different from PIPO. With a less stringent condition than PIPO, CYCL may discard a packet even when there may not actually be a loop, but at the same time, it can avoid some loops that are not avoided by PIPO. For example, in Table 6, a packet to destination E arriving at B from F is forwarded by PIPO to A resulting in a loop whereas it will be discarded by CYCL since F is along the shortest path from B to E. The computational complexity of CYCL is similar to that of PIPO as both require only a single shortest path tree computation.

3.2.3. Discard if NO Forward Progress (NOFP)

Discard a packet if there is no forward progress towards its destination from its previous hop to the next hop of this node, i.e., $\mathscr{F}^d_{j\to i} = \ominus$ if $\mathscr{C}^d_{\mathscr{R}^d_i} \geqslant \mathscr{C}^d_j$.

NOFP discards a packet if its previous hop is not farther from its destination than the next hop of this node.[1] In such a case, there is a potential for a loop and NOFP discards such packets. For example, in Table 6, a packet to destination D arriving at B from F is discarded by NOFP since the cost from F to D is 2 whereas the cost from the next hop C is three. This is in contrast to both PIPO and CYCL which forward the packet to C. While such discarding by NOFP seems unnecessary, NOFP can prevent more loops than PIPO and CYCL even when links are asymmetric and the state of multiple links change simultaneously. For example, in topology shown in Fig. 1(b), suppose link F–E failed. Further, assume that all nodes except nodes B and C are notified of the failure and their forwarding tables reflect the failure. In this scenario, under PIPO and CYCL, a packet from A to D is forwarded along a loop A–B–G–C–A–B$\cdots$. On the other hand, under NOFP, it is discarded by B since, according to B's view, the cost of three from next hop G to D is not smaller than the cost from A to D which is also three. A straightforward method to implement NOFP requires a computation of $O\left(\frac{|\mathscr{E}|}{|\mathscr{V}|}\right)$ times Dijkstra on the average (to compute the shortest path trees rooted at each neighbor), whereas PIPO and CYCL have the same complexity as Dijkstra.

3.2.4. Discard if UNusual Incoming Interface (UNIN)

Discard a packet if it arrives at a node through an unusual incoming interface, i.e., $\mathscr{F}^d_{j\to i} = \ominus$ if $i \notin \mathscr{R}^d_j$.

UNIN discards a packet if it is not supposed to arrive at this node from a neighbor node according

[1] A similar rule for preventing forwarding loops is also used in [15].

Table 7
Differences among LISF methods in discarding packets arriving at node B

From	To	Failed link	PIPO	CYCL	NOFP	UNIN
C	D	C–D	Discard	Discard	Discard	Discard
F	E	F–E	Forward to A	Discard	Discard	Discard
F	D	F–E	Forward to C	Forward to C	Discard	Discard
C	E	C–D	Forward to A	Forward to A	Forward to A	Discard

to this node's view of the network. As mentioned before, an unusual arrival of a packet indicates the existence of inconsistencies among routers and UNIN discards all such packets to prevent any potential loops. Effectively, UNIN sets all the entries marked **X** in Table 3 to $\ominus$ as in Table 6. UNIN is the most aggressive of all LISF methods in avoiding loops at the expense of discarding packets that may not be caught in a loop. In terms of computational complexity, UNIN also requires $O\left(\frac{|\mathscr{E}|}{|\mathscr{V}|}\right)$ times Dijkstra.

3.2.5. Differences between LISF methods

The difference between the actions of the above four methods is clearly evident in the presence of failures of links C–D and F–E in Fig. 1 as shown in Table 7. Here it is assumed that B is not yet aware of the failed links and the forwarding tables of B do not reflect the change. When only F–E fails, packets from F to D arriving at B are discarded by NOFP and UNIN whereas PIPO and CYCL forward them along a loop-free path via C. Essentially these methods achieve different trade-offs between loop-avoidance and packet-discarding, which can be summed up as follows:

- packet looping probability: PIPO $\geqslant$ CYCL $\geqslant$ NOFP $\geqslant$ UNIN,
- packet discard probability: PIPO $\leqslant$ CYCL $\leqslant$ NOFP $\leqslant$ UNIN,
- network convergence delay: PIPO = CYCL = NOFP = UNIN.

4. Proof of loop-free property of LISF

We now prove that the LISF methods described in the previous section ensure loop-freedom when at most a single link or node state change is being propagated in a network with symmetric links. It is clear that if PIPO is loop-free, the other three methods are also loop-free since whenever PIPO discards a packet, they would too. Therefore, it suffices to provide the proof for PIPO which is given below. In the following, we first prove the loop-freedom under PIPO in case of a change in the status of a single link and then extend the proof to cover the case of a change in the status of a single node also.

4.1. Change in the state of a single link

Let ℓ be the link whose state has changed. A change in the state of ℓ could be due to either an increase or a decrease in its cost. A link-down event, which means that a link is broken and thus no packets could be forwarded through it, could be viewed as an increase in its cost to infinity (∞); a link-up event, which means that a broken link becomes working, could be viewed as a decrease in its cost from infinity to a certain finite value. If the cost of ℓ increases, some shortest paths that were previously traversing ℓ may not contain ℓ any more, whereas more shortest paths pass through ℓ when its cost decreases.

Since the forwarding to a destination is independent of forwarding to other destinations, we prove that a packet for a specific destination D will not loop under PIPO in case of a single link state change. We use the notation listed in Table 4. Let $\mathscr{T}^D$ be the shortest path tree with undirected links rooted at destination D. Note that this tree contains the shortest paths, possibly multiple equal cost paths, from every node to D since the links are symmetric. We use $S(i,\mathscr{T})$ to denote the subtree of $\mathscr{T}$ below the node i, *before* the change if link cost *increases*, whereas $S(i,T)$ denotes the subtree of T below the node i *after* the change if link cost *decreases*. We let $S^{\ell}(i,\mathscr{T})$ denote the subtree which only contains the nodes whose shortest paths to D are affected by the state change of ℓ.

If ℓ *not in* $\mathscr{T}^D$, the forwarding path to D is the same with or without the failure of ℓ. Therefore, the forwarding to D is consistent at every node along the path and hence packets destined for D will not be caught in a loop due to the failure of ℓ. In the rest of the proof, it is assumed that ℓ *in* $\mathscr{T}^D$.

Let ℓ = F–L and F be the upstream node to L on the path from F to D as in Fig. 2. When the state of ℓ changes, only those nodes in $S^{\ell}(F,\mathscr{T}^D)$ will be affected (i.e., all nodes outside the subtree will forward along the same path with or without F–L for destination D). Now consider a packet originating from any node in $S^{\ell}(F,\mathscr{T}^D)$. That packet may be

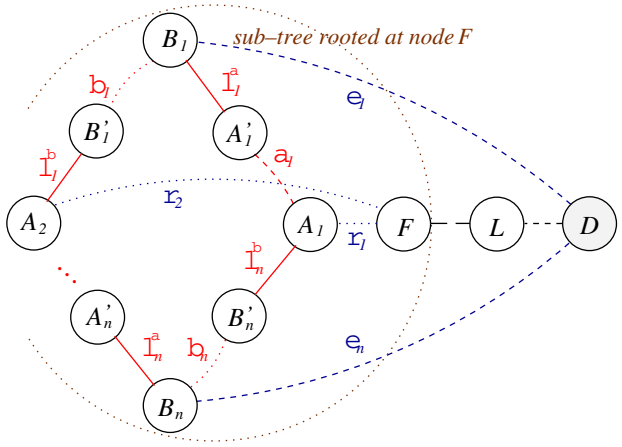


Fig. 2. Scenarios for illustrating loop-freedom under PIPO.

forwarded to node F and then get rerouted, or get rerouted somewhere in $S^{\ell}(F,\mathscr{T}^D)$. Once the packet goes outside the subtree, it will be forwarded to D consistently. So the only possible loop is the one consisting of nodes which are all within $S^{\ell}(F,\mathscr{T}^D)$. Thus, to prove loop-freedom under PIPO, we only need to show that there will not be a loop within $S^{\ell}(F,\mathscr{T}^D)$ when the state of ℓ changes.

Suppose there is a loop in subtree $S^{\ell}(F,\mathscr{T}^D)$. The loop must contain some nodes that intend to forward packets through ℓ and some that forward packets to go around ℓ. This forwarding decision discrepancy exists among nodes because at some time within a link-state propagation period it is possible that a portion of nodes in $S^{\ell}(F,\mathscr{T}^D)$ get to know about the link state change of ℓ while the other nodes are still not aware of it. If the link state change is a cost increasing event, nodes that are aware of this link state change forward packets to circumvent ℓ, while nodes that are unaware of the link state change forward packets to go through ℓ; under a cost decreasing event, however, the nodes that are unaware could forward packets to go around the link ℓ while the aware nodes may forward packets through ℓ. Thus, irrespective of the change, the nodes can be categorized into those that forward through ℓ, and those that forward around ℓ. The rest of the proof is based on this categorization of nodes into two distinct groups and therefore is applicable to both cost decreasing and increasing events.

We use Fig. 2, where each edge labelled with its cost, as an illustration to aid the proof. Pick an arbitrary node, A_1, in the loop that forwards packets to go around ℓ. As shown in Fig. 2, suppose the packet starts from A_1 and is routed along zero or more "A" nodes, that forward packets to circumvent ℓ in consistence with A_1, and reaches A'_1, the last "A" node in this stretch. A'_1 forwards the packet to B_1, a node that is unaware of the failure. Note that from A_1 to B_1, the forwarding is consistent. We use the *dashed* line to indicate subpath of the route as per the views of "A" nodes. B_1 then routes the packet differently from that intended by "A" nodes: instead of towards D via the dashed path, it forwards the packet via the *dotted* path towards B'_1, i.e., it chooses $B_1 \rightsquigarrow B'_1 \rightarrow A_2 \rightsquigarrow F \rightarrow L \rightsquigarrow D$. Similarly, the packet is routed differently at A_2, which intends to forward it to D through the next stretch of dashed path. This process continues until the packet is forwarded back to A_1 by B_n $(n \geqslant 1)$.

Now we show that there is a contradiction if such a loop exists. Note that in such a loop, any B_i cannot forward a packet back to A'_i after getting the packet from A'_i (e.g., $A'_{i+1} \neq A'_i$), as the packet will get dropped under PIPO when being forwarded back to A'_i. Then consider the routing decision at node B_i $(1 \leqslant i \leqslant n)$. Since the node B_i chooses the path $B_i \rightsquigarrow A_{i+1} \rightsquigarrow F$ over $B_i \rightsquigarrow A_{i-1} \rightsquigarrow F$, we have

$$\sum_{i=1}^{n-1}(b_i + l_i^b + r_{i+1}) < \sum_{i=1}^{n-1}(a_i + l_i^a + r_i), \tag{1}$$

$$b_n + l_n^b + r_1 < a_n + l_n^a + r_n. \tag{2}$$

Adding them together, we have

$$\sum_{i=1}^{n}(b_i + l_i^b) < \sum_{i=1}^{n}(a_i + l_i^a). \tag{3}$$

Similarly, consider the routing decision made at node A_i. Since it chooses the path $A_i \rightsquigarrow B_i \rightsquigarrow D$ over the path $A_i \rightsquigarrow B_{i-1} \rightsquigarrow D$

$$\sum_{i=1}^{n-1}(a_{i+1} + l_{i+1}^a + e_{i+1}) < \sum_{i=1}^{n-1}(b_i + l_i^b + e_i), \tag{4}$$

$$a_1 + l_1^a + e_1 < b_n + l_n^b + e_n. \tag{5}$$

Adding them together, we get

$$\sum_{i=1}^{n}(a_i + l_i^a) < \sum_{i=1}^{n}(b_i + l_i^b). \tag{6}$$

Obviously, inequality (6) above contradicts (3). Therefore, a forwarding loop is not possible under PIPO in case of any change in the state of a single link.

4.2. Change in the state of a single node

We now prove that a packet destined for a node D will not loop under PIPO in case of a single node state change, i.e., when a node goes down or comes up. The loop-freedom proof of single node state

change could be derived from that of single link state change. We assume that any single node failure will not partition the whole network. For simplicity, we still use Fig. 2 for illustration. In case that node F fails, all the links connected to F go down, and these links come up after the node recovers from failure. Upon a state change of F, each neighbor of F propagates notification of adjacent link state change to other nodes, and thus each node in the network will receive multiple notifications. Note that when a link is down, a node updates its FIB only after it receives a notification from both nodes adjacent to that link. Therefore, in case of failure of F, a node will know that F is down and updates its FIB only after it receives notifications from all the neighbors of F.

Thus, before all the nodes in $S(F, \mathscr{T}^D)$ finish updating their FIBs, there are only two categories of nodes in $S(F, \mathscr{T}^D)$: those that maintain their old FIBs, which means that they have not updated their FIBs with the information that node F has changed its state; and those who have already calculated new routes by removing/adding node F from/to the base topology and updated their FIBs. During convergence, nodes in one category try to forward through F to destinations, while nodes in the other category intend to forward packets to go around F. With the same method as we used in the previous section, we can prove that no loop could happen in case of a single node state change.

5. Limitations of LISF methods

Previous sections have proved that LISF methods guarantee loop-free convergence under a single link or node state change for a network with symmetric links. In this section, we illustrate that LISF methods may incur forwarding loops during convergence if multiple link failures happen simultaneously or if a single failure happens in a network with asymmetric links. As discussed previously, among all the LISF methods, UNIN is the most aggressive in discarding packets and thus has the least probability of forwarding loops. However, even UNIN cannot prevent forwarding loops completely under these conditions. Fortunately, as we will see in the evaluation section, these forwarding loops will rarely occur when LISF is applied to the networks in real world.

5.1. Single link failure with asymmetric links

We have shown that all the LISF methods prevent loops in case of any single link state change in a network with symmetric links. We now contrast these methods in their ability to handle a single link failure in a network with asymmetric links. Consider the topology shown in Fig. 3 where each directed edge is labelled with its cost. For clarity, links in the opposite direction are not shown here as it is assumed that their costs are so high that they are not included in any of the shortest paths to destination D. Suppose the link A → D goes down and the corresponding link state advertisement is being propagated in the network. During the convergence period, some nodes would have received the notification and updated their FIBs while others have not.

Table 8 lists various possible scenarios, in each scenario the set of nodes that updated their FIBs and the corresponding behavior under each LISF method. In all scenarios, a packet from source A to destination D would loop under PIPO. It is surprising that there exists a scenario where even UNIN cannot prevent a loop. When all nodes except C and H have new FIBs, the packet gets forwarded from A to H consistently along the path A → B → E → F → G → H → D. But H forwards it to C since the packet arrives through the usual

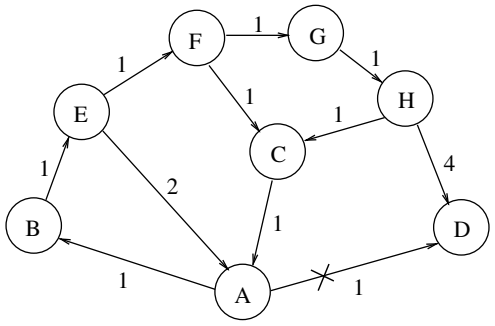


Fig. 3. Topology with asymmetric links to illustrate the limitations of LISF methods.

Table 8
Differences between LISF methods in forwarding from A to D after A → D is down

Nodes with new FIBs	PIPO	CYCL	NOFP	UNIN
A, D	loop(A–B–E–A···)	A–B-drop	A–B-drop	A–B-drop
A, B, D	loop(A–B–E–A···)		A–B–E-A-drop	A–B–E-A-drop
A, B, D, E	loop(A–B–E–F–C–A···)			A–B–E-F-drop
A, B, D, E, F, G	loop(A–B–E–F–G–H–C–A···)			

interface according to its out-of-date FIB. Similarly node C forwards to A. The node A too does not discard it, since A is the usual next-hop from C to D according to the new FIB at A and forwards to B. Thus the packet from A to D, gets caught in a loop $A \rightarrow B \rightarrow E \rightarrow F \rightarrow G \rightarrow H \rightarrow C \rightarrow A \rightarrow B \cdots$. This is because the interfaces at the boundaries of new and old FIBs along the path, i.e., $G \rightarrow H$ and $C \rightarrow A$, are deemed usual. The essence of this illustration is that when a node's view of usual interface remains same before and after a link state change, interface-specific forwarding based approach may not be able to prevent a loop.

5.2. Multiple failures with symmetric links

We now focus on the scenarios of multiple link state changes being propagated in the network simultaneously. Since we have already shown that there is a possibility of looping under LISF in networks with asymmetric links, here we consider a network with symmetric links as shown in Fig. 4. Assume that links A–D and E–H are down. Table 9 lists various possible scenarios, in each scenario the set of nodes whose FIBs reflect the failure of A–D, those that reflect the failure of E–H and the corresponding forwarding behavior of LISF methods. Note that since links are symmetric, there would not be any difference between PIPO and CYCL methods. In the first scenario, when A's FIB reflects the failure of A–D but not E–H, A would forward a packet destined for D along the new shortest path to B. The node B which is aware of the failure of E–H but not A–D drops the packet and avoids a loop under all LISF methods. On the other hand, in the last scenario, even UNIN cannot prevent a loop. The node A forwards the packet to B which in turn forwards it to E. But E, with a view different from A and B, considers this as a usual arrival and forwards it to G, which in turn forwards to F. Node F, which is aware of the failure of E–H only, forwards it to C, since the shortest path from G to D is via F. Node C, which is unaware of any of the failures, forwards it to its usual next hop A, since $F \rightarrow C$ is usual interface to destination D. Thus, the packet arrives back at A forming a loop A–B–E–G–F–C–A–B $\cdots$. Once again, this example illustrates that when a node's view of usual interface remains the same even when a neighbor's view of the network is different, interface-specific forwarding based approach may not be able to prevent a loop. However, our evaluation of these methods on real topologies under multiple failures show that LISF methods would be quite effective in preventing loops in practice.

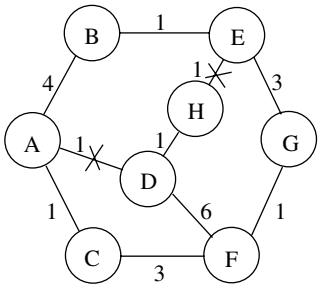


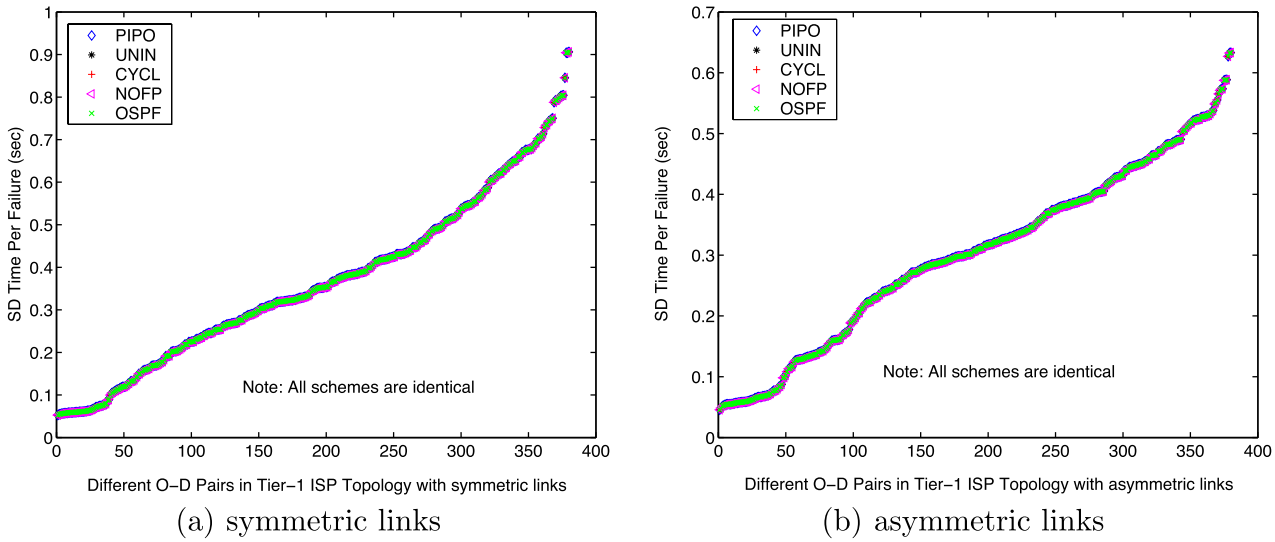
Fig. 4. Topology with symmetric links to illustrate the limitations of LISF methods.

Table 9
Differences in LISF methods in forwarding from A to D after A–D, E–H are down

Aware of A–D	Aware of E–H	PIPO/CYCL	NOFP	UNIN
A, D	B, E, H	A–B-drop	A–B-drop	A–B-drop
A, B, D, E	B, D, E, F, G, H	loop(A–B–E–G–F–C–A$\cdots$)	A–B-drop	A–B-drop
A, B, D, E	D, E, F, G, H	loop(A–B–E–G–F–C–A$\cdots$)		A–B–E–G-drop
A, B, D, E, G	D, E, F, G, H	loop(A–B–E–G–F–C–A$\cdots$)		

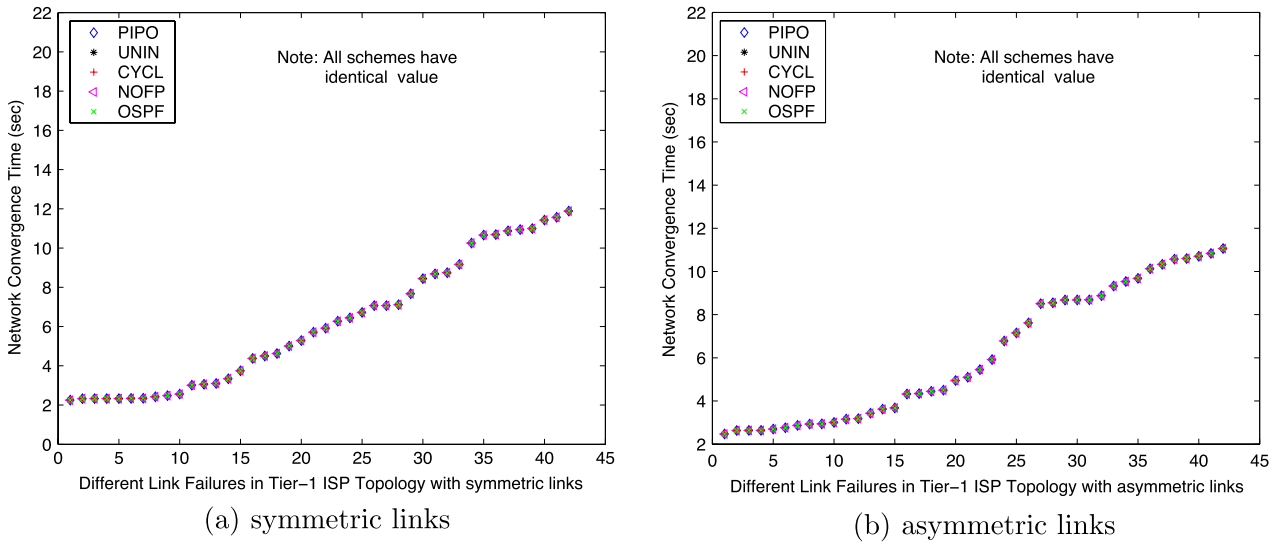
6. Performance evaluation

In this section, we evaluate the performance of LISF methods and compare them against OSPF. We present our evaluation results in three different settings. First, we consider single link failures in a Tier-1 ISP topology with randomly assigned symmetric and asymmetric weights. Then, we use the real Abilene and three other backbone topologies measured by Rocketfuel with the given symmetric weights for evaluation under single node, single and double link failures. Finally, we simulate multiple link and node failures in a Tier-1 ISP topology with both symmetric and asymmetric weights. These experiments under different settings demonstrate that LISF methods prevent a significant number of loops that are possible under OSPF. Furthermore, they reveal the trade-offs of LISF methods between packet-discarding and loop-avoidance.



(a) symmetric links (b) asymmetric links

Fig. 5. Average service disruption time per link failure for various O–D pairs.

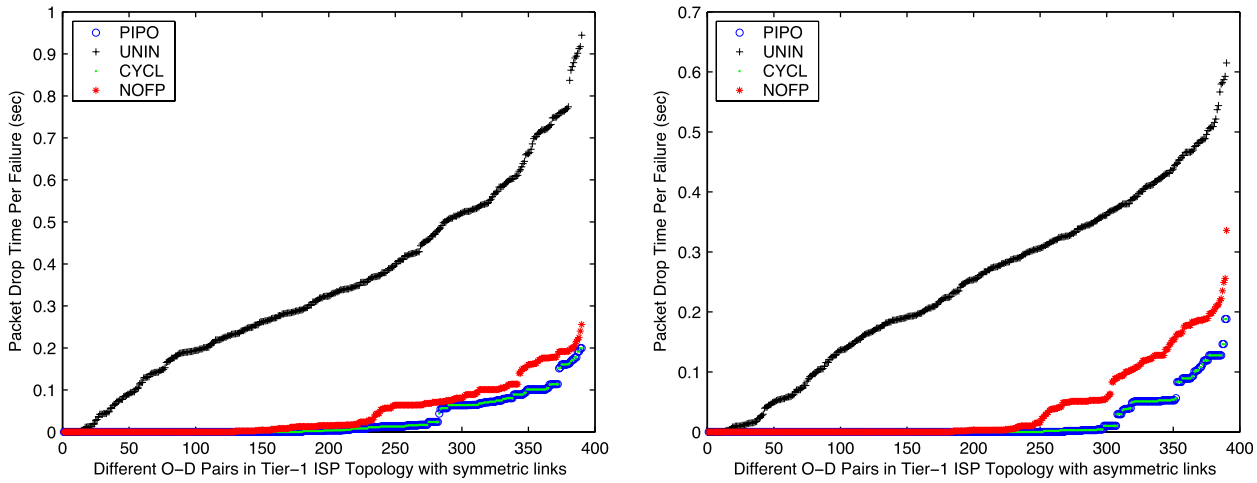


(a) symmetric links (b) asymmetric links

Fig. 6. Network convergence time due to various single link failures.

6.1. Single link failures in a Tier-1 ISP

To evaluate LISF methods, we built a control-plane simulator that emulates intra-domain routing dynamics in the presence of link failures and measures both service disruption (SD) time between different origin–destination (OD) pairs and network convergence (NC) time as presented in [16]. We use a Tier-1 ISP backbone (PoP-level) topology with 20 nodes and 42 links in our simulations which was used earlier in [16]. We assign OSPF link weights to different links by randomly picking integer values between 1 and 5. We consider both symmetric links where $X \rightarrow Y$ has the same OSPF weight as $Y \rightarrow X$, and asymmetric links where the OSPF weight for $X \rightarrow Y$ could be different from $Y \rightarrow X$. The forwarding table at each node includes entries for all the prefixes in the Internet. We assume that the rate of FIB update is 20 entries/ms and the number of prefixes is 161,352. The other parameters in the simulator are set based on the findings in [17]. In every simulation run, we fail each link in the network exactly once. The results presented below represent the effect of all the link failures in a simulation run.



(a) symmetric links (b) asymmetric links

Fig. 7. Average packet discard time per link failure for various OD pairs.

The total time during which a forwarding loop exists under OSPF is observed to be 11.851 s, whereas LISF methods, as expected, have no loops in case of symmetric weights. For the case of asymmetric weights, OSPF has loops for a duration of 6.989 s, while it is 0.056 s for PIPO. In both cases, there are no loops under CYCL, NOFP, or UNIN. These results demonstrate the effectiveness of LISF methods in avoiding loops. We proceed to show that this is achieved not at the expense of larger convergence delay or longer service disruption.

Fig. 5(a) represents the average SD time experienced by all OD pairs due to a single link failure in this network with symmetric links. We can clearly see that the average SD time for a particular OD pair remains the same when LISF is implemented on top of OSPF. This shows that LISF does not add any extra SD time. Fig. 5(b) shows a very similar behavior of average SD time for this network with asymmetric links. Fig. 6(a) and (b) shows the convergence time due to different link failures for the backbone network with symmetric and asymmetric links, respectively. As expected, LISF behaves similar to OSPF, as it does not require any synchronization between nodes for updating their FIBs.

Fig. 7(a) shows the packet discard times due to various LISF methods in the ISP network with symmetric links. As expected, with symmetric link failures, the packet discard times of PIPO and CYCL are identical. While UNIN has much higher packet discard times, NOFP is quite close to PIPO. A similar pattern is observed even with asymmetric links (Fig. 7(b)). However, the packet discard times under PIPO and CYCL are not identical due to the fact that more loops are avoided by CYCL than PIPO.

Table 10
Real backbone topologies used in evaluation

AS number	Name	# of routers	Degrees
	Abilene	12	1.33
1221	Telstra (Australia)	108	2.82
1755	Ebone (Europe)	87	3.70
3967	Exodus (US)	79	3.72

6.2. Single and multiple link failures in Abilene and Rocketfuel topologies

In this section, we evaluate LISF methods using real Internet backbone topologies. Particularly, we use Abilene topology and three other AS backbone topologies measured by Rocketfuel [18]. These networks are summarized in Table 10. Note that all networks contain links with symmetric weights only.

In simulating OSPF and LISF methods, for the sake of simplicity, we assume that once a router is notified of a failure, it will take LSP processing delay, SPF computation time and FIB update delay for a router to update its forwarding tables to handle the failure. We also assume that the LSP packet containing failure notification needs 10 ms propagation delay across each link and LSP processing delay is 20 ms. As before, we assume that the FIB update rate is 20 entries/ms and the number of prefixes is

161,352. The SPF computation time is assumed to be 5 ms for Abilene and 60 ms for other networks.

We ran three types of failure scenarios: 1-link, 1-node, and 2-link failures. For 1-link or 1-node failure scenario, we failed one link or node for each run and examined each source–destination (OD) pair affected by this failure during the convergence time. For each OD pair, we define the delivery ratio as the ratio of the time window during when its packet can be successfully delivered to the total network convergence time. When a packet is not delivered, it is either discarded by a node along the path or caught in a forwarding loop. Correspondingly, we define the drop ratio and loop ratio for an OD pair in a similar fashion. When we collect the delivery, drop and loop ratios for each affected OD pair during a test run, we average these measurements among all OD pairs and report the average values for that test run. We have done 2-link failure simulation a little differently. Since it is almost impossible to run every combination of two link failure, we randomly select 500 2-link failure cases for each network and then run the simulation accordingly.

Figs. 8–10 show the simulation results of OSPF and LISF methods using the real backbone topologies. These figures consistently show that OSPF incur forwarding loops under all simulation scenarios within a range of 2—10%. On the other hand, LISF methods do not have forwarding loops at all for any network under any failure scenario. It shows that even under 2-link failure cases, LISF methods can rarely cause forwarding loops in real network topologies, though in theory loops may be possible in such cases. Meanwhile, PIPO, CYCL and NOFP have exactly the same delivery ratio as OSPF, with only one exception in Fig. 10(b) where NOFP has a slightly lower delivery ratio and a higher drop ratio. We can also see that UNIN, the most aggressive method, has the lowest delivery ratio and the highest drop ratio among all LISF methods, since it discards packets aggressively to prevent loops. This is consistent with the illustrations of the previous sections.

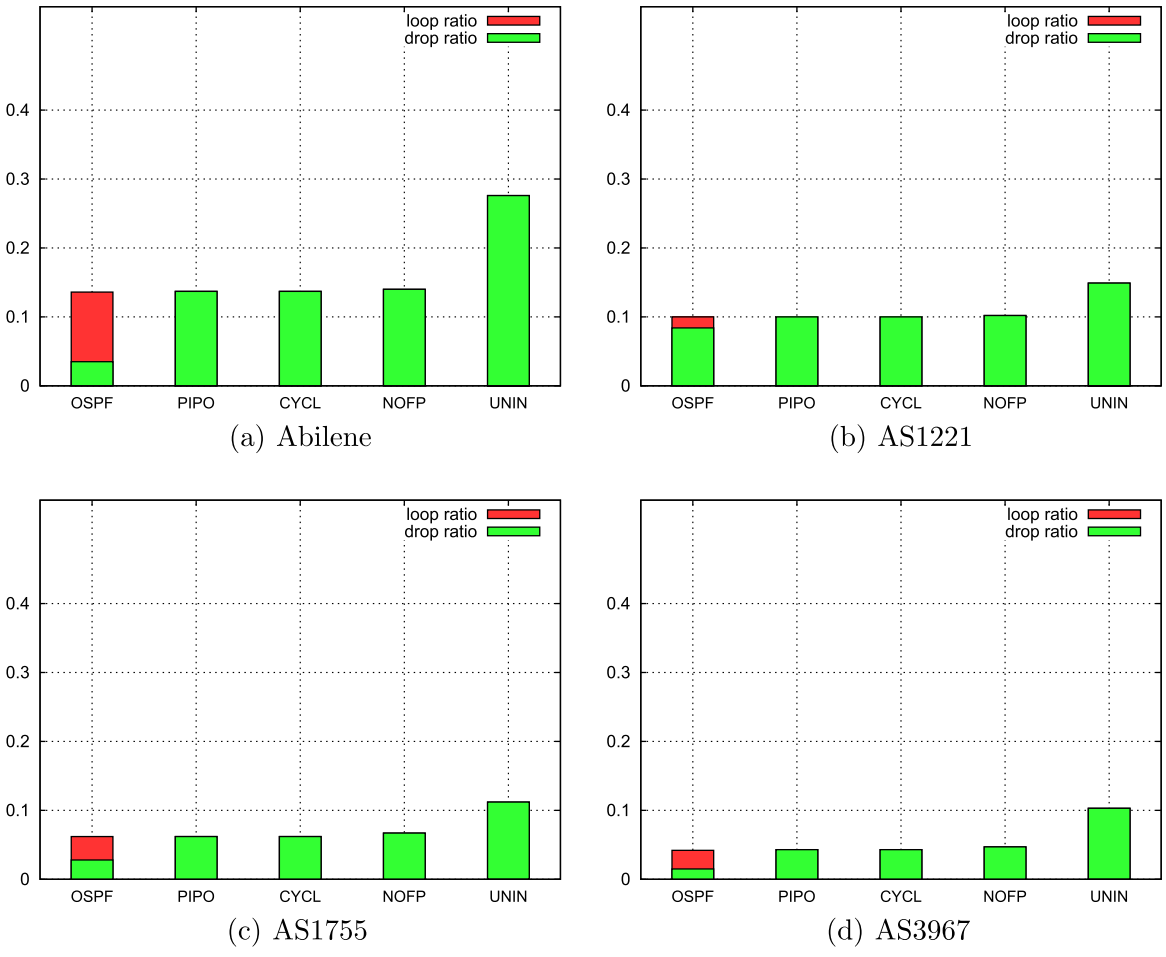


Fig. 8. Performance under 1-link failure.

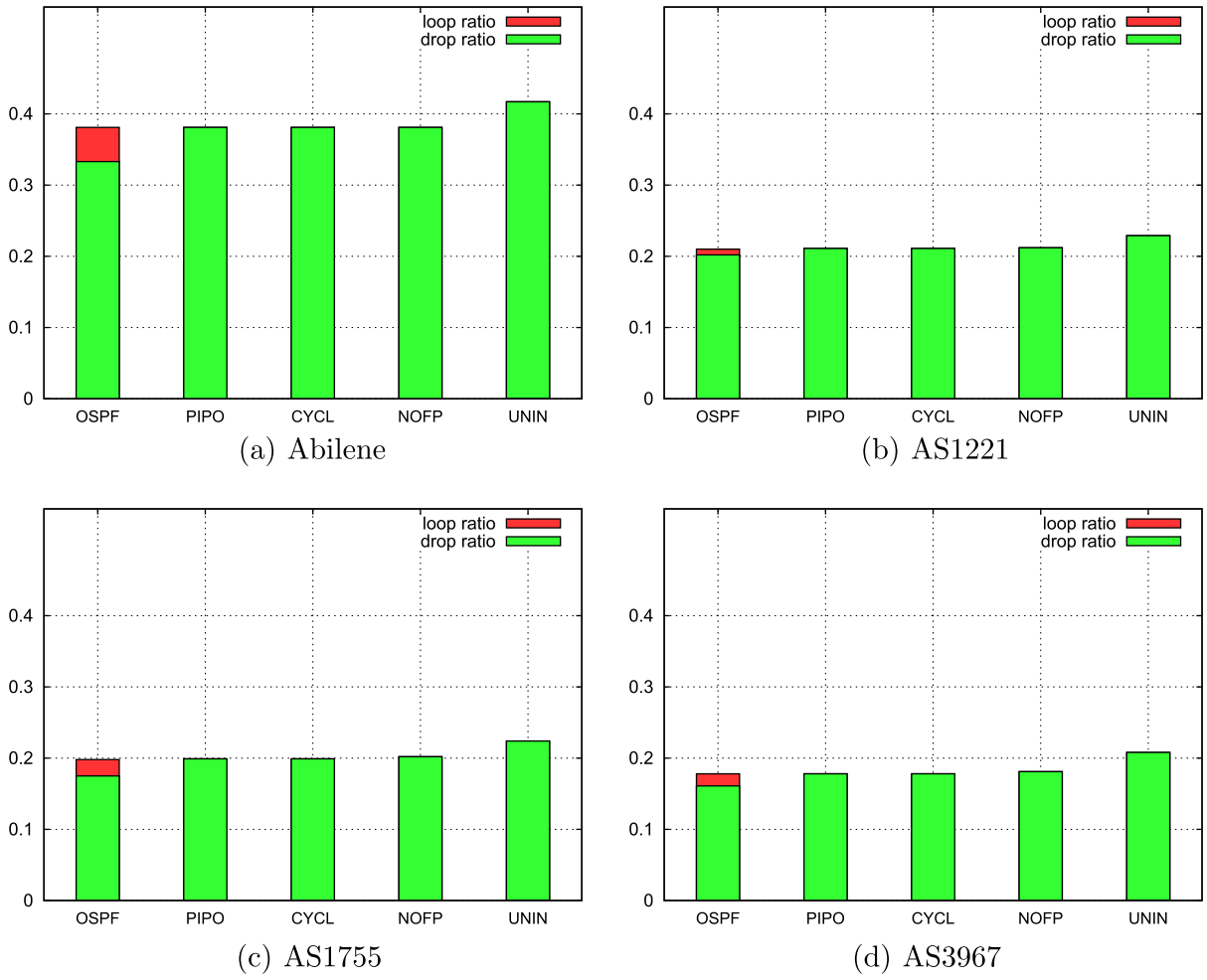


Fig. 9. Performance under 1-node failure.

6.3. Multiple failures in a Tier-1 ISP

To further evaluate LISF methods, we simulated failures of multiple links and nodes. For this study, we used the Tier-1 ISP topology as before but assigned different symmetric and asymmetric weights to induce forwarding loops. Furthermore, we did not use the control-plane simulator mentioned above as it can only handle single link failures. Instead, we used a simplified model to approximate link-state propagation and route computation and FIB update times. It is assumed that link-state propagation takes 1 time unit per hop and route computation and FIB update time is 3 units. We simulate single node failures and also simultaneous failures of 2 nodes, and also 2 links and 3 links. In each failure scenario, we forward a packet between every pair of nodes and count the number of node pairs for whom packets are undeliverable and also those that get caught in a loop.

Tables 11 and 12 show the relative performance of different LISF methods and OSPF in terms of their ability to avoid loops and deliver packets under symmetric and asymmetric weights, respectively. Note that PIPO and CYCL yield identical performance when the link weights are symmetric. There are several things to observe here. Compared to OSPF, loops are close to 1000 times less likely to happen with PIPO and CYCL. In terms of packet delivery, both PIPO and CYCL have the same performance as OSPF. UNIN prevents loops under all scenarios whereas NOFP has a very few loops and only under asymmetric weights with 3-link and 2-node failures. However, the delivery ratio of NOFP is only slightly worse than OSPF while that of UNIN is significantly worse. Considering that NOFP prevents almost all loops without excessive discarding of packets, we believe LISF approach with NOFP method is a viable alternative for avoiding transient loops during the convergence of intra-domain routing schemes in IP networks. On the

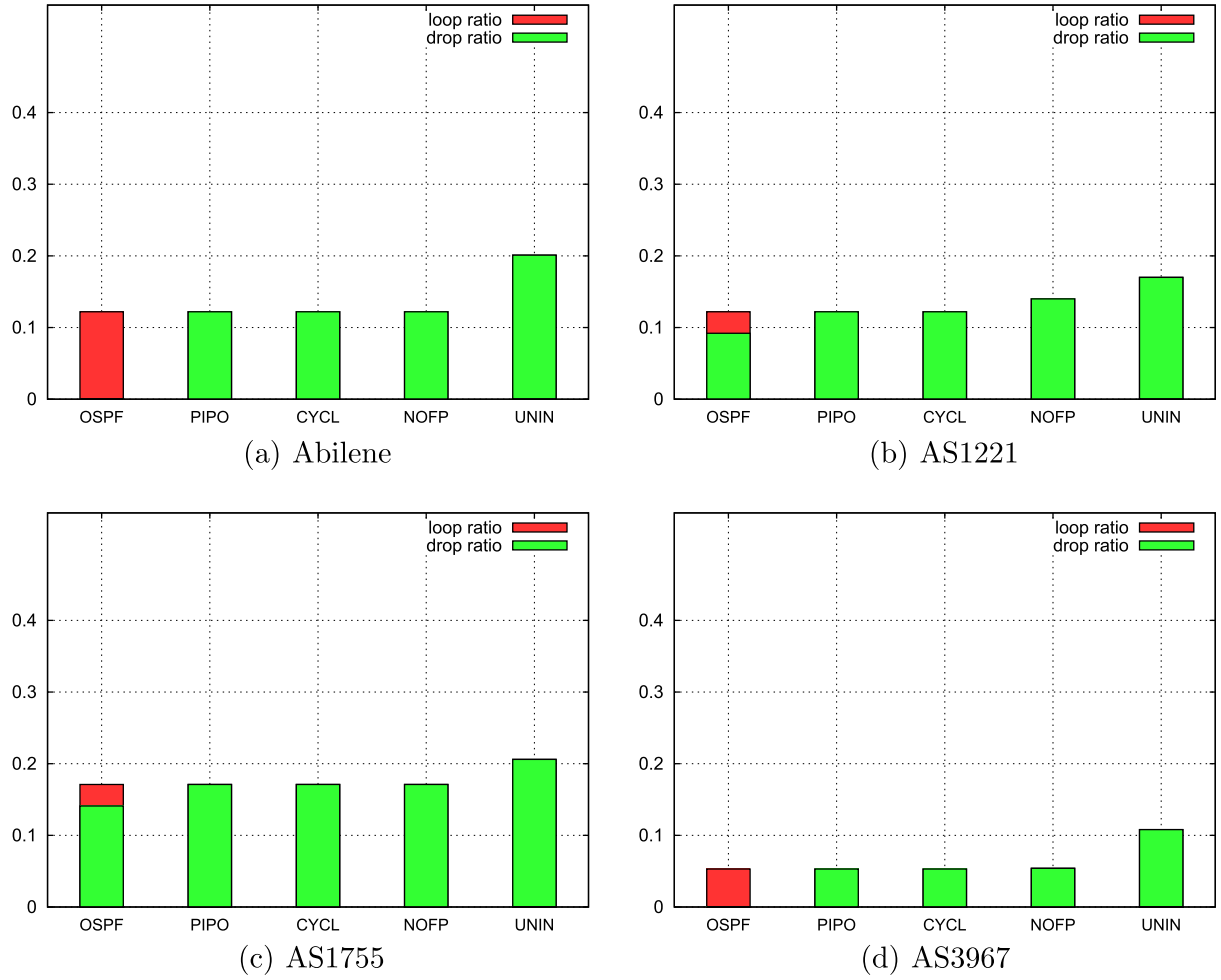


Fig. 10. Performance under 2-link failure.

Table 11
Comparison of OSPF and LISF under multiple failures and symmetric weights

Failures	Looping probability					% of undeliverable node pairs				
	OSPF	PIPO	CYCL	NOFP	UNIN	OSPF	PIPO	CYCL	NOFP	UNIN
2 links	$10^{-2.3}$	$10^{-5.3}$	$10^{-5.3}$	0	0	5.9	5.9	5.9	6.2	7.3
3 links	$10^{-2.1}$	$10^{-4.9}$	$10^{-4.9}$	0	0	8.6	8.6	8.6	9.1	10.7
1 node	$10^{-3.0}$	0	0	0	0	4.3	4.3	4.3	4.4	4.8
2 nodes	$10^{-2.7}$	$10^{-3.7}$	$10^{-3.7}$	0	0	8.1	8.1	8.1	8.2	9.1

Table 12
Comparison of OSPF and LISF under multiple failures and asymmetric weights

Failures	Looping probability					% of undeliverable node pairs				
	OSPF	PIPO	CYCL	NOFP	UNIN	OSPF	PIPO	CYCL	NOFP	UNIN
2 links	$10^{-2.4}$	$10^{-4.5}$	$10^{-4.8}$	0	0	6.0	6.0	6.0	6.4	7.8
3 links	$10^{-2.2}$	$10^{-4.0}$	$10^{-4.3}$	$10^{-6.4}$	0	8.7	8.7	8.7	9.3	11.3
1 node	$10^{-2.9}$	0	0	0	0	4.4	4.4	4.4	4.7	5.4
2 nodes	$10^{-2.5}$	$10^{-3.7}$	$10^{-4.4}$	$10^{-5.0}$	0	8.3	8.3	8.3	8.8	10.0

other hand, PIPO is easier to implement and is quite adequate in mitigating most of the loops since majority of the failures involve a single link or a single node.

7. Conclusions

In this paper, we proposed a simple interface-specific forwarding based approach called LISF to avoid transient forwarding loops during the network convergence periods. LISF approach selectively discards packets arriving through unusual interfaces when they are likely to be caught in a loop. We have demonstrated that LISF incurs no additional message overhead compared to OSPF and avoids forwarding loops without increasing the network convergence time. We have presented several LISF methods and evaluated their performance under various network topologies and failure scenarios. We observed that simple PIPO is effective in eliminating most of the loops particularly when weights of links are symmetric and NOFP provides the best trade-off between packet-discarding and loop-avoidance.

Acknowledgement

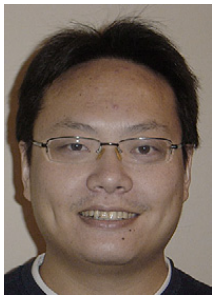
This work is partly supported by National Science Foundation CAREER Award CNS-0448272 and CRI Grant CNS-0551650. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] A. Markopulu, G. Iannaccone, S. Bhattacharya, C.-N. Chuah, C. Diot, Characterization of failures in an IP backbone, in: Proceedings of the IEEE Infocom, 2004.
[2] U. Hengartner, S.B. Moon, R. Mortier, C. Diot, Detection and analysis of routing loops in packet traces, in: IMW, Marseilles, France, 2002.
[3] S. Bryant, M. Shand, A framework for loop-free convergence, internet draft, in preparation. <draft-ietf-rtgwg-lf-conv-frmwk-01.txt>.
[4] A. Zinin, Analysis and minimization of microloops in link-state routing protocols, internet draft, in preparation. <draft-ietf-rtgwg-microloop-analysis-01.txt>.
[5] J. Garcia-Luna-Aceves, S. Murthy, A path-finding algorithm for loop-free routing, IEEE/ACM Trans. Networking 5 (1) (1997) 148–160.
[6] P. Francois, O. Bonaventure, Avoiding transient loops during IGP convergence in IP networks, in: IEEE Infocom, 2005.
[7] Routing Area Working Group, 2004. <http://psg.com/~zinin/ietf/rtgwg>.
[8] P. Francois, O. Bonaventure, M. Shand, S. Bryant, S. Previdi, Loop-free convergence using oFIB, internet draft, in preparation. <draft-ietf-rtgwg-ordered-fib-01.txt>.
[9] P. Francois, M. Shand, O. Bonaventure, Disruption-free topology reconfiguration in OSPF networks, in: IEEE Infocom, 2007.
[10] Z. Zhong, R. Keralapura, S. Nelakuditi, Y. Yu, J. Wang, C.N. Chuah, S. Lee, Avoiding transient loops through interface-specific forwarding, in: Proceeding of the International Workshop on Quality of Service (IWQoS), 2005.
[11] H. Chao, Next generation routers, Proc. IEEE 90 (9) (2002) 1518–1558.
[12] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, C.-N. Chuah, Fast local rerouting for handling transient link failures, IEEE/ACM Trans. Networking 15 (2) (2007) 359–372.
[13] S. Bryant, M. Shand, S. Previdi, IP fast reroute using Not-via addresses, Internet Draft, in preparation. draft-ietf-rtgwg-ipfrr-notvia-addresses-01.txt.
[14] A. Kvalbein, A. Hansen, T. Cicic, S. Gjessing, O. Lysne, Fast IP network recovery using multiple routing configurations, in: Proceedings of the IEEE Infocom, 2006.
[15] X. Yang, D. Wetherall, Source selectable path diversity via routing deflections, in: ACM SIGCOMM, 2006.
[16] R. Keralapura, C.N. Chuah, G. Iannaconne, S. Bhattacharrya, Service availability: a new approach to characterize IP backbone topologies, in: Proceedings of the International Workshop on Quality of Service (IWQoS), 2004.
[17] G. Iannaccone, C.-N. Chuah, S. Bhattacharyya, C. Diot, Feasibility of IP restoration in a tier-1 backbone, IEEE Network Mag., Special Issue on Protection, Restoration and Disaster Recovery, 2004.
[18] N. Spring, R. Mahajan, D. Wetherall, Measuring ISP topologies with Rocketfuel, in: ACM SIGCOMM, 2002.

Srihari Nelakuditi received his B.E. in Computer Science from Andhra University, Visakhapatnam, India, M.Tech. in Computer Science from Indian Institute of Technology, Madras, India, and Ph.D. in Computer Science from University of Minnesota, Minneapolis. He is currently an Assistant Professor at University of South Carolina, Columbia. He received the National Science Foundation CAREER Award in 2005. His main research interests are resilient routing and streaming in wired and wireless networks.

Zifei Zhong received his B.E. in Computer Science and Technology from Wuhan University, China, in 2004, and M.S. in Computer Science and Engineering from the University of South Carolina, Columbia, in 2006. He is currently a Ph.D. Student in Computer Sciences at the University of Texas, Austin. He received MCD Fellowship and Dean's Excellence Award from the University of Texas, Austin, in 2006. His research interests include distributed systems and networking.



Junling Wang received his Ph.D. degree in Computer Science and Engineering from University of South Carolina, Columbia in 2007. His research focus is on network routing resiliency. He is currently working as a financial software developer at Bloomberg LP.

Ram Keralapura is currently a Senior Member of Technical Staff at Narus Inc in Mountain View, CA. He received his BE in Electrical Engineering from Bangalore University, India in 1998, MS in Computer Science from the University of Alabama in Huntsville in 2000, and PhD in Electrical and Computer Engineering from University of California, Davis in 2007. His research interests include distributed network security, measurements, and monitoring. He is also interested in Internet routing, traffic engineering, and overlay/P2P networks.

Chen-Nee Chuah (S′92, M′01, SM′06) is currently an Associate Professor in the Electrical and Computer Engineering Department at the University of California, Davis. She received her B.S. in Electrical Engineering from Rutgers University, and her M.S. and Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley in 1997 and 2001, respectively. Before joining UC Davis, she spent 9 months as a visiting researcher at Sprint Advanced Technology Laboratories. Her research interests lie in the area of computer networking and distributed systems, ranging from Internet measurements, network management, overlay/peer-to-peer systems, network security, wireless/mobile networking, to opportunistic communications. She has published more than 70 research papers and received various awards and grants, including a NSF CAREER Award, an ACM Recognition of Service Award, and the UC Davis College of Engineering Outstanding Junior Faculty Award.