Avoiding Transient Loops Through Interface-Specific Forwarding

Zifei Zhong[1], Ram Keralapura[2], Srihari Nelakuditi[1], Yinzhe Yu[3], Junling Wang[1], Chen-Nee Chuah[2], and Sanghwan Lee[3]

[1] Department of Computer Science & Engineering, University of South Carolina, Columbia, SC 29208, USA
{zhongz, srihari, wang257}@cse.sc.edu
[2] Department of Electrical & Computer Engineering, University of California at Davis, Davis, CA 95616, USA
{rkeralap, chuah}@ece.ucdavis.edu
[3] Department of Computer Science & Engineering, University of Minnesota, Minneapolis, MN 55414, USA
{yyu, sanghwan}@cs.umn.edu

Abstract. Under link-state routing protocols such as OSPF and IS-IS, when there is a change in the topology, propagation of link-state announcements, 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 increased convergence delay. We propose an alternate approach, *loopless interface-specific forwarding* (LISF), that averts transient loops by forwarding a packet based on both its incoming interface and destination. LISF requires no modifications to the existing link-state routing mechanisms. It is easily deployable with current routers since they already maintain a FIB at each interface for lookup efficiency. This paper presents the LISF approach, proves its correctness, discusses three alternative implementations of it and evaluates their performance.

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

H. de Meer and N. Bhatti (Eds.): IWQoS 2005, LNCS 3552, pp. 219–232, 2005.
© IFIP International Federation for Information Processing 2005

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 detection and the FIB updates at all the routers, is referred to as the *convergence delay*. During the convergence 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. 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 [2,3,4,5,6][1] and an IETF working group has been addressing this issue [9]. Path locking with safe neighbors approach [2] 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 [3] 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. Recently, similar methods have been proposed in [4,5], 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. Although these schemes do avoid transient loops, they require additional messages to be exchanged among routers, to enforce the ordering of the updates of forwarding tables, resulting in increased convergence delay.

In this paper, we propose an alternate approach – *loopless interface-specific forwarding* (LISF) – that exploits the existence of one forwarding table per interface to avoid transient loops without requiring any changes to the existing link state routing mechanisms. 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 avoids transient loops without increasing the convergence delay

[1] Many other schemes have been proposed to deal with failures through fast local rerouting [7,5,8]. However, the focus of this paper is on schemes specifically designed for loop avoidance during convergence after a network-wide link state update.

and without employing any additional mechanisms to synchronize the forwarding table updates in different nodes.

The rest of this paper is structured as follows. In Section 2, we illustrate the problem of transient loops. Our LISF approach for avoiding forwarding loops and three possible implementations of it are described in Section 3. In Section 4, we prove that the proposed LISF methods prevent loops in case of symmetric single link failures. The results of our simulations evaluating the LISF methods are presented in Section 5. We finally conclude the paper in Section 6.

2 Transient Looping Problem and Existing Approaches

We now illustrate the occurrence of transient loops, discuss a recently proposed approach for avoiding them, and point out the need for an alternate approach.

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

Consider a scenario where link E−D fails at time 0s. 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 0.65s, 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 0.55s. During the interval between the forwarding table updates in nodes E and F (i.e, between 0.45s and 0.55s), 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 was proposed in [4] that

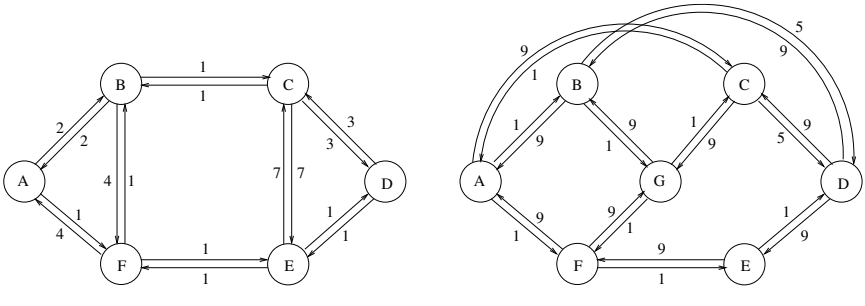


Fig. 1. Topologies used for illustration

Table 1. Summary of routing events under OSPF, ORDR, and LISF

	OSPF		LISF	ORDR
Time	Events	A to D pkts	A to D pkts	Events
0s	Failure of link E-D	A-F-E-drop	A-F-E-drop	Failure of link E-D
0.05s	D,E: failure detected	A-F-E-drop	A-F-E-drop	D,E: failure detected
0.15s	C,F: failure notified	A-F-E-drop	A-F-E-drop	C,F: failure notified
0.25s	A,B: failure notified	A-F-E-drop	A-F-E-drop	A,B: failure notified
0.35s	B: route update	A-F-E-drop	A-F-E-drop	B: route update
0.45s	D,E: route update	A-F-E-F-...(**loop**)	A-F-E-F-**drop**	
0.55s	C,F: route update	A-F-B-C-D	A-F-B-C-D	C: route update
0.65s	A: route update	A-B-C-D	A-B-C-D	A: route update
1.05s				D: route update
1.15s				F: route update
1.65s				E: route update

ensures ordered installation of forwarding table entries by exchanging messages between neighboring nodes. Here, we consider a similar approach (which we refer to as ORDR) for avoiding loops in case of unplanned failures (around 80% of all failures according to [1]) of unprotected links. The routing events under ORDR corresponding to Fig. 1(a) are shown in Table 1. Note that with this method, F updates its forwarding table 500 ms (assuming update time of 400 ms and the message propagation and processing time of 100 ms) after A updates its table. While this method avoids forwarding loops, its drawback is that it increases the network convergence delay. For the scenario discussed here, this method extends the convergence delay to 1.65s.

Our objective is to develop a scheme that combines the best features of OSPF and ORDR, i.e., low convergence delay and disruption time of OSPF and loop avoidance of ORDR. 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 0.45s to 0.55s). Consequently, the ideal scheme would have the convergence delay of 0.65s and service disruption time of 0.45s while also avoiding forwarding loops. In the following sections, we present and evaluate a scheme that closely approximates this ideal behavior.

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. In this section, we first briefly explain

interface-specific forwarding and argue how it can be exploited to avoid loops. We then present three 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. 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(a) 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 forwarding tables *interface-specific*. Table 3 gives the possible set of interface-specific forwarding table entries at node B of Fig. 1(a). Each entry is marked with '–', **X**, or a nexthop node. The entries marked '–' are obviously never used. The entries marked **X** are not referenced normally, i.e., when there is no failure and all nodes in the network have the same consistent view. For example, in Fig. 1(a), 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 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

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 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.

Before we proceed to present various LISF methods, we first introduce some notation that would help describe them. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the graph with vertices $\mathcal{V}$ and edges $\mathcal{E}$ representing the network. We use $\mathcal{R}_i^d$ to denote the next hop[2] from i to d in $\mathcal{G}$. Let $\mathcal{F}_{j \to i}^d$ denote the forwarding table entry, i.e., the next hop to d for packets arriving at i through the interface associated with neighbor j. We use $\mathcal{P}_i^d$ to refer to the shortest path from i to d given the graph $\mathcal{G}$. Similarly, the cost of the shortest path is denoted by $\mathcal{C}_i^d$.

We now present three different LISF methods. The difference between these methods 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 operation of these methods, when a packet for destination d arrives at node i from neighbor j, is summarized in Table 4 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.

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

PIPO discards a packet only when it arrives at a node from its next hop, i.e., along the 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 a loop. Otherwise, without PIPO, in such a scenario, packets will ping-pong between two neighboring nodes. For example, in Table 5, a packet to destination D arriving

[2] Note that LISF works even with Equal Cost Multipath (ECMP) routing. But for ease of explanation, it is assumed that there is only one shortest path per destination.

Table 4. Differences in LISF methods in discarding a packet to d arriving at i from j

method	discard condition	discard criterion
PIng-POng (PIPO)	$j \in \mathcal{R}_i^d$	in and out interfaces are same
CYCLe (CYCL)	$j \in \mathcal{P}_i^d$	previous node along the path
NO Forward Progress (NOFP)	$\mathcal{C}_{\mathcal{R}_i^d}^d \geq \mathcal{C}_j^d$	no forward progress

Table 5. 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**)

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.

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

CYCL discards a packet when it arrives from a node which falls along the shortest path from this 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 5, 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.

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., $\mathcal{F}_{j\to i}^d = \ominus$ if $\mathcal{C}_{\mathcal{R}_i^d}^d \geq \mathcal{C}_j^d$.

Table 6. Differences among LISF methods in discarding packets arriving at node B

from	to	failed link	PIPO	CYCL	NOFP
C	D	C−D	discard	discard	discard
F	E	F−E	forward to A	discard	discard
F	D	F−E	forward to C	forward to C	discard
C	E	C−D	forward to A	forward to A	forward to A

NOFP discards a packet if its previous hop is not farther from its destination than the next hop of this node. In such a case, there is a potential for a loop and NOFP discards such packets. For example, in Table 5, 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 3. 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 3 from next hop G to D is not smaller than the cost from A to D which is also 3. The downside however is that, a straightforward method to implement NOFP requires a computation of $O(\frac{|\mathcal{E}|}{|\mathcal{V}|})$ 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.

The difference between the actions of the above three methods is clearly evident in the presence of failures of links C−D and F−E in Fig. 1(a) as shown in Table 6. 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 whereas PIPO and CYCL forward them along a loop-free path via C. Essentially these methods achieve different tradeoffs between loop-avoidance and packet-discarding, which can be summed up as follows.

- packet looping probability: PIPO $\geq$ CYCL $\geq$ NOFP
- packet discard probability: PIPO $\leq$ CYCL $\leq$ NOFP
- network convergence delay: PIPO = CYCL = NOFP

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 state change is being propagated in a

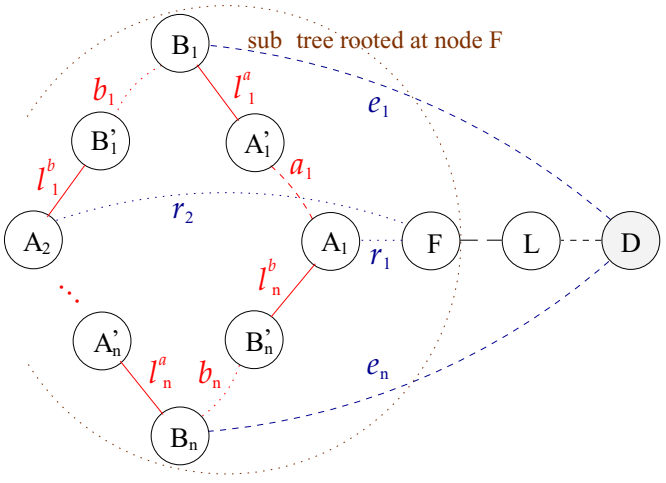


Fig. 2. Scenarios for illustrating loop-freedom under PIPO

network with symmetric links. It is clear that if PIPO is loop-free, other two 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.

We use the following notation in this section. Let $\mathcal{T}^D$ be the shortest path tree with undirected links rooted at a destination D. Note that this tree contains the shortest paths from every node to D since the links are symmetric. We use $S(i, \mathcal{T})$ to denote the subtree of $\mathcal{T}$ below the node i. Since the forwarding to a destination is independent of forwarding to other destinations, in the following we prove it for a destination D.

Theorem 1. *A packet destined for D will not loop under PIPO in case of the failure of a link ℓ.*

Proof. If $\ell \notin \mathcal{T}^D$, 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 $\ell \in \mathcal{T}^D$.

Let $\ell = \text{F}-\text{L}$ and F be the upstream node to L on the path from F to D as in Fig. 2. When ℓ is down, only those nodes in $S(F, \mathcal{T}^D)$ 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 $S(F, \mathcal{T}^D)$. That packet may be forwarded to node F and then get rerouted, or get rerouted somewhere in $S(F, \mathcal{T}^D)$. 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 $S(F, \mathcal{T}^D)$. Thus, to prove loop-freedom under PIPO, we only need to show that there will not be a loop within $S(F, \mathcal{T}^D)$.

Suppose there is a loop in subtree $S(F, \mathcal{T}^D)$. The loop must contain some nodes that are *aware* and some that are *unaware* of the failed link F−L. Pick

an arbitrary node, A_1, in the loop that is aware of the failure. As shown in Fig. 2, suppose the packet starts from A_1 and is routed along zero or more "A" nodes (i.e., nodes that are aware of the failure and forward in consistence as A_1), and reaches $A_1^{'}$, the last "A" node in this stretch. $A_1^{'}$ forwards the packet to B_1, a node unaware of the failure. Note that from A_1 to B_1, the forwarding is consistent (with $\mathcal{P}^D_{A_1}(\mathcal{E} \setminus \ell)$). We use the dashed line to indicate this subpath of $\mathcal{P}^D_{A_1}(\mathcal{E} \setminus \ell)$. B_1 then reroutes the packet: 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 rerouted 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 $B_n (n \geq 1)$.

Now we show that there is a contradiction if such a loop exists. Note that in such a loop, any B_i can not 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 reroute decision at node $B_i (1 \leq i \leq n)$. 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 rerouting 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, formula (6) above contradicts formula (3). Therefore, a forwarding loop is not possible under PIPO in case of a single link failure.

Using similar arguments as above, we can show that PIPO avoids forwarding loops during the convergence period after not only a failure but any change in the state of a link.

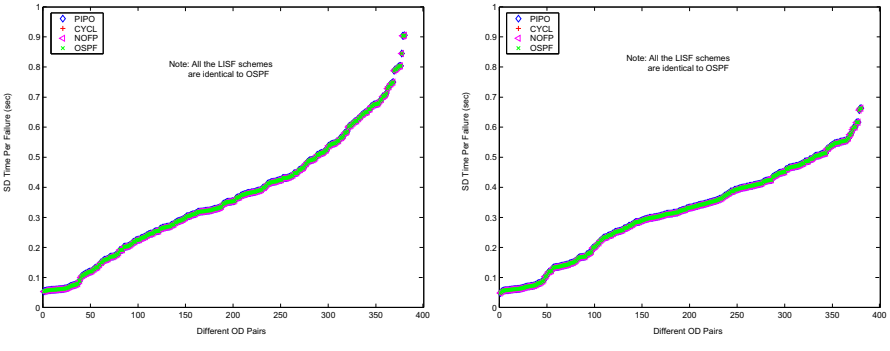
5 Performance Evaluation

In this section, we evaluate the performance of LISF methods and compare them against OSPF and ORDR. We first simulate single link failures and demonstrate that LISF methods prevent a significant number of loops that are possible under OSPF, and also have lower convergence delay than ORDR. We then experiment with scenarios of multiple failures to further study the tradeoffs of LISF methods between packet-discarding and loop-avoidance.

5.1 Single Link Failures

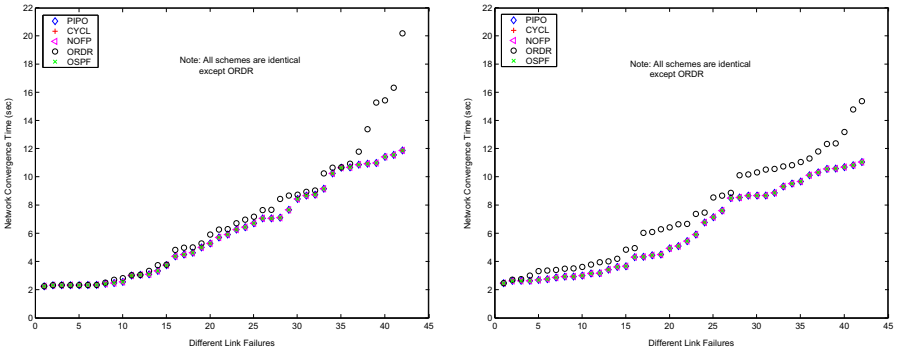
To evaluate LISF methods, we built a control plane simulator that emulates intra-domain routing dynamics in the presence of link failures and measures both service disruption (SD) time between different origin-destination (OD) pairs and network convergence (NC) time as presented in [10]. We use a Tier-1 ISP backbone (PoP-level) topology with 20 nodes and 42 links in our simulations which was used earlier in [10]. 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 161352. The other parameters in the simulator are set based on the findings in [11]. 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.

The total time during which a forwarding loop exists under OSPF is observed to be 11.851s, whereas LISF methods, as expected, have no loops in case of symmetric link failures. For the case of asymmetric link failures, OSPF has loops for a duration of 6.989s, while it is 0.056s for PIPO. In both cases, there are no loops under CYCL or NOFP. These results demonstrate the effectiveness



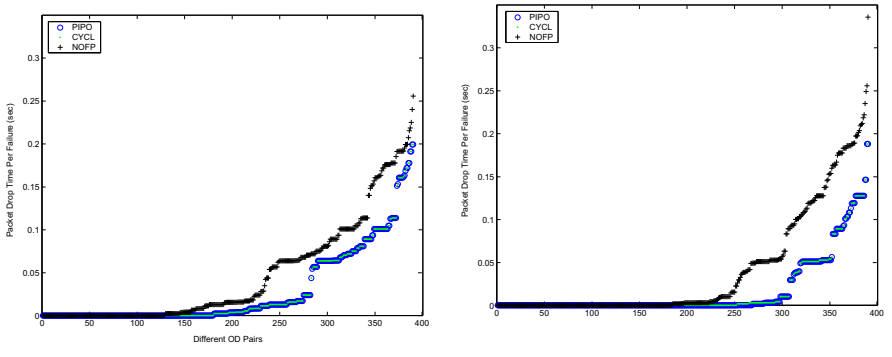
(a) symmetric links (b) asymmetric links

Fig. 3. Average service disruption time per link failure for various O-D pairs



(a) symmetric links (b) asymmetric links

Fig. 4. Network convergence time due to various single link failures



(a) symmetric links (b) asymmetric links

Fig. 5. Average packet discard time per link failure for various OD pairs

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.

Figure 3(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. Figure 3(b) shows a very similar behavior of average SD time for this network with asymmetric links.

Let NC time represent the total time taken for the network to converge after a link failure. Fig. 4(a) and Fig. 4(b) show the NC time due to different link failures for the backbone network with symmetric and asymmetric links

respectively. Given that some nodes in the network have to wait for a longer time to update their FIBs under ORDR, it is easy to see that a network with ORDR exhibits higher NC times when compared to a network with LISF where no synchronization is needed between nodes for updating their FIBs.

Fig. 5(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, and NOFP is quite close to PIPO. A similar pattern is observed even with asymmetric links (Fig. 5(b)). However, the packet discard times under PIPO and CYCL are not identical due to the fact that more loops are avoided by CYCL compared to PIPO.

5.2 Multiple Link Failures

To further evaluate LISF methods, we simulated failures of multiple links and nodes. For this study, 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.

Table 7. Comparison of LISF methods and OSPF in case of multiple failures

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

Table 7 shows the relative performance of different LISF methods and OSPF in terms of their ability to avoid loops and deliver packets. ORDR is not included here as it is not designed to deal with multiple failures. PIPO and CYCL yield identical performance since the links are symmetric. Compared to OSPF, loops are close to 1000 times less likely to happen with PIPO and CYCL, whereas no loops occur under NOFP. In terms of packet delivery, both PIPO and CYCL have the same performance as OSPF. The delivery ratio of NOFP is only slightly worse than OSPF. Considering that NOFP prevents 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.

6 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 like ORDR without increasing the network convergence time. We have presented several LISF methods and evaluated their performance. We observed that simple PIPO is effective in eliminating most of the loops and NOFP provides the best trade-off between packet-discarding and loop-avoidance.

References

1. Markopulu, A., Iannaccone, G., Bhattacharya, S., Chuah, C.N., Diot, C.: Characterization of failures in an IP backbone. In: Proc. IEEE Infocom. (2004)
2. Zinin, A.: Analysis and minimization of microloops in link-state routing protocols (2004) Internet draft, draft-zinin-microloop-analysis-00.txt, work in progress.
3. Garcia-Luna-Aceves, J., Murthy, S.: A path-finding algorithm for loop-free routing. IEEE/ACM Transactions on Networking **5** (1997)
4. Francois, P., Bonaventure, O.: Avoiding transient loops during IGP convergence in IP networks. In: IEEE Infocom. (2005)
5. Bryant, S., Filsfils, C., Previdi, S., Shand, M.: IP Fast Reroute using tunnels (2004) Internet draft, draft-bryant-ipfrr-tunnels-00.txt, work in progress.
6. Bryant, S., Zhang, M.: A framework for loop-free convergence (2004) Internet draft, draft-bryant-shand-lf-conv-frmwk-00.txt, work in progress.
7. Zhong, Z., Nelakuditi, S., Yu, Y., Lee, S., Wang, J., Chuah, C.N.: Failure Inferencing based Fast Rerouting for Handling Transient Link and Node Failures. In: Global Internet Symposium, Miami (2005)
8. Atlas, A.: U-turn alternates for IP/LDP fast-reroute (2005) Internet draft, draft-atlas-ip-local-protect-uturn-02, work in progress.
9. Routing Area Working Group: http://psg.com/ zinin/ietf/rtgwg (2004)
10. Keralapura, R., Chuah, C.N., Iannaconne, G., Bhattacharrya, S.: Service availability: A new approach to characterize IP backbone topologies. In: Proc. International Workshop on Quality of Service (IWQoS). (2004)
11. Iannaccone, G., Chuah, C.N., Bhattacharyya, S., Diot, C.: Feasibility of IP restoration in a tier-1 backbone. IEEE Network Magazine, Special Issue on Protection, Restoration and Disaster Recovery (2004)