Handling Multiple Failures in IP Networks through Localized On-demand Link State Routing

Glenn Robertson, Srihari Nelakuditi

***Abstract*—It has been observed that transient failures are fairly common in IP backbone networks and there have been several proposals based on *local rerouting* to provide high network availability despite failures. While most of these proposals are effective in handling single failures, they either cause loops or drop packets in the case of multiple independent failures. To ensure forwarding continuity even with multiple failures, we propose *Localized On-demand Link State* (LOLS) routing. Under LOLS, each packet carries a *blacklist*, which is a minimal set of failed links encountered along its path, and the next hop is determined by excluding the blacklisted links. We show that the blacklist can be reset when the packet makes *forward progress* towards the destination and hence can be encoded in a few bits. Furthermore, blacklist-based forwarding entries at a router can be *precomputed* for a given set of failures requiring protection. While the LOLS approach is generic, this paper describes how it can be applied to ensure forwarding to all reachable destinations in case of any two link or node failures. Our evaluation of this failure scenario based on various real network topologies reveals that LOLS needs 6 bits in the worst case to convey the blacklist information. We argue that this overhead is acceptable considering that LOLS routing deviates from the optimal path by a small stretch only while routing around failures.**

I. Introduction

The Internet is increasingly being used for mission-critical applications and it is expected to be always available. Unfortunately, service disruptions happen even in well-managed networks due to link and node failures. There have been some studies [1]–[3] on frequency, duration, and type of failures in an IP backbone network. [2] reported that failures are fairly common and most of them are transient: 46% last less than a minute and 86% last less than ten minutes. To support emerging time-sensitive applications in today's Internet, these networks need to survive failures with minimal service disruption. For example, a disruption time of longer than 50 ms is considered intolerable for mission-critical applications [4]. Therefore, providing uninterrupted service availability despite *transient* failures is a major challenge for service providers.

While a majority of the failures were observed to be single failures, one study [2] has found that approximately 30% of unplanned failures (which constitute 80% of all failures) involve multiple links, which is a significant fraction that needs to be addressed. Moreover, the extent of service disruption caused by multiple failures can be quite significant. Hence, it is important to devise schemes that protect the network against not only single failures but also *multiple independent failures*. Our work is motivated by this need, which is also the focus of some of the recently proposed routing schemes [5]–[7].

The commonly deployed link state routing protocols such as OSPF and ISIS are designed to route around failed links but they lack the resiliency needed to support high availability [1]. The remedies suggested in [8], [9] can achieve convergence in less than one second. However, bringing it down below the 50ms threshold runs the risk of introducing routing instability due to hot-potato routing, which can cause relatively small internal link-state changes to trigger a large churn of external routes [10]. MPLS [11] can handle transient failures effectively with its label stacking capability. However, we argue that it is not scalable to configure many backup label switched paths for protection against various combinations of multiple independent failures. In [12], authors attempt to make MPLS based recovery scalable to multiple failures, but assume that probable failure patterns based on past statistics on the network failures are known to the MPLS control plane.

There have been several fast reroute proposals for handling transient failures in IP networks by having the adjacent nodes perform local rerouting without notifying the whole network about a failure [13]–[17]. However, most of these schemes are designed to deal with single or correlated failures only. Recently, [7] proposed an approach to handle dual link, but only single node failures. On the other hand, failure carrying packets (FCP) [5] and packet recycle (PR) [6] try to forward packets to reachable destinations even in case of arbitrary number of failures. The drawbacks, however, are that FCP carries failure information in each packet all the way to the destination whereas PR forwards packets along long detours.

We propose a scalable *Localized On-demand Link State* (LOLS) routing [18] for protection against multiple failures. LOLS considers a link as *degraded*[1] if its current state (say "down") is worse than its *globally advertised* state (say "up"). Under LOLS, each packet carries a *blacklist* (a minimal set of degraded links encountered along its path), and the next hop is determined by excluding the blacklisted links. A packet's blacklist is initially empty and remains empty when there is no discrepancy between the current and the advertised states of links along its path. But when a packet arrives at a node with a degraded link adjacent to its next hop, that link is added to the packet's blacklist. The packet is then forwarded to an alternate next hop. The packet's blacklist is reset to empty when the next hop makes *forward progress*, i.e., the next hop has a shorter path to the destination than any of the nodes traversed by the packet. With these simple steps, LOLS propagates the state of

Manuscript received August 08, 2011; revised Feb 19, 2012; accepted May 15, 2012 by the Associate Editor Marcus Brunner.

This work was supported in part by the National Science Foundation (NSF) under grants CNS-0448272 and CNS-0551650.

G. Robertson and S. Nelakuditi are with the Department of Computer Science and Engineering, University of South Carolina, Columbia, SC 29208.

[1]A down link is not considered degraded if it was advertised as down.

degraded links only when needed, and as far as necessary, and ensures loop-free delivery to all reachable destinations.

LOLS has several attractive features: 1) When there are no degraded links, forwarding under LOLS is identical to shortest path forwarding; 2) Even with degraded links, LOLS paths deviate from the optimal only by a small stretch; 3) LOLS forwarding entries can be precomputed for a given scenario of failures requiring protection; 4) Due to localized propagation of a packet's blacklist, it can be conveyed in just a few bits. With these features, LOLS compares favorably against FCP and PR. In short, unlike FCP, LOLS propagates failure information only locally. Compared to PR, forwarding paths are much shorter with LOLS. We provide a detailed contrast of LOLS with these and other related works in the next section.

The rest of the paper is organized as follows. Section III presents the LOLS approach for handling arbitrary number of failures. Section IV describes a practical implementation of LOLS for protection against a predefined set of failures. Section V reports the results of the performance evaluation. We discuss the issues related to the deployment of LOLS in Section VII. Finally, we conclude the paper in Section VIII.

II. Related Work

Numerous approaches have been proposed in the past to make networks more resilient to failures [17]. We categorize them into schemes that protect against single or correlated failures and those that can deal with multiple independent failures. Also, some of them attempt to reduce the routing overhead by controlling the range/frequency of link state updates. Another group of schemes construct a set of diverse paths for forwarding, even across domains. We briefly describe a few schemes belonging to each of these categories in the following. Finally, we elaborate on the similarities between forwarding around failures by LOLS, and forwarding around voids by geographic routing schemes like GPSR [19].

Single or Correlated Failures: The Not-via approach [20] locally reroutes a packet around a known failure by encapsulating the packet to an address that implicitly identifies the failed network component to be avoided. Another scheme, known as Multiple Routing Configurations (MRC), proposed in [14] separates all failures into multiple routing configurations and lets the packet carry the configuration information upon detecting a failure, so that the downstream routers can select the path consistently. Failure Inferencing based Fast Rerouting (FIFR) infers failures based on a packet's flight (the inbound interface on which they are received), precomputes interface-specific forwarding tables, and triggers local rerouting upon detecting an adjacent failure. The above and other similar schemes [21]–[26] offer resilience against single or correlated failures but are not designed to recover from multiple unrelated failures. They do, however, help make LOLS practical — blacklist size can be reduced with interface-face specific forwarding, and not-via addresses can be used to encode blacklist information.

Multiple Independent Failures: Convergence-free routing using Failure Carrying Packets (FCP) [5] can recover from an arbitrary number of failures. FCP, which was proposed after LOLS, is most relevant to our work. FCP also carries information about the failed links in the data packet and intermediate routers exclude those links while computing the next hop. But, unlike our scheme, the failure information under FCP is carried all the way to the destination, which is undesirable. Packet Re-cycling (PR) [6] is a technique that also aims to reduce the number of bits needed to be carried in a packet header to ensure successful rerouting. PR takes advantage of cellular graph embeddings to reroute packets that would otherwise be dropped in case of failures. It needs only in the order of $\log_2(d)$ bits in the header to cover all non-disconnecting failure combinations, where d is the diameter of the network. While the low header overhead is remarkable, packets under PR take longer detours than LOLS. Similarly, the backup paths under 2DMRC, an extension of MRC for recovery against two link failures, have higher costs than those under our scheme. This is because backup configurations under 2DMRC are insensitive to link costs and have global scope. Another recently proposed scheme [7] handles dual link or single node failures, whereas LOLS approach can handle two or more independent link or node failures.

Localized Link State Updates: To make link state routing scalable for mobile ad hoc networks, limited dissemination-based schemes have been proposed [27]. Fisheye state routing (FSR) [28] and hazy sighted link state (HSLS) [27] routing schemes update the nearby nodes at a higher frequency than the remote nodes that lie outside a certain scope. The drawback is that the chosen scope can be more than sufficient in some cases and less than necessary in other cases resulting in needless updates or forwarding loops. LOLS can be considered a form of limited dissemination based routing scheme that ensures loop-free forwarding while notifying only a small subset of nodes in the vicinity of a failure.

Forwarding over Diverse Set of Paths: RON [29] is a pioneering approach for recovering from path outages using a resilient overlay network. RON is primarily an attempt to overcome the slow convergence of BGP. A set of routing deflection rules are developed in [30] that enable routers to independently deflect packets and thereby collectively construct a diverse set of paths. This approach amounts to a form of implicit source routing in which end-systems set a four byte tag in the packet header to select non-shortest path routes. Path splicing, which is proposed in [31], perturbs link weights to produce multiple routing trees and allows traffic to switch trees at any hop en route to the destination While the path splicing can sustain connectivity in the face of multiple link and node failures, there is a small probability of forwarding loops and the number of splicing header bits needed is proportional to the number of hops. In contrast, LOLS is an intra-domain approach that is loop-free with less than one byte of overhead per packet and it deviates from shortest paths only near a failure.

Geographic Position based Routing: Greedy Perimeter Stateless Routing (GPSR) and similar schemes [19], [32] based on geographical positions forward in *greedy* mode, where each hop takes the packet closer (in terms of Euclidean distance) to

the destination. However, when the packet reaches a *deadend*, i.e., when the forwarding node is closer to the destination than any of its adjacent nodes, the forwarding is switched to *face* mode. In face mode, a packet is forwarded along the boundaries of a planarized subgraph. A packet is forwarded back in greedy mode when it reaches a node closer to the destination that the node at which it entered the face mode. While position based routing is highly scalable, it yields suboptimal paths compared to topology-based routing. We will see in the following section that LOLS is similar in spirit to position-based routing and can be thought of as switching between greedy and recovery modes during a packet's flight.

Protection Routing with Traffic Engineering: Various traffic engineering approaches have been proposed in the past to distribute load across a network [33]. One of the recent proposals attempts to assign link weights such that the load remains balanced even after a failure [34], but it does not ensure forwarding continuity during convergence. Towards providing protection and balancing traffic, a version of MRC [35] has been suggested that optimizes link weights for the backup configurations based on the observed traffic patterns. However, this method does not scale well for all possible link failures, and does not address multiple failure scenarios. [36] proposes an efficient algorithm for computing backup paths where spare capacity is shared between different backup paths but it focuses only on single link or node failures. Recently, a centralized routing approach [26] has been proposed that incorporates both protection routing and traffic engineering. While this approach has a great feature of trading off protection and performance, it can not guarantee 100% coverage even for single failures. Our focus is on ensuring guaranteed forwarding to all reachable destinations despite multiple failures, which in itself is a challenging problem even without traffic engineering. However, we show later in Section V that the potential overloading caused by local rerouting with LOLS is not that significant.

III. Localized On-demand Link State Routing

In this section, we start with an illustration to present the intuition behind the localized on-demand link state routing approach. We then formally describe the forwarding operation at each router along the path traversed by the packet.

A. Intuition and Illustration

Consider the topology shown in Fig. 1. Suppose each link's state is *advertised* to be *up* with the labelled cost, but *currently* the dashed links are *down* with cost ∞. Further assume that the nodes adjacent to a dashed link are aware of its current failed state (we use state and cost interchangeably as the cost of a link reflects its state). We refer to the dashed links whose current cost is worse than their advertised cost as *degraded*. If a link's current cost is worse than advertised, local rerouting around that link can cause loops. On the other hand, if a link's current state is better than advertised, forwarding over that link is loop-free. Hence, we focus only on degraded links.

Now, assume that A is the source and H is the destination for a packet. The usual shortest path from A to H is via C. But since A−C link is currently degraded, we have to find an alternate next hop. We need to choose a next hop such that the packet does not get caught in a forwarding loop. A way to guarantee loop-freedom is to employ *greedy forwarding* [16] that forwards the packet along a path with *decreasing cost* to the destination, i.e., each hop makes *forward progress* towards the destination. It is important that the path cost is determined consistently at all nodes based on the advertised topology.

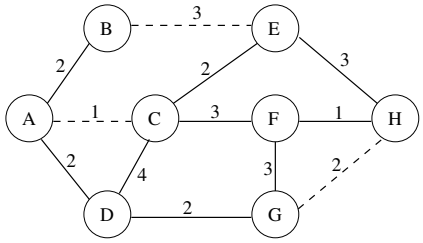


Fig. 1. Topology used for illustration. Degraded links are shown as dashed.

For example, in the topology of Fig. 1, the cost to reach H from A, B, D is 5, 6, and 4 respectively. Therefore, when A−C is degraded, the only feasible next hop from A is D, as per greedy forwarding. Again at G, since G−H is degraded, packets to H are deflected to F. Thus, a packet from A is delivered to H successfully even though all nodes do not have the accurate view of the network. However, greedy forwarding is not always feasible even if there exists a path. A packet may arrive at a *deadend* node whose cost to the destination is smaller than any of the possible next hops. For example, in Fig. 1, a packet from A destined to C is dropped when A−C is degraded. Note that a packet encounters a deadend only when there is a degraded link adjacent to the deadend node.

When there is no discrepancy between the advertised and current states of a link, none of its adjacent nodes can be a deadend if there exists a path to the destination. For example, if A−C is advertised to be down, the cost to reach C from A and D would be 6 and 4 respectively. Hence, D becomes a feasible next hop for A to reach C. But triggering a network-wide update upon every link state change causes significant overhead. It would be ideal to inform only those nodes in the neighborhood of the degraded link that would be affected by the degradation. But it is hard to determine the right scope for an update in the presence of multiple simultaneous or overlapping failures in the network. As an alternative, we propose *localized on-demand link state* (LOLS) routing which includes degraded links that cause a deadend in the packet itself so that forwarding by intermediate routers is loop-free.

Under LOLS, a packet can be thought of as being forwarded in two modes: *greedy* and *recovery*. A packet is normally forwarded in greedy mode to a next hop along the path with decreasing cost (w.r.t. the advertised topology) to the destination. When a packet hits a deadend in greedy mode, instead of discarding the packet, it is forwarded in recovery mode. In recovery mode, packets carry a *blacklist*, which is a set of degraded links encountered along the path. A

packet's next hop is chosen along a route that does not include blacklisted links. The forwarding of a packet is switched back to greedy mode, i.e., *the blacklist is reset to empty*, when it arrives at a node with lower cost (w.r.t. the advertised topology) to the destination than the node at which it entered the recovery mode. Thus, LOLS effectively propagates link state on demand, and only to as many nodes as necessary. This approach ensures loop-free forwarding to reachable destinations, even in the presence of many degraded links.

Note that there is no need for an explicit forwarding mode in LOLS. The greedy and recovery modes are used for ease of explanation only. Instead, a packet has a blacklist field in the header (which is nonempty in recovery mode) and its next hop is determined based on both its destination and blacklist. Consider again the example scenario of Fig. 1, where a packet is being forwarded from its source A to destination C. We have seen that under greedy forwarding, A would not be able to find a feasible next hop, and therefore, drops the packet. With LOLS, instead of dropping the packet, A includes the link A→C in the packet's blacklist, and forwards it to the alternate next hop D. Node D then computes the next hop without the blacklisted link A→C and finds that the next hop is C itself. Since the cost to the destination C from the next hop C is 0, and therefore smaller than that from A (which made blacklist nonempty) the blacklist is reset to $\emptyset$. The packet thus arrives at C along the alternate path A→D→C.

Now consider another scenario where B is the source and E is the destination for a packet. Since B→E is currently degraded, B would include B→E in the packet's blacklist and forward to A. Similarly, A would add A→C to the blacklist and forward to D. The node D determines C as the next hop based on the packet's blacklist. Before forwarding, it resets the blacklist to $\emptyset$ since the cost of 2 from C to E is less than that from B which is 3. Thus, the path taken by the packet would be B→A→D→C→E. The corresponding blacklist values at each of these hops would be {B→E}, {A→C, B→E}, $\emptyset$ and $\emptyset$ respectively. This illustrates how the blacklist of a packet *grows when necessary, and shrinks if possible* during the flight to its destination so as to ensure loop-free delivery.

These examples show how LOLS delivers a packet to its destination if there exists a path. We provide a proof of this in Appendix A. Moreover, only a few nodes near a degraded link must be notified of the link's state. In our example topology, a packet to any destination is delivered by LOLS *without* nodes G, F and H being informed of the degradation of A−C or B−E, and without informing nodes A, B, C, D and E of the degradation of G−H. Such an *on-demand propagation of link state* makes LOLS a scalable scheme for reliable forwarding.

B. Blacklist and Next-Hop Computation

In the following, we first describe the greedy forwarding procedure and then build upon it to develop the blacklist-based forwarding algorithm. We also show that blacklist-based forwarding can be performed by a simple table lookup based on both destination and blacklist fields of a packet.

Before we present the algorithms, we introduce some notation used in this paper, which is listed in Table I. We denote by $\widetilde{\mathcal{E}}$, the set of all links in the network and by $\tilde{c}_{i\mapsto j}$, the cost of a link $i\mapsto j$ according to the most recent link state advertisement. The set of links that are adjacent to i and currently in the degraded state are denoted by $\widetilde{\mathcal{B}}_i$. We use $\mathcal{P}_{i\rightsquigarrow d}(\mathcal{E})$ to refer to the shortest path from i to d with links in $\mathcal{E}$ and the corresponding cost is denoted by $\mathcal{C}_{i\rightsquigarrow d}(\mathcal{E})$.

TABLE I
NOTATION

$\widetilde{\mathcal{V}}$	set of all nodes in the network
$\widetilde{\mathcal{E}}$	set of all links in the network
$\tilde{c}_{i\mapsto j}$	cost of link $i\mapsto j$ as per the last advertisement
$\widetilde{\mathcal{B}}_i$	set of degraded adjacent links known to i
$\mathcal{P}_{i\rightsquigarrow d}(\mathcal{E})$	shortest path from i to d w.r.t. link set $\mathcal{E}$
$\mathcal{C}_{i\rightsquigarrow d}(\mathcal{E})$	cost of the shortest path from i to d w.r.t. $\mathcal{E}$

1) Greedy Forwarding: The procedure Greedy for selecting a next hop from node i to destination d, given the set of all links $\mathcal{E}$ and the set of degraded links $\mathcal{B}$, is shown in Alg 1. A neighbor j is considered a feasible next hop if the cost of the shortest path from j to d is smaller than that from i to d (line 4). Greedy returns $\emptyset$ when there is no such feasible next hop. If more than one feasible candidate exists, it picks the neighbor j^* by which i has the shortest path to d (lines 5-8). We need to point out that Greedy is a variant of classic greedy forwarding as it does not always choose a next hop with maximum forward progress. Instead, Greedy chooses a next hop such that it amounts to shortest path forwarding when there are no degraded links, which is certainly a desirable feature. Under this version of greedy forwarding, a node i forwards a packet p destined for p.dest to next hop k, where $k = \text{Greedy}(i, p.\text{dest}, \widetilde{\mathcal{E}}, \widetilde{\mathcal{B}}_i)$. It is important to note here that node i is aware of only its adjacent degraded links $\widetilde{\mathcal{B}}_i$. When there is no feasible next hop, i.e., if $k = \emptyset$, the packet is discarded. It is easy to show that forwarding using Greedy is loop-free, since forward progress is consistently ascertained at each hop w.r.t. the same set of links $\widetilde{\mathcal{E}}$ [16].

Algorithm 1 : Greedy Forwarding : Greedy$(i, d, \mathcal{E}, \mathcal{B})$

1: $j^* \Leftarrow \emptyset$
2: $h^* \Leftarrow \infty$
3: **for all** $j \in$ neighbors$(i, \mathcal{E} \setminus \mathcal{B})$ **do**
4: **if** $\mathcal{C}_{j\rightsquigarrow d}(\mathcal{E}) < \mathcal{C}_{i\rightsquigarrow d}(\mathcal{E})$ **then**
5: $h \Leftarrow \tilde{c}_{i\mapsto j} + \mathcal{C}_{j\rightsquigarrow d}(\mathcal{E})$
6: **if** $h < h^*$ **then**
7: $j^* \Leftarrow j$
8: $h^* \Leftarrow h$
9: **return** j^*

2) Blacklist-based Forwarding: Under LOLS, each packet p carries a blacklist p.blist in its header, and it is forwarded based on both p.dest and p.blist. The blist field of p is initialized to $\emptyset$ at the source and revised as needed during its forwarding. We show the LOLS forwarding procedure in Alg 2. In LOLS, we first look for a next hop with the smallest path cost and forward progress without the links in the packet's blacklist (line 1). If no such next hop is found, at least one

adjacent link of node i must be in a degraded state. Hence, we update the packet's blacklist by adding the degraded link(s) to the blacklist. The degraded links adjacent to i that need to be blacklisted are identified as follows. First, we find the neighbor with the smallest path cost using only the links in $\widetilde{\mathcal{E}} \setminus$ p.blist (line 3). If the link to that neighbor is currently degraded, then it is added to the blacklist (line 4-5). This process is repeated until either i) we find a next hop, the link from node i to which is not degraded, or ii) there is no such next hop (lines 4-6). In the latter case (j=$\emptyset$), the destination is *unreachable* and the packet is dropped. Otherwise, it is forwarded to j. Before forwarding, p.blist is reset to $\emptyset$ (lines 8-10), if j has shorter path cost to p.dest than the forwardmost deadend node (defined as the head node of the first link added to the p.blist). Thus, only during the recovery (until forward progress can be made), a packet is forwarded with the aid of a non-empty blacklist.

Algorithm 2 : Blacklist based Forwarding : LOLS(i, p)

1: $j \Leftarrow$ Greedy$(i, p.\text{dest}, \widetilde{\mathcal{E}} \setminus p.\text{blist}, \widetilde{\mathcal{B}}_i)$
2: **if** $j = \emptyset$ **then**
3: $\quad j \Leftarrow$ Greedy$(i, p.\text{dest}, \widetilde{\mathcal{E}} \setminus p.\text{blist}, \emptyset)$
4: $\quad$ **while** $j \neq \emptyset$ & $i{\rightarrow}j \in \widetilde{\mathcal{B}}_i$ **do**
5: $\quad\quad p.\text{blist} \Leftarrow p.\text{blist} \cup \{i{\rightarrow}j\}$
6: $\quad\quad j \Leftarrow$ Greedy$(i, p.\text{dest}, \widetilde{\mathcal{E}} \setminus p.\text{blist}, \emptyset)$
7: **if** $j \neq \emptyset$ and $p.\text{blist} \neq \emptyset$ **then**
8: $\quad k \Leftarrow$ Head(First(p.blist))
9: $\quad$ **if** $\mathcal{C}_{j \rightsquigarrow p.dest}(\widetilde{\mathcal{E}}) < \mathcal{C}_{k \rightsquigarrow p.dest}(\widetilde{\mathcal{E}})$ **then**
10: $\quad\quad p.\text{blist} \Leftarrow \emptyset$
11: **return** j

The rules for updating the blacklist of a packet p, p.blist, at node i can be summarized as follows:

- link $i{\rightarrow}j$ is *added* to p.blist if
  - $i{\rightarrow}j$ is currently in degraded state ($i{\rightarrow}j \in \widetilde{\mathcal{B}}_i$)
  - had $i{\rightarrow}j$ not been degraded, j would be the next hop ($j \in$ Greedy$(i, p.\text{dest}, \widetilde{\mathcal{E}} \setminus p.\text{blist}, \emptyset)$)
  - no feasible next hop exists without $i{\rightarrow}j$ (Greedy$(i, p.\text{dest}, \widetilde{\mathcal{E}} \setminus p.\text{blist}, \{i{\rightarrow}j\}) = \emptyset$)
- p.blist is *reset* to $\emptyset$ if
  - there exists a feasible next hop j ($j \in$ Greedy$(i, p.\text{dest}, \widetilde{\mathcal{E}} \setminus p.\text{blist}, \widetilde{\mathcal{B}}_i)$)
  - the cost from j to p.dest is smaller than that from any other node visited so far by p ($\mathcal{C}_{j \rightsquigarrow p.dest}(\widetilde{\mathcal{E}}) < \mathcal{C}_{k \rightsquigarrow p.dest}(\widetilde{\mathcal{E}})$ where k is Head(First(p.blist)))

Thus, blacklist-aided recovery ends and greedy forwarding resumes when the packet arrives at a node which makes forward progress. Such resetting of a packet's blacklist keeps it as minimal as possible and yet ensures loop-free forwarding.

C. Blacklist based Lookup Table for Forwarding

The description of blacklist-based forwarding thus far focused on the functionality in terms of how a next hop is selected for a packet, and how its blacklist is updated along the path to destination. We now show that blacklist-based forwarding boils down to a simple table lookup using both the p.dest and p.blist fields of a packet. Consider the operations performed by a node i while forwarding a packet p. It has to select a next hop j after excluding the links in p.blist and also update the p.blist. The p.blist may get *reset* to $\emptyset$, remain *unchanged*, or *grow* with the addition of adjacent links of i that are currently degraded. In all these cases, the *forwarding operation amounts to mapping p.dest and p.blist to a next hop j and a new p.blist based on the degraded links adjacent to i ($\widetilde{\mathcal{B}}_i$) and the last advertised state of all links $\widetilde{\mathcal{E}}$*. This mapping from $\langle p.\text{dest},p.\text{blist}\rangle$ to $\langle j,p.\text{blist}\rangle$ can be computed on-demand and remembered when node i first encounters this $\langle p.\text{dest},p.\text{blist}\rangle$ pair. Thereafter, any packet with $\langle p.\text{dest},p.\text{blist}\rangle$ combination can be forwarded simply by a table look up. This mapping has to be recomputed only when $\widetilde{\mathcal{B}}_i$ changes which is local, and when $\widetilde{\mathcal{E}}$ changes which is infrequent.

The blacklist based forwarding tables at nodes A, D, and G of the topology in Fig. 1 are shown in Table II. When a packet p arrives at a node, its forwarding table is looked up with tuple $\langle p.\text{dest},p.\text{blist}\rangle$ to determine the outgoing p.blist and the next-hop. For instance, when a packet destined for E with blacklist $\{\text{B}{\rightarrow}\text{E}\}$ arrives at node A, the packet's blacklist is changed to $\{\text{A}{\rightarrow}\text{C}, \text{B}{\rightarrow}\text{E}\}$ and forwarded to D. When this packet arrives at D with blacklist $\{\text{A}{\rightarrow}\text{C}, \text{B}{\rightarrow}\text{E}\}$, according to D's table the packet's blacklist is reset to $\emptyset$ and forwarded to C. We now describe how a node obtains its forwarding table.

The entries associated with blacklist of $\emptyset$ are computed in advance and the other entries are installed on-demand. Let us look at the forwarding table for node A. The usual next hop to reach C, F, and H is C. But since the link A$\rightarrow$C is degraded, A has to find alternate next hops. For H, it selects D as the next hop for H without having to blacklist A$\rightarrow$C. On the other hand, for reaching C, there is no feasible next hop with an empty blacklist. So it has to add A$\rightarrow$C to the blacklist and choose D as the next hop with blist set to A$\rightarrow$C. Similarly for F, we must add A$\rightarrow$C to the blacklist as well. Packets for destination E may arrive at A with three possible blacklists: $\emptyset$, B$\rightarrow$E, or $\{\text{A}{\rightarrow}\text{C}, \text{B}{\rightarrow}\text{E}\}$. In the first case, the next hop would be B with blist B$\rightarrow$E, and in the latter two cases next hop would be D with blist $\{\text{A}{\rightarrow}\text{C}, \text{B}{\rightarrow}\text{E}\}$.

Now consider the forwarding table at node D. Since it does not have any adjacent degraded links, the blacklist of any packet would not grow at node D but actually shrink. For example, the cost to F from node A where the link A$\rightarrow$C was blacklisted is 4, while it is only 3 from the D's next hop G. So D resets the blacklist to $\emptyset$. In the case of G, which is not an uncommon case, no additional entries are added since G would not receive a packet with a non-empty blacklist even though there are currently three degraded links in the network. Thus, for reliable delivery, LOLS needs only a *few* additional entries in the forwarding tables of nodes around failed links.

IV. Practical Version of LOLS: Protection against Predefined Set of Failures

The LOLS approach as described above can deal with an arbitrary number of failures. However, it requires a node to

TABLE II
BLACKLIST BASED FORWARDING TABLES AT NODES A, D AND G.

destination	blacklist: ∅ next	∅ blist	B→E next	B→E blist	A→C, B→E next	A→C, B→E blist
B	B	∅				
C	D	A→C				
D	D	∅				
E	B	A→C	D	A→C, B→E	D	A→C,B→E
F	D	A→C				
G	D	∅				
H	D	∅				

(A)

destination	blacklist: ∅ next	∅ blist	A→C next	A→C blist	A→C,B→E next	A→C,B→E blist
A	A	∅				
B	A	∅				
C	A	∅	C	∅		
E	A	∅			C	∅
F	G	∅	G	∅		
G	G	∅				
H	G	∅				

(D)

destination	blacklist: ∅ next	∅ blist
A	D	∅
B	D	∅
C	D	∅
D	D	∅
E	D	∅
F	F	∅
H	F	∅

(G)

dynamically compute a next hop for a packet whenever it carries a new blacklist. Therefore, we develop a practical version of LOLS that prepares for a number k (say 2) link or node failures, precomputes the necessary blacklist-based forwarding entries, and forwards each data packet with a simple table lookup. In the following, we first describe how a node can gather different blacklists carried in packets to the same destination. We then show how the blacklist field in the packet header can be represented using just a few bits.

A. Collection of Blacklists at a Router

We have illustrated earlier that for the failure scenario given in Fig. 1, a packet destined to E may arrive at D with three possible blacklists: $\emptyset$, B→E, or {A→C, B→E}, whereas the blacklist in packets to all other destinations arriving at D is always $\emptyset$. Given all such potential blacklists per destination at each node, the necessary forwarding entries can be computed as in Table II. Then, any packet can be forwarded around that scenario of failures without any next-hop computation "on the fly". Therefore, the first step in preparing for a set of failures is to infer the potential blacklists per destination at each node.

We collect this information by simulating the forwarding of a *virtual* packet between every source and destination pair for each failure scenario, according to the blacklist based forwarding in Algorithm 2. For instance, when preparing for any one or two link/node failures, the possible failure scenarios that need to be simulated are: one link, one node, a pair of links, a pair of nodes, or one link and one node. Note that a virtual packet is not actually forwarded to the next hop. Instead, each router independently simulates the forwarding process. The blacklists carried by the virtual packets are accumulated at all the routers these packets traverse.

Let $\mathbb{B}_i^d$ denote the set of possible blacklists seen by a node i in packets destined for d. When a packet p arrives at a node i in a simulation, $\mathbb{B}_i^d$ is updated, i.e., $\mathbb{B}_i^d \leftarrow \mathbb{B}_i^d \cup \{p.\text{blist}\}$. Note that p.blist is itself a set, and hence $\mathbb{B}_i^d$ is a set of sets. Once $\mathbb{B}_i^d$ is accumulated, for each distinct blacklist $b \in \mathbb{B}_i^d$, a forwarding entry that maps $\langle d,b\rangle$ to $\langle j,b'\rangle$ is computed, where j is the next hop and b' is the outgoing blacklist. Henceforth, packets are forwarded with a simple table lookup without any dynamic computation. While this simulation procedure seems involved, it need to be repeated only in response to infrequent long-term changes in the globally advertised topology, since short-term link and node failures are handled with local rerouting. An analysis of the forwarding table computation complexity and lookup overhead with LOLS is presented in Section VI-B.

B. Encoding Blacklist in a Packet

A straightforward way to convey a blacklist in the packet header is to represent it as a list of link identifiers. While that is a reasonable approach when dealing with arbitrary number of failures, precomputation of potential blacklists and forwarding tables at each node for protection against predefined failure scenarios offers an opportunity to reduce the number of bits needed to convey the blacklist information considerably.

The basic idea is to maintain $\mathbb{B}_i^d$ as an array of blacklists. Then, a neighbor h of node i can convey the blacklist in a packet destined for d by simply specifying the index into $\mathbb{B}_i^d$, or *bindx*. A bindx of 0 could be used to mean an empty blacklist. In other words, instead of a blacklist, packets now carry a bindx field which effectively conveys blacklist of $\mathbb{B}_i^d$[bindx] to node i. We should point out that every node independently but consistently computes $\mathbb{B}_i^d$ for each node i. Also, note that the bindx value is meaningful only between neighbors and therefore it is local in scope. Hence, its value could change at each hop. However, the forwarding table lookup is similar to that based on blist, $\langle d,b\rangle$ is mapped to $\langle j,b'\rangle$ where b is the incoming bindx, j is the next hop and b' is the outgoing bindx. The forwarding entries based on bindx corresponding to Table II are shown in Table III. Instead of blacklists {A→C}, {B→E}, {A→C, B→E}, in packets and these tables, we can use indices $1, 1, 2$ respectively. Since the size of $\mathbb{B}_i^d$ would be much smaller than all possible blacklists, the number of bits needed to encode bindx would be much lower. We will show later in the evaluation that bindx takes up less than a byte when preparing for two link/node failures.

It is possible that the addition of even a few bits to the IP header could be considered an obstacle to practical deployment of LOLS. Therefore, we propose an alternative approach similar to Not-via to convey blacklist information using encapsulation. With this approach, a node adjacent to a failure would, instead of encoding the blacklist in the packet header, encapsulate the packet with another header containing

TABLE III
BLACKLIST INDEX BASED FORWARDING ENTRIES CORRESPONDING TO TABLE II

destination	bindx 0 next	bindx 0 bindx	bindx 1 next	bindx 1 bindx	bindx 2 next	bindx 2 bindx
B	B	0				
C	D	1				
D	D	0				
E	B	1	D	2	D	2
F	D	1				
G	D	0				
H	D	0				

(A)

destination	bindx 0 next	bindx 0 bindx	bindx 1 next	bindx 1 bindx	bindx 2 next	bindx 2 bindx
A	A	0				
B	A	0				
C	A	0	C	0		
E	A	0			C	0
F	G	0	G	0		
G	G	0				
H	G	0				

(D)

destination	bindx 0 next	bindx 0 bindx
A	D	0
B	D	0
C	D	0
D	D	0
E	D	0
F	F	0
H	F	0

(G)

a new destination address. This address has a special meaning and performs the function of the bindx field in the packet. In other words, a packet with destination d and blacklist b is mapped to a destination with address d'. The encapsulated destination address d' indicates that the packet should be forwarded "not-via" the corresponding blacklist b. Note that similar to the bindx field in the packet, whose scope is local to a forwarding node and the next-hop, the not-via-blacklist addresses have only local meaning and are interpreted consistently by the two adjacent nodes along a path.

With such an implementation of LOLS, a packet is forwarded only based on the destination address field in the packet instead of destination and bindx. Once again, except that the blacklist information is conveyed by the destination address instead of the bindx field, there is no difference in a packet's flight. Moreover, we can map between a (destination, bindx) and a private not-via address (given the range of addresses), similar to the way bindx can be determined consistently by neighboring nodes without explicit signaling. The next section presents the evaluation results which demonstrate that LOLS needs only a fairly small number of not-via addresses in order to convey blacklist information.

V. PERFORMANCE EVALUATION OF LOLS

We present a proof in Appendix A that LOLS guarantees loop-free forwarding if the destination is reachable. Therefore, in this section, we focus on evaluating the overhead due to LOLS. First, we show that LOLS is more scalable than FCP in dealing with arbitrary failures. Second, we verify that LOLS does not reroute packets along long detours, and also does not cause overloading of links in the network.

For our evaluation, we use the Abilene topology as well as five other backbone topologies that are obtained with Rocketfuel [37]. These networks are summarized in Table IV.

A. Scalability of LOLS

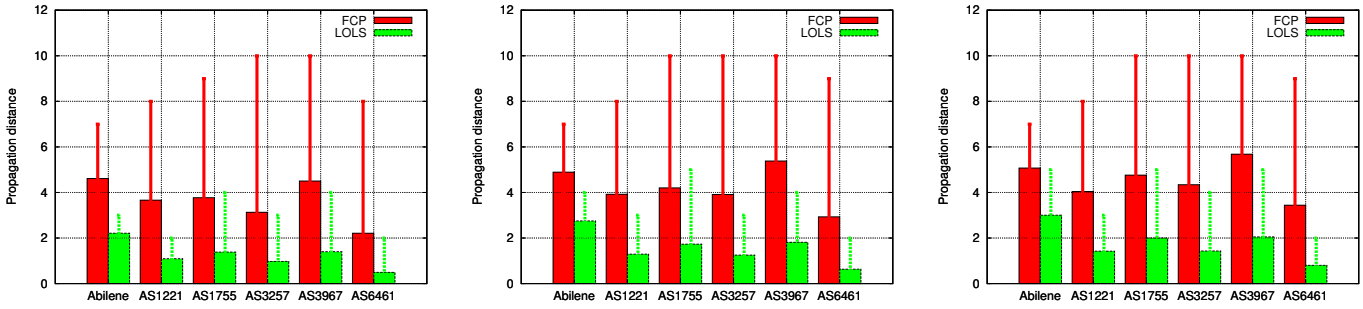
We measure the scalability of LOLS in terms of: i) how far the information about a degraded link is propagated via a packet's blacklist and ii) how large is the blacklist of a packet under LOLS. Obviously, the shorter the propagation distance, the smaller the blacklist size, the better the scalability.

TABLE IV
REAL NETWORKS USED IN EVALUATION

AS Number	Name	# of routers	avg. degree
	Abilene (US)	12	2.33
1221	Telstra (Australia)	108	2.82
1755	Ebone (Europe)	87	3.70
3257	Tiscali (Europe)	161	4.07
3967	Exodus (US)	79	3.72
6461	Abovenet (US)	141	5.30

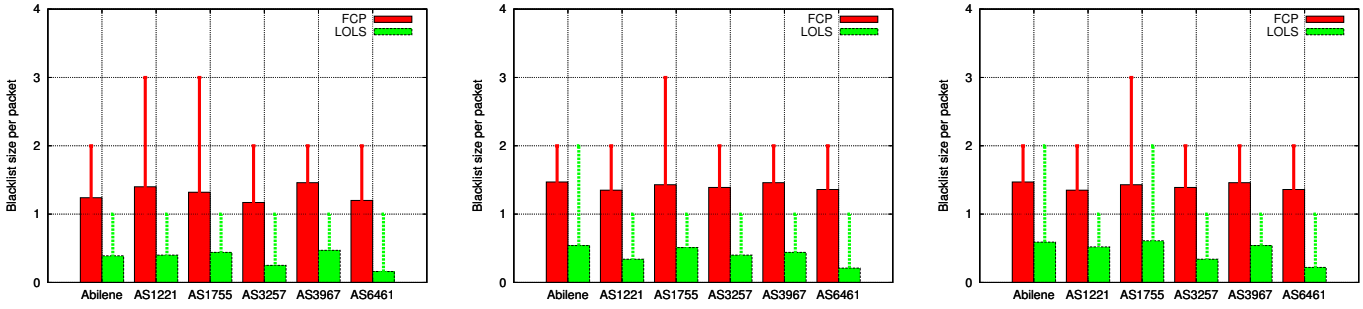
Failure propagation distance: We keep track of the distance in hops from the node adjacent to a failure to the farthest node that received the failure information through a packet's blacklist. Fig. 2 presents the failure propagation distance under both LOLS and FCP for scenarios with 2, 3, and 4 failures. Clearly, the propagation distance with LOLS is much shorter than that under FCP. On average, LOLS propagates a failure to nodes that are 2 or 3 hops away. But in some cases, it needs to inform nodes that are farther than 5 hops to ensure loop-free forwarding. This points out the limitation of schemes such as [28] based on locally scoped updates with a fixed scope. The chosen scope could be more than sufficient in some cases and less than necessary in other cases resulting in either unnecessary overhead or packet drops and forwarding loops. At the other extreme, FCP guarantees to be loop-free by carrying the blacklist information all the way to the destination which in most cases is unnecessary. On the other hand, LOLS localizes the blacklist propagation whenever possible and in rare cases propagates the blacklist to distant nodes when necessary for ensuring loop-free packet delivery.

Size of blacklist in a packet: Another measure of the overhead of LOLS and FCP is the size of blacklist carried by each packet. Apart from additional bits needed to convey this information in the packet, blacklist size also impacts the forwarding complexity at each hop. Therefore, we compare the blacklist size under LOLS and FCP in Fig. 3. It is evident that most LOLS packets do not have to carry a non-empty blacklist even when there are 4 failed links/nodes in the network. On the contrary, blacklist size under FCP is several times more than



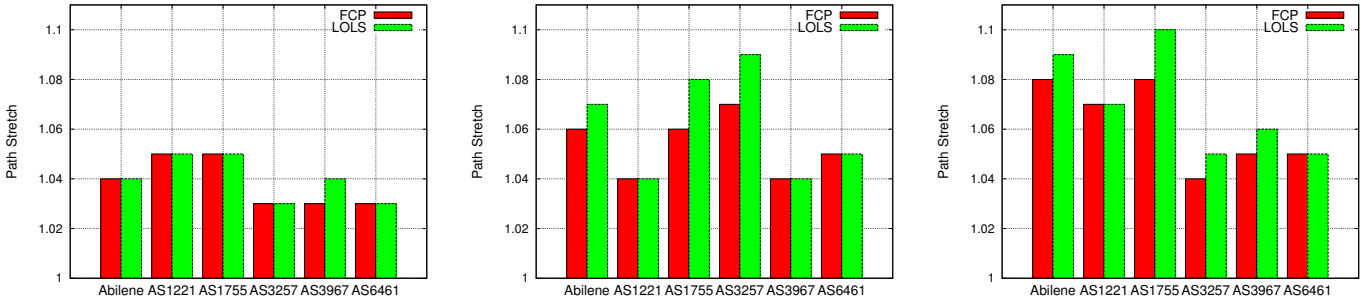
(a) Average propagation distance for 2 failures (b) Average propagation distance for 3 failures (c) Average propagation distance for 4 failures

Fig. 2. Failure propagation distance: Using LOLS, information about a failure is propagated to nodes that are much fewer hops away than under FCP. The above graphs depict the average propagation distance for 2, 3, or 4 failures, respectively. The error bars represent the 90th percentile distance for each case.



(a) Average blacklist size for 2 failures (b) Average blacklist size for 3 failures (c) Average blacklist size for 4 failures

Fig. 3. Blacklist size in a packet: With LOLS, a packet's blacklist is reset as soon as it makes forward progress towards the destination. Consequently, most packets do not have to carry a non-empty blacklist. Overall, the size of blacklist in a packet under LOLS is only a fraction of that under FCP. The graphs depict the average blacklist size of a packet for 2, 3, or 4 failures, respectively. The error bars represent the 90th percentile blacklist size for each case.



(a) Average path stretch for 2 failures (b) Average path stretch for 3 failures (c) Average path stretch for 4 failures

Fig. 4. Path stretch: Both LOLS and FCP yield low stretch paths. Although LOLS resets the blacklist upon forward progress, its stretch is similar to FCP's.

that under LOLS. This is a testimony of the effectiveness of on-demand state propagation approach of LOLS. These results indicate that overheads due to LOLS are relatively small, and establish LOLS as a scalable scheme for reliable delivery.

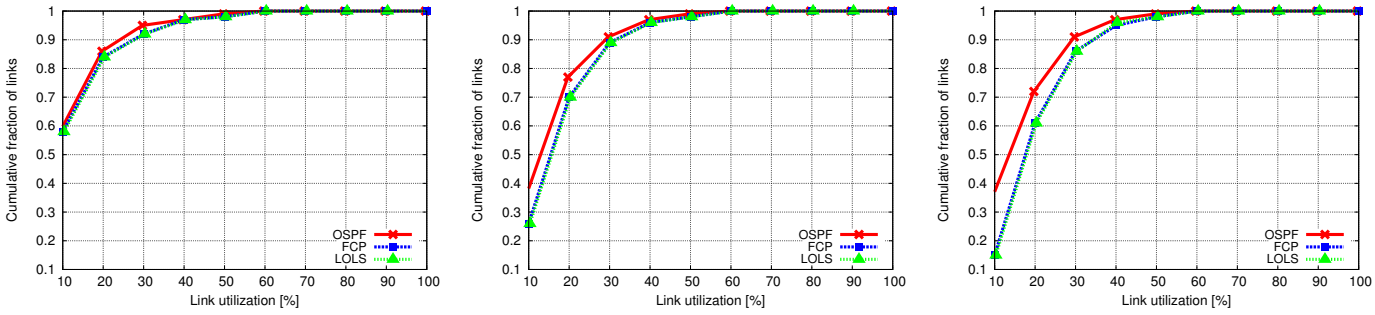
B. Optimality of LOLS

Under both LOLS and FCP, a packet takes the usual shortest path until it encounters a failure and then gets rerouted along the alternate path. Consequently, in the presence of failures, these schemes may forward packets along longer paths compared to the optimal paths computed based on the global link state updates. Furthermore, with LOLS, it is possible that after the packet's blacklist gets reset, a failure downstream towards the destination can cause a packet to backtrack, gather a larger blacklist, and reach the destination via a longer detour. FCP is not expected to have this problem as it carries failure information all the way to the destination. Hence, it is important to investigate whether the gains of LOLS over FCP in terms of failure propagation and blacklist size come at the expense of suboptimal paths. Additionally, we study whether local rerouting of packets with LOLS causes excessive overloading of some links in the network.

Path stretch: The *stretch* of a path between a pair of nodes is defined as the ratio of the path cost under a given scheme, and the optimal shortest path. Without failures, there is no difference between the LOLS paths and the optimal shortest paths, and so the stretch is 1. The stretch due to LOLS and FCP for the pairs of nodes affected by failures is shown in Fig. 4.

TABLE V
MAXIMUM LINK UTILIZATION ACROSS DIFFERENT TOPOLOGIES

Topology	Abilene	AS1221	AS1755	AS3257	AS3967	AS6461
One failure (OSPF)	0.593	0.697	0.568	0.571	0.573	0.504
One failure (LOLS)	0.609	0.697	0.645	0.526	0.707	0.532
One failure (FCP)	0.609	0.697	0.645	0.526	0.707	0.532
Two failures (OSPF)	0.632	0.697	0.740	0.647	0.751	0.567
Two failures (LOLS)	0.738	0.697	0.817	0.675	0.840	0.535
Two failures (FCP)	0.706	0.697	0.817	0.675	0.840	0.535
Three failures (OSPF)	0.692	0.759	0.736	0.636	0.811	0.568
Three failures (LOLS)	0.738	0.803	0.819	0.675	0.804	0.532
Three failures (FCP)	0.728	0.803	0.817	0.675	0.804	0.532



(a) Maximum utilization for 1 failure (b) Maximum utilization for 2 failures (c) Maximum utilization for 3 failures

Fig. 5. Cumulative distribution of the top 20% utilized links in AS6461: OSPF represents shortest path routing excluding the failed links/nodes. Compared to OSPF, LOLS increases the utilization on some moderately loaded links but does not heavily overload any one link. LOLS and FCP are almost indistinguishable.

The average stretch due to LOLS, across varying number of failures, is less than 1.1 and quite similar to that of FCP. In other words, resetting (forgetting) the blacklist upon making forward progress does almost no harm in finding shortest paths to the destination. These results affirm that compared to FCP, the LOLS offers similar reliability and optimality, but better scalability in dealing with an arbitrary number of failures.

Load Distribution: As mentioned earlier, LOLS does not explicitly attempt to distribute load while rerouting around a failure. However, to understand the impact of local rerouting with LOLS on network congestion, we compare the load distribution under LOLS with that of OSPF and FCP.

To simulate traffic matrix, we adopt the gravity model as it was used in [26]. Briefly, this model generates traffic $r(s,t)$ between a source s and a destination t such that $r(s,t) = b_s \frac{e^{a_t}}{\sum_{i \in V \setminus \{s\}} e^{a_i}}$. Here, b_s represents the total traffic originating at a node s, which is chosen randomly, and a_t represents the "mass" of a node, which is proportional to the number of links adjacent to that node. The larger the mass of a node, the higher the traffic it attracts, thus the model's name. In short, the traffic originating at each node is chosen randomly, while the traffic attracted by a node is derived based on this model. Given the traffic matrix, we determine the load on every link in the network under shortest path routing. We then provision each link in the network with a capacity equal to twice that of the highest link load in the network. Therefore, the utilization for any link in the network is ≤ 0.5 prior to any failures.

To compare the distribution of load under LOLS with that under OSPF and FCP, we run simulations of 1 to 3 random failures in the network and collect statistics over 100 runs of each category. For each scenario of failures, we record the highest link utilization observed in the network, which is shown in Table V. To provide more insight on the performance, in Fig. 5, we plot the cumulative distribution of utilization of the top 20% of links in the AS6461 topology (other topologies also exhibited similar pattern and are not shown here for brevity of space). Fig. 5 shows that the performance of LOLS is comparable to FCP and lower than OSPF but not by large margin. LOLS increases the utilization of some moderately loaded links but does not heavily overload any one particular link. We believe this is because LOLS reroutes to the destination (not to the other end of the failed link/node as done by other fast reroute schemes such as Not-Via). Therefore, not all the flows that were previously passing through the failed link/node are rerouted along the same recovery path. Instead, these flows get rather distributed along different alternate shortest paths to their respective destinations, thus mitigating the potential for congestion with LOLS. This aspect of LOLS requires more thorough investigation which we intend to pursue as part of our future work.

VI. PERFORMANCE EVALUATION OF PRACTICAL LOLS

We now evaluate the practical version of LOLS that protects against a predefined set of failures. We focus particularly on

recovering from up to two link or node failures. We demonstrate that LOLS needs only a modest number of additional bits or not-via addresses for providing protection against any two link/node failures. In addition, we analyze the computational complexity of both real-time forwarding and blacklist precomputation using practical LOLS.

A. Practicality of LOLS

First, we describe an optimization we implemented for reducing the size of blacklist array at a router. Next, we present a way to perform interface-specific forwarding that allows a blacklist to be encoded using fewer bits. We then report the evaluation results which reveal that LOLS needs only 6 bits to encode a blacklist. Finally, we show how the blacklist can be conveyed using a relatively small number of not-via addresses.

Reducing the array of blacklists at a router: Recall that a blacklist carried by a packet under LOLS is a set of directed links. When a link $i \rightarrow j$ fails, node i may add it to a packet's blacklist. Even in case of node j failure, node i may treat it as a link failure, and hence blacklist $i \rightarrow j$ only. Ideally, we need to blacklist node j or all the links adjacent to node j. Otherwise, it is possible that a packet may get rerouted to another node, say x, that is adjacent to j which adds $x \rightarrow j$ to the blacklist. Apart from elongating the stretch, this increases the set of unique blacklists at a router. This is because different packets may arrive at the router carrying a different subset of links adjacent to the failed node. To mitigate this problem, we propose to infer node failures from the number of links in the blacklist. When preparing for two failures, if there are three or more links in the blacklist, we check to see if two of those links are adjacent to the same node. For instance, if $i \rightarrow j$ and $x \rightarrow j$ are two of the three links in the blacklist, we can blacklist node j by adding all its adjacent links to the blacklist. While this may appear to increase the size of a packet's blacklist, it actually reduces the set of blacklists at a router and in turn the number of bits needed to encode blacklist in the packet.

Interface-specific forwarding: It is possible to achieve better blacklist encoding by leveraging the existing router architectures. Currently, though forwarding is based on destination IP address only, routers maintain a copy of the forwarding table at each line card of an interface for lookup efficiency. This allows us to compute a different forwarding table for each interface and perform interface-specific forwarding. To compute interface-specific forwarding tables, a set of possible blacklists seen by the node has to be maintained per interface as follows. Let $\mathbb{B}^d_{h \rightarrow i}$ denote the set of possible blacklists seen by a node i in packets destined for d arriving from the previous hop h. When a packet p arrives through interface $h \rightarrow i$ in a simulation, $\mathbb{B}^d_{h \rightarrow i}$ is updated, i.e., $\mathbb{B}^d_{h \rightarrow i} \leftarrow \mathbb{B}^d_{h \rightarrow i} \cup \{p.\text{blist}\}$. Then, a neighbor h of node i can convey the blacklist in a packet destined for d by simply specifying the index into $\mathbb{B}^d_{h \rightarrow i}$. Since the size of $\mathbb{B}^d_{h \rightarrow i}$ would be smaller than all blacklists $\mathbb{B}^d_i$ seen by node i with many interfaces, the number of bits needed to encode the index would be lower.

Blacklist encoding overhead with LOLS: We now present the results of our evaluation of per-packet overhead with LOLS to convey the blacklist information. Table VI shows the average and maximum size of the blacklist array accumulated at a node for dealing with two link/node failures in several real topologies. Here, interface-specific corresponds to the approach described above where forwarding of a packet depends on its incoming interface, and interface-agnostic refers to conventional interface independent forwarding. These results show that on average, a node sees less than 10 distinct blacklists among all possible one or two link/node failures across all topologies. As expected, interface-specific forwarding further reduces the blacklist array size, bringing down the average to less than 1.5. Even in the worse case, the maximum number of distinct blacklists at a node even without being interface-specific is less than 60. Therefore, 6 bits would be sufficient to encode blacklist information for providing protection against any one or two link/node failures in all of these topologies.

TABLE VI
SIZE OF BLACKLIST ARRAY

Topology	interface-specific		interface-agnostic	
	Avg	Max	Avg	Max
Abilene	0.79	6	4.36	10
AS1221	1.08	37	6.18	42
AS1775	1.43	41	9.40	58
AS3257	0.69	27	4.82	37
AS3967	1.23	31	7.77	55
AS6461	0.52	36	3.97	53

TABLE VII
NUMBER OF NOT-VIA ADDRESSES REQUIRED

AS Number	interface-specific	interface-agnostic
Abilene	31	79
1221	304	656
1755	774	1973
3257	755	1746
3967	781	2063
6461	1015	1979

Blacklist encoding through not-via addresses: The above evaluation results show that LOLS requires less than a byte of space in the packet to encode the blacklist, which is quite reasonable for guaranteeing protection against any two failures in the network. To make this scheme more amenable to practical deployment, we can convey blacklist information using not-via addresses as explained in Section IV-B. To reiterate, a packet with destination d which is supposed to carry blacklist b is encapsulated in another packet with not-via address d', where d' implicitly indicates that the packet should be forwarded "not-via" the corresponding blacklist b. Note that such an encapsulation is done only when a packet has to carry a non-empty blacklist over a few hops near a failure. One may argue that the Not-via scheme, which is designed for single failures, could be applied for protection against two failures. A straightforward application of Not-via for dealing with two

failures may potentially need O($|\mathcal{E}|^2|\mathcal{V}|$) not-via addresses (and forwarding entries). We show that LOLS can handle two failures with several orders of magnitude less addresses. Table VII gives the results for six real topologies considered in our evaluation. As before, interface-specific forwarding requires fewer addresses. Even with interface-agnostic forwarding on topologies with up to 160 nodes, the implementation of LOLS needs only around 2000 not-via addresses. Once again, these results establish the scalability and practicality of LOLS.

B. Complexity Analysis of LOLS

LOLS deviates from the conventional destination based forwarding and instead determines the next hop based on both blacklist and destination. Therefore, it is pertinent to ask, what is the cost of computing and looking up the blacklist based forwarding table. Below, we analyze the complexity of the practical version of LOLS for dealing with any two failures.

Blacklist based forwarding table lookup: The traditional IP address lookup involves identifying the longest matching prefix. A wide variety of methods have been proposed to speed up lookup time [38]. A representative approach performs binary search on prefix length, which takes O($\log w$) time, where w is the number of bits in the IP address [38]. Along those lines, it is possible to perform blacklist based forwarding table lookup in O($\log(w+b)$) time, where b is the number of bits needed to encode blacklist. Considering that 6 bits are sufficient to encode blacklist for the evaluated topologies, we argue that the lookup time for forwarding under LOLS would be in the same order as that of traditional forwarding.

Blacklist based forwarding table computation: The time consuming part of this computation is the collection of blacklists at each router. It involves simulating the forwarding of packets for each possible failure scenario. Since typical networks are sparse, as is the case with the real topologies listed in Table IV, the number of failure scenarios to consider are $O(|\widetilde{\mathcal{V}}|^2)$. For each failure scenario, we need $|\widetilde{\mathcal{V}}|$ computations of Dijkstra – one for each router in the network, which takes $O(|\widetilde{\mathcal{V}}|^2 \log |\widetilde{\mathcal{V}}|)$. We then simulate forwarding of packets from nodes that are adjacent to failures to all other nodes in the network, which is $O(|\widetilde{\mathcal{V}}|)$. In general, it is not necessary that a packet has to be forwarded all the way to its destination, since blacklist gets reset upon forward progress. Considering that blacklist propagation distance in most cases is less than 4 hops, the overall complexity is $O(|\widetilde{\mathcal{V}}|^5 \log |\widetilde{\mathcal{V}})|$. While this complexity seems high, note that only when a failure lasts beyond a threshold duration, non-adjacent nodes get notified of the failure and recompute their blacklist based forwarding tables. Since most failures are transient, they do not trigger link state advertisements and cause recomputation of forwarding tables. Nevertheless, we intend to improve upon this complexity by making use of optimizations such as incremental shortest path tree computation.

VII. Issues and Discussion

We have thus far presented LOLS, evaluated its performance, and demonstrated its potential benefits. Now we discuss some of the issues pertinent to its practical deployment.

What would be the disruption time with LOLS? The disruption time due to a link failure under LOLS comprises the time periods for the link failure detection and initiation of local rerouting. There are various link failure detection mechanisms employed in practice, with response time ranging from tens of milliseconds to a few seconds. For example, a line card can typically detect loss of electrical connectivity or loss of light in tens of milliseconds. On the other hand, hellos as implemented in IGP protocols result in much slower failure detection, typically in seconds. Given the dominance of packet over SDH/SONET links in backbone networks that have built-in hardware failure detection capability, it is reasonable to assume that the link failure detection time is around $20ms$ [9]. Assuming that the forwarding plane is pre-loaded with failover forwarding tables, a router upon detecting an adjacent failure can initiate local rerouting within a few milliseconds, keeping the total disruption time sub-50ms.

How to realize changes to packet format and forwarding process? A practical hurdle to deployment of LOLS is that it requires carrying blacklist information in the packet and forwarding based on both blacklist and destination. Though the blacklist is mostly empty and less than a byte is needed to encode it, LOLS necessitates changes to the forwarding plane. A way to convey blacklist information in the packet without changing its format is to employ not-via addressing as described earlier. A straightforward application of not-via addressing for dealing with two failures may potentially need O($|\widetilde{\mathcal{E}}|^2$) not-via addresses (and forwarding entries) to convey the two failure information.LOLS can deal with two failures using much fewer number of not-via addresses as shown in Table VII. This gain is due to the local semantics of these addresses and thus at the expense of not-via address potentially changing at a few hops around the failures. We plan to study whether we can assign global addresses to convey blacklists and eliminate this per-hop forwarding overhead without using excessive number of not-via addresses. Another alternative suggested previously [25] is to exploit the MPLS infrastructure in todays hardware routers. A blacklist index may be embedded into an MPLS label upon a packets entrance to an ISPs network and swapped at each hop during forwarding. We need to further explore different ways of implementing LOLS.

When to trigger link state advertisement and initiate route convergence? The main idea behind LOLS is to perform local rerouting in case of a failure instead of advertising it to the whole network. This approach of suppressing failure notification makes sense when most of the failures are short-lived, lasting no longer than a few minutes [2]. However, if a failure persists beyond a threshold duration (say 5 minutes), it may be better to advertise it and let all routers in the network recompute their routing tables. Longer threshold duration increases the probability of multiple suppressed failures.

However, since LOLS continues forwarding despite multiple failures, the choice of threshold value mainly affects the routing optimality. We should however point out that currently LOLS can not guarantee loop-free forwarding during route convergence. Previously proposed approaches such as ordered updates for avoiding loops during convergence only work with single failures [39]. A recently proposed scheme for fast reroute with fast convergence (FCFR) [40] is also meant for single failures but amenable for extension to multiple failures. We are currently integrating FCFR with LOLS to ensure loop-free rerouting and convergence even with multiple failures.

VIII. Conclusions and Future Work

In this paper, we presented LOLS, a localized on-demand link state routing for handling multiple failures in IP backbone networks. The core idea behind LOLS is to have packets carry a blacklist of degraded links encountered along the path that are to be avoided in order to ensure loop-free forwarding. The key feature of LOLS is that a packet's blacklist is reset as soon as it makes forward progress towards the destination, limiting the propagation of failure information to just a few hops. We have proved that LOLS guarantees loop-free forwarding to reachable destinations regardless of the number of failures in the network. We have evaluated the overhead due to LOLS using several large real topologies and shown that it scales better than the recently proposed scheme FCP which has similar failure resilience objectives. We have also presented a practical version of LOLS for protecting against predefined failures, and shown that it needs only a modest number of header bits or not-via addresses for handling any two link/node failures. Our plan is to implement a prototype of LOLS using Mininet system [41] to demonstrate its deployability.

IX. Acknowledgements

This research is partially funded by the National Science Foundation (NSF) through grants CNS-0448272 and CNS-0551650. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

References

[1] G. I. et al, "Analysis of link failures in an IP backbone," in *Proc. ACM IMW*, Nov. 2002.

[2] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational ip backbone network," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, Aug. 2008. [Online]. Available: http://dx.doi.org/10.1109/TNET.2007.902727

[3] A. Gonza andlez and B. Helvik, "Analysis of failures characteristics in the uninett ip backbone network," in *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, march 2011, pp. 198 –203.

[4] O. B. et al, "Achieving Sub-50 Milliseconds Recovery Upon BGP Peering Link Failures," in *CoNEXT*, Oct. 2005.

[5] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets." in *SIGCOMM*, 2007, pp. 241–252.

[6] S. S. Lor, R. Landa, and M. Rio, "Packet re-cycling: Eliminating packet losses due to network failures," in *HotNets*, Oct. 2010.

[7] S. Kini, S. Ramasubramanian, A. Kvalbein, and A. Hansen, "Fast Recovery from Dual Link or Single Node failures in IP Networks Using Tunneling," *IEEE/ACM Trans. Networking*, vol. 18, no. 6, pp. 1988–1999, Dec. 2010.

[8] C. Alattinoglu and S. Casner, "ISIS routing on the Qwest backbone: A recipe for subsecond ISIS convergence," NANOG meeting, Feb. 2002.

[9] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, "Achieving subsecond IGP convergence in large IP networks," in *ACM SIGCOMM Computer Communication Review*, Jul. 2005.

[10] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford, "Dynamics of hot-potato routing in IP networks," in *Proc. ACM Sigmetrics*, Jun. 2004.

[11] V. Sharma and F. Hellstrand, "Framework for MPLS-based recovery," RFC 3469, Feb. 2003.

[12] M. Tacca, K. Wu, A. Fumagalli, and J.-P. Vasseur, "Local detection and recovery from multi-failure patterns in mpls-te networks," in *Communications, 2006. ICC '06. IEEE International Conference on*, vol. 2, june 2006, pp. 658 –663.

[13] S. Bryant, M. Shand, and S. Previdi, "IP fast reroute using not-via addresses," Internet Draft(work in progress), Mar. 2006, draft-bryantshand-IPFRR-notvia-addresses-02.txt.

[14] A. K. et al, "Fast IP Network Recovery using Multiple Routing Configurations," in *Proc. IEEE Infocom*, Apr. 2006.

[15] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah, "Fast Local Rerouting for Handling Transient Link Failures," *IEEE/ACM Trans. Networking*, vol. 15, no. 2, pp. 359–372, Apr. 2007.

[16] S. I. et al, "An approach to alleviate link overload as observed on an IP backbone," in *Proc. IEEE Infocom*, Mar. 2003.

[17] S. Rai, B. Mukherjee, and O. Deshpande, "IP Resilience within an Autonomous System: Current Approaches, Challenges, and Future Directions," *IEEE Commun. Mag.*, pp. 142–149, Oct. 2005.

[18] S. N. et al, "Blacklist-aided forwarding in static multihop wireless networks," in *SECON*, Sep. 2005.

[19] B. Karp and H. T. Kung, "GPSR: Greedy Perimeter Stateless Routing for wireless networks," in *Proc. ACM Mobicom*, 2000, pp. 243–254. [Online]. Available: citeseer.ist.psu.edu/karp00gpsr.html

[20] S. Bryant, M. Shand, and S. Previdi, "IP Fast Reroute using Not-via Addresses," Internet Draft(work in progress), Jul. 2007, draft-ietf-rtgwg-ipfrr-notvia-addresses-01.txt.

[21] A. Atlas, "U-turn alternates for IP/LDP fast-reroute," IETF Internet Draft, Feb. 2006, draft-atlas-ip-local-protect-uturn-03.txt.

[22] M. Menth, M. Hartmann, R. Martin, T. Cicic, and A. Kvalbein, "Loop-free alternates and not-via addresses: A proper combination for ip fast reroute?" *Computer Networks*, vol. 54, no. 8, pp. 1300–1315, 2010.

[23] T. Cicic, A. F. Hansen, A. Kvalbein, M. Hartmann, M. Menth, R. Martin, S. Gjessing, and O. Lysne, "Relaxed Multiple Routing Configurations: IP Fast Reroute for Single and Correlated Failures," *IEEE Transactions on Network and Service Management*, vol. 6, no. 1, 2009.

[24] G. Retvari, J. Tapolcai, G. Enyedi, and A. Csaszar, "Ip fast reroute: Loop free alternates revisited," in *INFOCOM*, 2011.

[25] A. Li, X. Yang, and D. Wetherall, "SafeGuard: Safe Forwarding during Routing Changes," in *CoNEXT*, 2009.

[26] K.-W. Kwong, L. Gao, R. Guerin, and Z.-L. Zhang, "On the Feasibility and Efficacy of Protection Routing in IP Networks," in *INFOCOM*, 2010.

[27] C. Santivanez, R. Ramanathan, and I. Stavrakakis, "Making link-state routing scale for ad hoc networks," in *Proc. ACM MobiHOC*, 2001.

[28] M. Gerla, X. Hong, and G. Pei, "Fisheye state routing protocol for ad hoc networks," IETF Internet Draft, Jun. 2002, draft-ietf-manet-fsr-03.txt.

[29] D. Anderson, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient overlay networks," in *SOSP*, 2001.

[30] X. Yang and D. Wetherall, "Source selectable path diversity via routing deflections," in *Proc. ACM Sigcomm*, Sep. 2006.

[31] M. Motiwala, N. Feamster, and S. Vempala, "Path splicing: Reliable connectivity with rapid recovery," in *SIGCOMM*, 2008.

[32] F. Kuhn, R. Wattenhofer, Y. Zhang, and A. Zollinger, "Geometric Ad-Hoc Routing: Of Theory and Practice," in *Proc. 22nd ACM Int. Symposium on the Principles of Distributed Computing (PODC)*, 2003. [Online]. Available: citeseer.ist.psu.edu/kuhn03geometric.html

[33] N. Wang, K. H. Ho, G. Pavlou, and M. Howarath, "An overview of routing optimization for internet traffic engineering," *IEEE Communications Surveys & Tutorials*, vol. 10, no. 1, pp. 36–56, 2008.

[34] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot, "Igp link weight assignment for operational tier-1 backbones," *IEEE/ACM Transactions on Networking*, vol. 15, no. 4, pp. 789–802, Aug. 2007.

[35] A. Kvalbein, T. Cicic, and S. Gjessing, "Post-failure routing performance with multiple routing configurations," in *IEEE INFOCOM 2007*, E. M. George Kesidis and R. Srikant, Eds. IEEE, 2007.
[36] W. Lau and S. Jha, "Failure-oriented path restoration algorithm for survivable networks," *Network and Service Management, IEEE Transactions on*, vol. 1, no. 1, pp. 11 –20, 2004.
[37] N.Spring, R. Mahajan, and D. Wetherall, "Measureing ISP topologies with Rocketfuel," in *ACM SIGCOMM*, Aug. 2002.
[38] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of ip address lookup algorithms," *Network, IEEE*, vol. 15, no. 2, pp. 8 –23, mar/apr 2001.
[39] P. Francois and O. Bonaventure, "Avoiding Transient Loops during IGP Convergence in IP Networks," *ACM Transactions on Networking*, vol. 15, no. 6, pp. 1280–1292, Dec. 2007.
[40] G. Robertson, J. Bedenbaugh, and S. Nelakuditi, "Fast convergence with fast reroute in ip networks," in *IEEE HPSR*, Jun. 2010.
[41] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks (at scale!)," in *ACM HotNets*, Oct. 2010.

APPENDIX

A. Proof of Loop-Free Forwarding with LOLS

We now prove that LOLS guarantees loop-free delivery of a packet if the destination is reachable. Intuitively, this is true because of the invariant that at each hop we have either (i) *a decrease in the cost to p.dest* in $\mathcal{G} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$; or ii) *an addition to p.blist*; or iii) *a decrease in the cost to p.dest in graph* $\mathcal{G} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}} \setminus p.\text{blist})$. Based on this observation, we define a metric on a packet's forwarding state such that the metric is always strictly decreasing at each hop, thereby guaranteeing loop-freedom[2]. The formal proof is given below.

Theorem: Assuming that the set of all links $\widetilde{\mathcal{E}}$ and the set of all degraded links $\widetilde{\mathcal{B}}$ (where $\widetilde{\mathcal{B}} = \bigcup_i \widetilde{\mathcal{B}}_i$) do not change during the life time of a packet p, LOLS guarantees that *if there exists a path* between p.source and p.dest in $\mathcal{G} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}} \setminus \widetilde{\mathcal{B}})$, then p *will be delivered* in a finite number of hops, and *if there is no path, p will be dropped* in a finite number of hops.

Proof: We first prove that p will never be caught in an infinite forwarding loop. Suppose i is the node currently forwarding p. Let us assume that link costs are integers as they can be scaled if necessary. Making use of the invariant property mentioned above, we can define an integer metric $m(p)$ based on the forwarding state of p such that $m(p)$ always decreases at each hop. Specifically, we define $m(p) = \alpha \times \mathcal{C}_{i \rightsquigarrow p.dest}(\tilde{\mathcal{E}}) - \beta \times |p.\text{blist}| + \mathcal{C}_{i \rightsquigarrow p.dest}(\tilde{\mathcal{E}} \setminus p.\text{blist})$, where $|p.\text{blist}|$ is the size of p's blacklist[3]. Note that each of the three components of $m(p)$ corresponds to one of component in the invariant property. α and β are integer parameters (of graph $\mathcal{G} = (\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}})$) chosen such that the combined effects of the invariant property make $m(p)$ decreasing at each hop. For example, we can choose $\beta > \max_{x,y,\mathcal{E}} \{\mathcal{C}_{x \rightsquigarrow y}(\mathcal{E}) | \forall x, y \in \widetilde{\mathcal{V}}, \forall \mathcal{E} \subseteq \widetilde{\mathcal{E}}, \mathcal{C}_{x \rightsquigarrow y}(\mathcal{E}) < \infty\}$. In that way, when $\mathcal{C}_{i \rightsquigarrow p.dest}(\tilde{\mathcal{E}})$ does not change in a hop, the decrease of $m(p)$ due to increase of $|p.\text{blist}|$ (at least 1) always dominates the (potential) increase of $m(p)$ due to its third component. Similarly, we can define $\alpha > \beta \times (|\widetilde{\mathcal{E}}| + 1)$, such that any decrease of $\mathcal{C}_{i \rightsquigarrow p.dest}(\tilde{\mathcal{E}})$ (at least 1) will dominate the rest of $m(p)$. Based on the definition of $m(p)$, we observe that $m(p)$ is always an integer; it starts as a finite positive integer and never goes negative; and it strictly decreases at every hop. Therefore, p can not be caught in an infinite forwarding loop, since that would imply an inevitable negative $m(p)$ value after certain number of steps. In other words, p is either delivered or dropped after a finite number of hops.

We now show that p is always delivered if there exists a path in $(\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}} \setminus \widetilde{\mathcal{B}})$. We prove the contrapositive. Suppose p is dropped. This can only happen when LOLS algorithm returns $\emptyset$. In that case, there is no path from i to p.dest in $(\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}} \setminus p.\text{blist})$. Since $p.\text{blist} \subseteq \widetilde{\mathcal{B}}$, we therefore have $\widetilde{\mathcal{E}} \setminus p.\text{blist} \supseteq \widetilde{\mathcal{E}} \setminus \widetilde{\mathcal{B}}$, and there is no path between i and p.dest in $(\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}} \setminus \widetilde{\mathcal{B}})$. Further, the fact that there exists a path from p.source to i in $(\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}} \setminus \widetilde{\mathcal{B}})$ implies that there exists no path from p.source to p.dest in $(\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}} \setminus \widetilde{\mathcal{B}})$. Otherwise, tracing back from i to p.source and then to p.dest constitutes a path from i to p.dest.

Finally, suppose there exists no path from p.source to p.dest in $(\widetilde{\mathcal{V}}, \widetilde{\mathcal{E}} \setminus \widetilde{\mathcal{B}})$. Since p can not be delivered and will never be caught in an infinite loop, the only possibility is that p will be dropped in a finite number of hops, concluding the proof. ■

[2]Note that such a metric would have been impossible should there exists an infinite forwarding loop. We also note that under LOLS, a packet p may be forwarded to a node it has visited before. However, in such a circumstance, p always has either a different p.cost or a different p.blist. Since the number of combinations of cost and blacklist is finite for a given graph, LOLS guarantees that p is never caught in an infinite forwarding loop.

[3]We define $m(p)$ only when $\mathcal{C}_{i \rightsquigarrow p.dest}(\tilde{\mathcal{E}} \setminus p.\text{blist}) < \infty$. Otherwise, the packet p is dropped by LOLS, and $m(p)$ is undefined.

Glenn Robertson Glenn Robertson received the B.S. in Electrical Engineering from the United States Military Academy at West Point, the M.S. in Computer Engineering from the Colorado Technical University, and the Ph.D. in Computer Science and Engineering from the University of South Carolina, Columbia. Currently, he is an Assistant Professor at United States Military Academy at West Point. His current research focus is on designing fast reroute schemes for IP backbone networks.

Srihari Nelakuditi (M'01) received the B.E. from Andhra University College of Engineering, Visakhapatnam, the M.Tech. from Indian Institute of Technology, Madras, and the Ph.D. in Computer Science and Engineering from University of Minnesota, Minneapolis. He is currently an Associate Professor with the University of South Carolina, Columbia. His research interests are in resilient routing, wireless networking, and mobile computing. Dr. Nelakuditi was a recipient of the NSF CAREER Award in 2005.