Optimal Recovery from Large-Scale Failures in IP Networks

Qiang Zheng*, Guohong Cao*, Tom La Porta*, and Ananthram Swami†
*Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA
†Army Research Laboratory, Adelphi, MD
* {quz103, gcao, tlp}@cse.psu.edu † ananthram.swami@us.army.mil

***Abstract*—Quickly recovering IP networks from failures is critical to enhancing Internet robustness and availability. Due to their serious impact on network routing, large-scale failures have received increasing attention in recent years. We propose an approach called Reactive Two-phase Rerouting (RTR) for intra-domain routing to quickly recover from large-scale failures with the shortest recovery paths. To recover a failed routing path, RTR first forwards packets around the failure area to collect information on failures. Then, in the second phase, RTR calculates a new shortest path and forwards packets along it through source routing. RTR can deal with large-scale failures associated with areas of any shape and location, and is free of permanent loops. For any failure area, the recovery paths provided by RTR are guaranteed to be the shortest. Extensive simulations based on ISP topologies show that RTR can find the shortest recovery paths for more than 98.6% of failed routing paths with reachable destinations. Compared with prior works, RTR achieves better performance for recoverable failed routing paths and uses much less network resources for irrecoverable failed routing paths.**

I. Introduction

The Internet has become a significant and widely used infrastructure for a wide range of communication and other services. However, failures are fairly common in the Internet for various reasons. As foundation of the Internet, IP networks are intrinsically robust, but usually need a long time to recover from failures. Interior Gateway Protocols (IGP) for intra-domain routing are designed to recover from failures through updating the routing table of routers. When a router detects a failure, it disseminates topology updates to the other routers in the same Autonomous System (AS). The topology updates trigger these routers to recompute routing paths and update their routing tables. This *IGP convergence* process begins when the failure is detected and finishes after routers update their routing tables. Unfortunately, the IGP convergence is a time consuming process, which usually takes several seconds even for a single link failure [1]. During IGP convergence, a large number of packets may be dropped due to invalid routes. Disconnection of an OC-192 link (10 Gb/s) for 10 seconds can lead to about 12 million packets being dropped on a link (assuming that the average packet size is 1,000 bytes). Under large-scale failures, packet loss would be much more severe.

Many events like natural disasters and intentional attacks can cause large-scale failures which usually lead to serious routing disruption [2], [3]. For example, Hurricane Katrina [4], the Taiwan earthquake in December 2006 [5], and the Wenchuan earthquake in May 2008 [6] destroyed a large portion of the Internet near the disaster location and resulted in serious routing disruption. Additionally, intentional attacks like terroristic events (e.g., 911 attack [7]), link cuts attacks [8], and attacks by weapons of mass destruction [9] can also lead to large-scale failures.

Many approaches have been proposed to recover disrupted routing during IGP convergence. However, they cannot effectively deal with large-scale failures, i.e., routers and links within an area all fail as shown in Fig. 1. A straightforward recovery method is to accelerate the IGP convergence via appropriately tuning the parameters related to failure detection, topology update dissemination, and routing path recomputation [10]. However, rapidly triggering the IGP convergence may cause route flapping and make networks more unstable [11]. Therefore, most existing recovery approaches are proactive and use precomputed backup paths to reroute traffic affected by failures during IGP convergence, e.g., [13]–[15]. These proactive approaches are built on an implicit assumption that there are only sporadic and isolated link failures in IP networks. Hence, they are not suitable for large-scale failures, because links and their backup paths may fail simultaneously.

Two features make recovery from large-scale failures quite difficult. First, the failure area can be of any shape and at any place, and thus we cannot foresee which routers and links will be destroyed. This feature makes proactive recovery approaches unable to effectively deal with large-scale failures. Second, no individual router has the overall information of failures, and each router only knows whether its neighbors are reachable. For an unreachable neighbor, the router cannot differentiate whether the neighbor fails or the link connecting the neighbor fails. In Fig. 1, the router v_{11} finds that its neighbor v_4, v_6, and v_{10} are unreachable, but it does not know that v_{10} fails while v_4 and v_6 are live. Moreover, v_{11} cannot see that v_{10} is unreachable from v_5.

We propose an approach called Reactive Two-phase Rerouting (RTR) for intra-domain routing to quickly recover from large-scale failures with the shortest recovery paths. The basic idea is to collect failure information, and then calculate valid recovery paths and forward packets over them. To collect failure information, we design a protocol to forward packets around the failure area and record failure information in the packet header. This is a very efficient

This work was supported in part by the Defense Threat Reduction Agency under grant HDTRA1-10-1-0085.

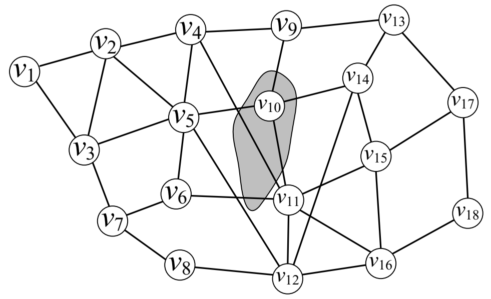


Figure 1. An example of large-scale failures. The failure area is denoted by shaded area.

process that allows us to fully characterize the failed region quickly. After gathering failure information, the second phase of RTR is invoked to calculate new shortest paths to bypass the failure area, and then forward packets along the new shortest paths through source routing.

RTR can deal with large-scale failures associated with an area of any shape and location and is free of permanent loops. Under a single link failure, RTR guarantees to recover all failed routing paths with the shortest recovery paths. For any failure area, the recovery paths provided by RTR are guaranteed to be the shortest. Experimental results show that RTR can find the shortest recovery paths for more than 98.6% of the failed routing paths with reachable destinations. Compared with prior works, RTR can find more shortest recovery paths with lower computational and transmission overhead. For the failed routing path with unreachable destinations, RTR can quickly identify them to save network resources. Compared with prior works, RTR can save 83.1% of computation and 75.6% of transmission for irrecoverable failed routing paths.

The rest of this paper is organized as follows. We describe the system model and our problem in Section II. In Section III, we present the design and properties of RTR. Section IV evaluates the performance of RTR. Section V reviews related work. Finally, Section VI concludes the paper.

II. Preliminaries

In this section, we first present the network and failure models, briefly introduce the recovery process, and then describe the design objectives.

A. Network Model and Failure Model

We model the network under study as an undirected graph with a set of nodes (routers) and a set of edges (links). The link from node v_i to node v_j is denoted by $e_{i,j}$, which has a cost $c_{i,j}$. Links can be asymmetric, i.e., $c_{i,j} \neq c_{j,i}$. We focus on intra-domain routing with a link-state protocol like OSPF and IS-IS. The routing path $p_{i,j}$ from node v_i to node v_j is the shortest path calculated based on the link cost. Its length is the sum of the costs of links along $p_{i,j}$.

In intra-domain routing, every node knows the network topology. We assume that routers have a consistent view of the network topology before large-scale failures occur. In addition, each node knows the coordinates of all nodes. RTR uses this information to assist in collecting failure information. This assumption is reasonable for the following reasons. First, in intra-domain routing, the administrator of an AS knows the coordinates of all routers in the AS and can record this information at each router. Second, we aim at providing recovery for high speed IP networks. Routers in such networks are immobile and are rarely moved after being deployed. Third, RTR does not require highly accurate coordinates information of routers and thus does not need special location devices like GPS. When the network administrator moves a router (although this rarely happens in high speed IP networks), the coordinates information stored at routers can also be updated.

We do not modify the mechanisms for failure detection and topology update dissemination in the current IP networks. A router only knows whether its neighbors are reachable, but cannot differentiate between a node failure and a link failure. To prevent route flapping, routers do not immediately disseminate topology updates upon detecting a failure. Thus, routers cannot quickly have the overall information of failures.

To be more practical, we do not make any assumption on the shape and location of the failure area. Similar to prior work [16], the failure area is modeled as a continuous area in the network. Routers within it and links across it all fail. A routing path fails if it contains a failed node or link. A node is *adjacent to* the failure area if it has a link across the failure area.

B. Recovery Process Overview

RTR is for rerouting traffic affected by failures during IGP convergence. When a router detects that the default next hop towards a destination is unreachable, the router invokes RTR to recover the routing path towards the destination. After IGP convergence finishes, every live router has a reachable default next hop for each destination in the routing table. RTR is not invoked and the link-state protocol is responsible for routing. Therefore, RTR only operates during IGP convergence for traffic whose routing paths fail. In Fig. 1, the routing path from v_7 to v_{17} is $v_7 \rightarrow v_6 \rightarrow v_{11} \rightarrow v_{15} \rightarrow v_{17}$ which is disconnected at $e_{6,11}$. When v_6 detects that v_{11} is unreachable, it invokes RTR to reroute traffic towards v_{17}, until IGP convergence completes.

C. Objectives

Large-scale failures may destroy a large number of routing paths. There are two cases if the source of a failed routing path is live. In the first case, the destination is still reachable from the source. For this case, RTR should find a recovery path as short as possible to reach the destination. This is the goal of existing recovery approaches. In the second case, the destination is unreachable from the source, because it fails or is in a different partition from the source. Packets towards an unreachable destination should be discarded as early as possible. However, this case has not been studied in any prior work. A possible reason is that most existing works focus on sporadic link failures where it is unlikely that there are many unreachable destinations, and hence discarding packets late has little negative impact. Large-scale failures may make many destinations unreachable, especially when the network is partitioned by failures. Keeping forwarding packets with unreachable destinations wastes network resources and degrades network performance.

Due to searching on-demand for usable routing paths, reactive recovery unavoidably introduces computational overhead to routers. Routers in high speed IP networks need to forward packets very quickly. Using too much CPU time for recovery computation interferes with forwarding the packets which are not affected by failures. Therefore, a reactive recovery mechanism should have low computational overhead and have as low transmission overhead as possible to save bandwidth.

III. RTR: Reactive Two-phase Rerouting

This section describes the design and properties of RTR.

A. Overview

RTR consists of two phases. The first phase is to collect failure information, i.e., identifying which links have failed. In this phase, packets are forwarded around the failure area, and routers adjacent to the failure area record failed links in the packet header. Then, the second phase calculates a new shortest path based on the collected failure information, and uses source routing to ensure that packets are forwarded along the new shortest path.

In Fig. 2, suppose v_7 is the source and v_{17} is the destination. The routing path $v_7 \rightarrow v_6 \rightarrow v_{11} \rightarrow v_{15} \rightarrow v_{17}$ is disconnected at $e_{6,11}$, and thus v_6 invokes RTR for recovery. Node v_6 is referred to as a *recovery initiator*, and a *recovery path* is from the recovery initiator to the destination. In the first phase, starting from v_6, packets towards v_{17} are forwarded around the failure area along the dotted lines. When v_5 receives a packet, it inserts the id of $e_{5,10}$ in the packet header. Similarly, v_9 inserts the id of $e_{9,10}$ in the packet header. After packets return to v_6, v_6 knows that five links have failed by checking the packet header. Then, in the second phase v_6 removes the failed links from the network topology and calculate a new shortest path to reach v_{17}, i.e., the dashed lines. When v_6 receives a packet for v_{17}, it forwards it along the recovery path through source routing. If a node is adjacent to the failure area and its default next hop towards a destination is unreachable, the node is a recovery initiator and it invokes RTR to recover the failed routing path. Multiple recovery initiators independently invoke RTR to recover their failed routing paths.

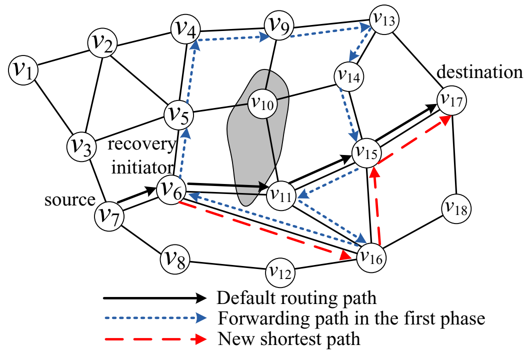


Figure 2. The recovery process of RTR on a planar graph.

We use data packets rather than designing a special control message to collect failure information due to two reasons. First, designing a special control message complicates the control plane of routers. Second, if a special control message is used, v_6 has to discard packets towards v_{17} before finishing collection of failure information, because v_6 may not have enough buffer space to store these packets. In RTR, the delay of the packets forwarded to gather failure information is increased, but no packet is dropped. As simulations in Section IV show, the first phase is quite short, and hence the increased delay is small.

The first phase of RTR needs to run only once at a recovery initiator and can benefit all destinations, because failure information can be used for calculating recovery paths for different destinations. The major challenge is to forward packets around the failure area and then back to the recovery initiator.

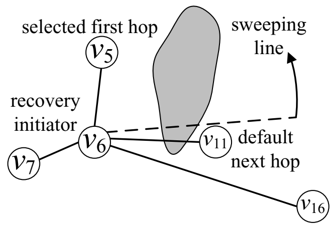
B. Collecting Failure Information on Planar Graphs

To facilitate exposition, we first explain how to collect failure information on planar graphs with no cross links, and then extend the method to a general graph in the next subsection. Starting from the recovery initiator, we use right-hand rule to select a live neighbor to forward packets. Suppose v_i is the recovery initiator whose default next hop v_j is unreachable, there are two cases.

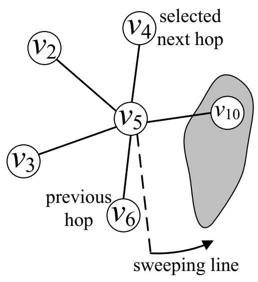
1) To select the first hop, v_i takes link $e_{i,j}$ as the sweeping line and rotates it counterclockwise, until it reaches a live neighbor. Then, v_i uses this live neighbor as the first hop.
2) In the other cases, suppose v_m receives a packet from v_n, v_m takes link $e_{m,n}$ as the sweeping line and rotates it counterclockwise, until reaching a live neighbor. Then, v_m takes this live neighbor as the next hop.

In Fig. 2, the recovery initiator v_6 selects v_5 as the first hop as shown in Fig. 3(a), and v_5 chooses v_4 as the next hop as shown in Fig. 3(b).

With the above rule, packets are forwarded out from the recovery initiator v_i to visit every node that is adjacent to the failure area and reachable from v_i, and finally back to v_i, no



(a) Select the first hop.



(b) In the other case.

Figure 3. The rule of selecting a live neighbor to forward packets in the first phase.

matter the failure area is inside the network or on the border of the network [17]. To control the forwarding process and collect failure information, we add the following fields in the packet header. The link id is represented by 16 bits.

- mode: When it is 0, the packet is forwarded by the default routing protocol. When it is 1, the packet is forwarded by the first phase of RTR.
- rec_init: The id of the recovery initiator.
- failed_link: It records the ids of failed links.

In the first phase, nodes take the following actions.

1) Initially, the recovery initiator v_i sets mode in the packet header to 1 and set rec_init to v_i. Then, v_i selects the first hop and forwards the packet.
2) When node v_j receives a packet, for each unreachable neighbor v_k other than the recovery initiator v_i, v_j inserts the id of link $e_{j,k}$ in failed_link in the packet header if $e_{j,k}$ is not already recorded. Finally, v_j selects the next hop and forwards the packet.
3) When the recovery initiator v_i receives a packet whose rec_init is v_i, it selects the next hop. If the next hop is the same as the first hop, it means that the packet has traveled the whole cycle around the failure area, and hence the first phase of RTR completes. Otherwise, v_i forwards the packet to the next hop to prevent missing some nodes on the cycle.

To reduce the transmission overhead, a failed link is not recorded in failed_link if v_i is one end of the link, because v_i knows this failure. After the first phase finishes, v_i removes the links in failed_link and its links to unreachable neighbors from its view of network topology. Then, v_i calculates the shortest path to the destination and forwards packets with source routing.

Fig. 2 shows the recovery process of RTR on a planar graph where v_6 is the recovery initiator and v_{17} is the destination. Along the forwarding path in the first phase, failed_link in the packet header records four links $e_{5,10}$, $e_{9,10}$, $e_{14,10}$, and $e_{11,10}$. These links together with $e_{6,11}$ are removed from the network topology when v_6 calculates the shortest path to v_{17}. The new shortest path shown by the dashed lines has 3 hops.

C. Collecting Failure Information on General Graphs

The forwarding rule introduced above works well on planar graphs, but it may fail to forward packets around the failure area on a general graph as shown in Fig. 4. The cycle formed by the dotted lines fails to enclose the failure area. At v_5, the forwarding rule selects v_{12} as the next hop, and thus the dashed cycle fails to enclose the failure area.

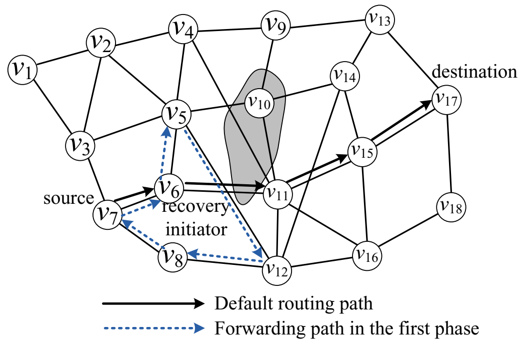


Figure 4. The forwarding rule may fail on a general graph.

This phenomenon looks similar to the permanent loops in geographic routing in wireless networks [18]. However, they are different problems and the solutions in [18] cannot be used to solve our problem. The permanent loop problem in geographic routing refers to the fact that the forwarding rule fails to jump out of local minimum and thus the forwarding path contains a permanent loop. In our problem, the forwarding path fails to enclose the failure area. Unlike geographic routing, we cannot remove some cross links to planarize the topology in advance, because it may incorrectly partition the network when failures occur. For example, the planarization technique may turn the topology in Fig. 4 into a planar graph by removing $e_{6,11}$, $e_{4,11}$ and $e_{12,14}$. The planarized topology is partitioned when $e_{6,5}$ and $e_{6,7}$ fail, but in fact the network is still connected.

We observe that the forwarding rule fails on a general graph because the forwarding path crosses the links between the recovery initiator and its unreachable neighbors. Intuitively, if the forwarding path surrounds the failure area, packets are forwarded clockwise around the failure area as shown in Fig. 2. However, when the forwarding path crosses the links between the recovery initiator and its unreachable neighbors, packets are possibly forwarded counterclockwise around the failure area. As a result, the forwarding path fails to enclose the failure area. To solve this problem, we propose the following constraint on the forwarding path.

Constraint 1: The forwarding path should not cross the links between the recovery initiator and its unreachable neighbors.

In Fig. 4, the forwarding disorder happens at v_5 when it selects v_{12} as the next hop. By Constraint 1, link $e_{6,11}$ prevents $e_{5,12}$ from being selected, and thus v_5 chooses v_4 as the next hop. In this way, packets are forwarded round the failure area.

In addition to the above type of forwarding disorder, another forwarding disorder may also cause problems. As shown in Fig. 5, packets traverse a link in each direction, making the forwarding path quite long. When receiving a packet from v_2, v_3 chooses v_4 as the next hop. The selected link $e_{3,4}$ crosses link $e_{1,2}$ that is traversed before by the forwarding path. This disorder causes the packet to be forwarded counterclockwise around the failure area. If $e_{3,4}$ is excluded, v_3 chooses the correct next hop v_5 and the forwarding path becomes $v_1 \to v_2 \to v_3 \to v_5 \to v_6 \to v_1$. Therefore, we have the second constraint on the forwarding path.

Constraint 2: The forwarding path should not contain cross links.

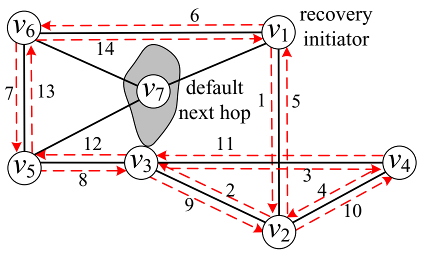


Figure 5. The forwarding disorder that leads to a long forwarding path. The numbers beside the dotted lines show the visiting order.

To prevent these two types of forwarding disorders, we add the `cross_link` field in the packet header to record some links. A link across any link in `cross_link` is excluded from being selected to forward packets. For each link, routers precompute the set of links across it. The action taken by each node in the first phase is as follows.

1) Initially, the recovery initiator v_i sets `mode` in the packet header to 1 and set `rec_init` to v_i. For each unreachable neighbor v_j, if link $e_{i,j}$ crosses other links, v_i inserts the id of $e_{i,j}$ in `cross_link`. Then, v_i selects the first hop and forwards the packet.
2) When node v_j receives a packet, for each unreachable neighbor v_k other than the recovery initiator v_i, v_j inserts the id of link $e_{j,k}$ in `failed_link` in the packet header if $e_{j,k}$ is not recorded. Then, v_j selects node v_m as the next hop, where link $e_{j,m}$ is not excluded by the links in `cross_link`. If there is a link which is across $e_{j,m}$ but not excluded by the links in `cross_link`, v_j inserts the id of $e_{j,m}$ in `cross_link`. Finally, v_j forwards the packet to v_m. (Node v_j is always able to find the next hop, because its previous hop satisfies the requirement.)
3) When the recovery initiator v_i receives a packet whose `rec_init` is v_i, v_i selects the next hop. If it is the same as the first hop, the first phase of RTR ends. Otherwise, v_i forwards the packet to the next hop.

The links recorded in `cross_link` by the recovery initiator are used for preventing the first type of forwarding disorder. All the other links in `cross_link` are used for preventing the second type of forwarding disorder.

Finally, we use an example in Fig. 6 to show how RTR works on a general graph. The content of `failed_link` and `cross_link` at each hop is shown in Table I. At v_5, $e_{6,11}$ prevents $e_{5,12}$ from being selected. Similarly, at v_{11}, $e_{14,12}$ blocks $e_{11,15}$ and $e_{11,16}$. The forwarding path in the first phase in shown by the dotted lines. After collecting failure information, v_6 computes a new shortest path towards v_{17} as shown by the dashed lines.

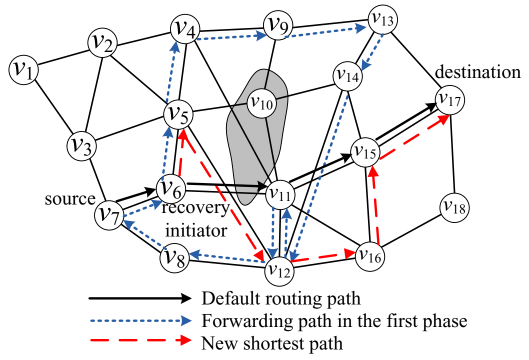


Figure 6. The recovery process of RTR on a general graph.

Table I
THE CONTENT OF THE TWO FIELDS AT EACH HOP

Hop count	**At node**	**`failed_link`**	**`cross_link`**
0	v_6		$e_{6,11}$
1	v_5	$e_{5,10}$	$e_{6,11}$
2	v_4	$e_{5,10}$, $e_{4,11}$	$e_{6,11}$
3–4	v_9, v_{13}	$e_{5,10}$, $e_{4,11}$, $e_{9,10}$	$e_{6,11}$
5–6	v_{14}, v_{12}	$e_{5,10}$, $e_{4,11}$, $e_{9,10}$, $e_{14,10}$	$e_{6,11}$, $e_{14,12}$
7–11	v_{11}, v_{12}, v_8, v_7, v_6	$e_{5,10}$, $e_{4,11}$, $e_{9,10}$, $e_{14,10}$, $e_{11,10}$	$e_{6,11}$, $e_{14,12}$

It is possible that the first phase omits some failures. For example, if there is a link between v_{10} and v_{15} in Fig. 6, this failed link is omitted. Recording all failed links requires visiting every node that is adjacent to the failure area and reachable from the recovery initiator. This usually leads to a much longer forwarding path and a more complex forwarding rule than the current RTR design. Although the first phase does not guarantee to collect all failures, we prove in Section III-E that RTR can guarantee to find the shortest recovery path to bypass any single link failure as proactive recovery approaches can do. Furthermore, simulations in Section IV show that RTR can find the shortest recovery paths for more than 98.6% of failed routing paths with reachable destinations.

D. Recomputation and Rerouting

In the second phase, RTR adopts incremental recomputation [19] to calculate the shortest path from the recovery initiator to the destination, which can be achieved within a few milliseconds even for graphs with a thousand nodes [20].

By caching the recovery paths, the recovery initiator needs to calculate the shortest path only once for each destination affected by failures.

After calculating the shortest path, RTR forwards packets along it through source routing, which is widely used in prior works to improve Internet resilience to failures [21]–[24]. The recovery initiator inserts the entire shortest path in the packet header. Routers along the shortest path simply forward packets based on the source route in the packet header. Since RTR may omit few failures in the first phase, the recovery path possibly contains a failure. In that case, RTR simply discards the packet.

E. Properties

This subsection summarizes three major properties of RTR.

Theorem 1: RTR is free of permanent loops.

Proof: In the first phase, permanent loops happen when packets cannot return to the recovery initiator. Suppose the forwarding path from the recovery initiator v_i to another node v_{i+k} is $v_i \rightarrow v_{i+1} \rightarrow \cdots \rightarrow v_{i+k}$ consisting of k links. These links do not cross the links between v_i and its unreachable neighbors; otherwise they are excluded from being selected. Moreover, any link across them will not be selected later. It means that these k links will not be excluded from being chosen again. Therefore, in the worst case, packets can be forwarded back to v_i along $v_{i+k} \rightarrow v_{i+k-1} \rightarrow \cdots \rightarrow v_i$. The second phase is free of permanent loops because it forwards packets along the shortest path. ■

Next we show the recovery performance and ability.

Theorem 2: For any failure area, the recovery paths provided by RTR are guaranteed to be the shortest.

Proof: Let G denote the topology before failures occur. Suppose E_1 is the set of failed links collected by the first phase of RTR, and E_2 is the actual set of failed links (ground truth). We have $E_1 \subseteq E_2$, because RTR may miss failed links but does not label live links as failed. Suppose the routing path from node s to node t is disconnected, and node i is the recovery initiator. Let $p^1_{i,t}$ be the shortest path from node i to node t in $G - E_1$, and $p^2_{i,t}$ be that in $G - E_2$. If RTR successfully recovers the failed routing path from node i to node t, it means that $p^1_{i,t}$ has no failure. Hence, $p^1_{i,t}$ is also a path in $G - E_2$. Since $p^2_{i,t}$ cannot be shorter than $p^1_{i,t}$, $p^1_{i,t}$ is the shortest path from i to t in $G - E_2$. In conclusion, $p^1_{i,t}$ is the shortest recovery path. ■

Theorem 3: For a single link failure, RTR guarantees to recover all failed routing paths with the shortest recovery paths.

Proof: Theorem 2 shows that the recovery path is the shortest if it does not contain failures. Under any single link failure, the recovery initiator knows the failed link and hence the recovery path does not contain it. Therefore, RTR guarantees to provide the shortest recovery paths. ■

Although we focus on a single failure area, RTR also works for multiple failure areas. Upon encountering a failure area F_1, RTR collects failure information of F_1 and computes a backup path to bypass F_1. In order to avoid routing loops, the packet header needs to carry failure information of F_1. When it encounters another failure area F_2, the recovery initiator removes all failed links recorded in the packet header. Through it, the computed recovery path can bypass both F_1 and F_2. To reduce the packet header overhead, we can use the mapping technique in [22] to reduce storage.

IV. Performance Evaluation

In this section, we evaluate the performance of RTR and compare it with other recovery approaches [13], [22].

A. Simulation Setup

The simulation is based on eight ISP topologies derived from the Rocketfuel project [25], which are summarized in Table II. For each topology, we randomly place nodes in a 2000×2000 area. All topologies adopt shortest path routing based on hop count. To simplify our simulation, the failure area is a circle randomly placed in the 2000×2000 area with a radius randomly selected between 100 and 300. The radius and location of the circular area are unknown to RTR. Nodes within the circle and links across the circle have all failed. There are three cases for a failed routing path.

1) The source fails and it cannot send packets. We ignore this type of failed routing paths in the simulation.
2) The source is live and the destination is still reachable from the source. This is a *recoverable* failed routing path.
3) The source is live but the destination is unreachable from the source. This is an *irrecoverable* failed routing path.

For a failed routing path with a live source, the recovery process is invoked at the recovery initiator. Some failed routing paths with the same destination may have the same recovery initiator. Their recovery processes are the same; thus we take them as one test case. Given a topology, a test case is determined by three factors, i.e., the recovery initiator, the destination, and the failure area. For each topology, we generate failure areas to collect 10,000 recoverable test cases and 10,000 irrecoverable test cases, and evaluate the performance.

We compare RTR with a proactive recovery approach MRC [13] and a reactive recovery approach FCP [22]. For FCP, we use the source routing version, which reduces the computational overhead of the original FCP.

B. Duration of the First Phase

The duration of the first phase of RTR affects the delay of packets forwarded during the first phase. RTR has the same first phase in both recoverable and irrecoverable test cases. The one-hop delay consists of the delay at a router and

Table II
SUMMARY OF TOPOLOGIES USED IN SIMULATION

Topology	AS209	AS701	AS1239	AS3320	AS3549	AS3561	AS4323	AS7018
# Nodes	58	83	52	70	61	92	51	115
# Links	108	219	84	355	486	329	161	148

the propagation delay on a link. We use 100 microseconds as the delay at a router, because packets in the first phase introduce quite low computational overhead to routers and 99% of packets in IP networks with OC-12 rate (622 Mb/s) experience less than 100 microseconds delay through a router [26]. The propagation delay on a link is about 1.7 milliseconds, assuming that links are 500 kilometers long on average. Hence, the one-hop delay is 1.8 milliseconds.

The cumulative distribution of the duration of the first phase is shown in Fig. 7. The first phase of RTR is quite short. None of the $10 \times 20,000 = 200,000$ test cases has the first phase longer than 110 milliseconds. In every topology, the first phase is shorter than 75 milliseconds in more than 90% of test cases. The duration in AS7018 is longer than that in the other topologies mainly because this topology has many tree branches. Each link on a tree branch may be traversed twice. Generally speaking, the first phase of RTR is quite short if the topology has high connectivity and few free branches like the Internet backbone.

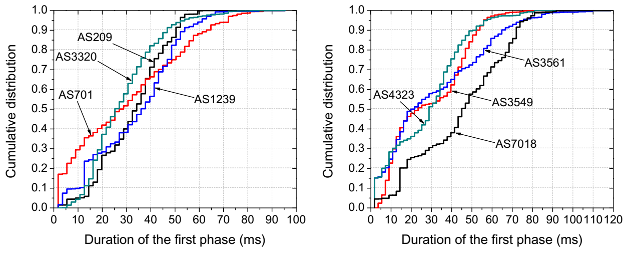


Figure 7. The cumulative distribution of duration of the first phase.

C. Recoverable Test Cases

We evaluate the performance with 10,000 test cases from the following four aspects.

- *Recovery rate*: It is defined as the percentage of successfully recovered test cases. We also measure the optimal recovery rate, i.e., the percentage of successfully recovered test cases with the shortest recovery paths.
- *Stretch*: For a pair of source and destination, the stretch is the ratio of the recovery path length to the length of the shortest path. The optimal value is 1.
- *Computational overhead*: RTR and FCP need to calculate the shortest paths on-demand. The computational overhead is defined as the number of shortest path calculations.
- *Transmission overhead*: RTR and FCP use the packet header to record necessary information for recovery. The transmission overhead is defined as the number of bytes used for recording information.

Table III summarizes the recovery rate, optimal recovery rate, maximum stretch, and maximum computational overhead. The recovery rate of RTR is higher than 97.7% in every topology. MRC cannot effectively deal with large-scale failures, because a routing path and its backup paths may fail simultaneously. Hence, we only compare RTR with FCP in the following part. Theorem 2 shows that if a failed routing path is successfully recovered by RTR, the recovery path is the shortest. Simulation results confirm this property and show that the recovery rate and optimal recovery rate of RTR are the same. RTR achieves higher optimal recovery rate than FCP in every topology. The average optimal recovery rate of RTR is 98.6% in all 100,000 test cases.

Next, we compare the stretch of recovery paths provided by RTR and FCP. Fig. 8 shows the cumulative distribution of the stretch of successfully recovered paths. RTR guarantees to have optimal stretch 1. FCP achieves small stretch in most test cases, but is still worse than RTR in every topology. As shown in Table III, FCP may have quite a large stretch, which is even worse than MRC.

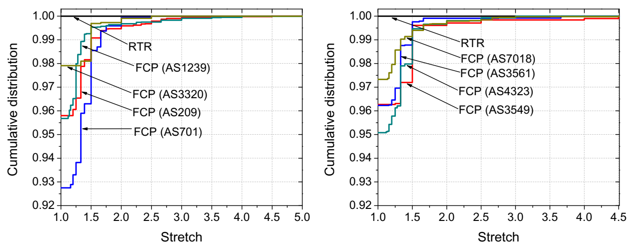


Figure 8. The cumulative distribution of stretch of recovery paths.

Next, we compare the computational overhead of RTR and FCP. Fig. 9 shows the cumulative distribution of the number of shortest path calculations. In each test case, RTR calculates the shortest path only once, regardless of whether or not it is successfully recovered. FCP uses many more calculations than RTR, because FCP calculates the shortest path whenever the packet encounters a failure not recorded in the packet header.

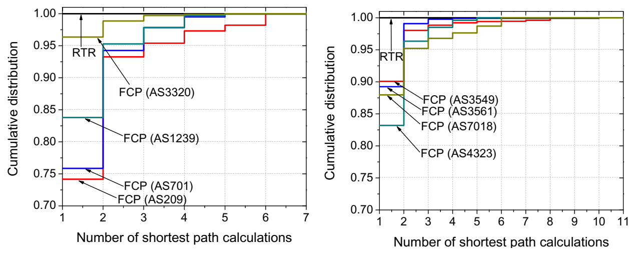


Figure 9. The cumulative distribution of computational overhead in recoverable test cases.

Table III
SUMMARY OF PERFORMANCE OF RTR, FCP, AND MRC IN RECOVERABLE TEST CASES

	Recovery rate (%)			**Optimal recovery rate (%)**			**Max stretch**			**Max computational overhead**	
	RTR	FCP	MRC	RTR	FCP	MRC	RTR	FCP	MRC	RTR	FCP
AS209	98.2	100	36.8	98.2	95.8	13.9	1	4.5	2.5	1	7
AS701	98.5	100	35.7	98.5	92.8	24.2	1	2.5	2.5	1	6
AS1239	97.7	100	15.5	97.7	95.7	8.2	1	5.0	2.0	1	5
AS3320	99.2	100	63.1	99.2	97.9	42.1	1	2.5	3.0	1	5
AS3549	98.9	100	63.9	98.9	96.3	39.5	1	4.5	2.0	1	9
AS3561	98.8	100	63.9	98.8	96.2	34.0	1	3.7	2.5	1	10
AS4323	99.2	100	33.1	99.2	95.1	25.9	1	3.0	2.0	1	5
AS7018	98.4	100	25.6	98.4	97.3	19.1	1	4.5	1.8	1	6
Overall	98.6	100	42.2	98.6	95.9	25.9	1	5.0	3.0	1	10

Finally, we compare the transmission overhead of RTR and FCP. In the first phase, the transmission overhead of RTR is caused by recording the id of some links. In the second phase, it is due to recording the source route. In FCP, the transmission overhead is from recording the id of encountered failed links and the source route. For each topology, we measure the average transmission overhead across all 10,000 test cases, until IGP convergence finishes. To clearly demonstrate it, we only show the result over the first second in Fig. 10. In RTR, the transmission overhead in the first phase is higher than that in the second phase. With the simulation going on, many test cases of RTR enter the second phase, and the transmission overhead quickly decreases. After about 100 milliseconds, all test cases finish the first phase and the transmission overhead converges to a fixed value, which is smaller than that of FCP in every topology.

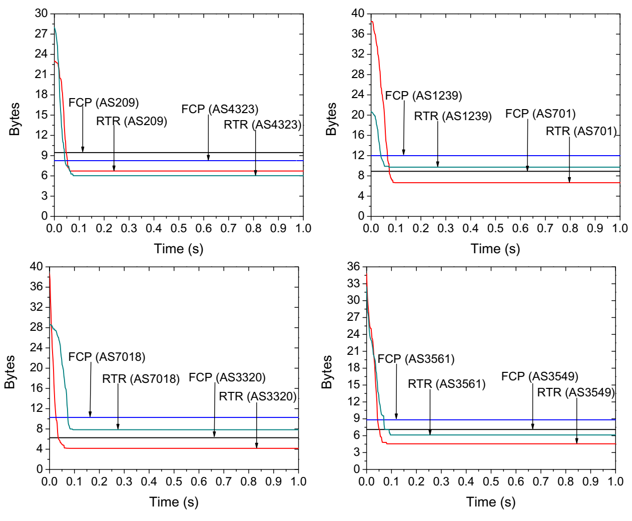


Figure 10. The average transmission overhead of RTR and FCP on recoverable test cases.

In summary, the optimal recovery rate of RTR on all 100,000 test cases is as high as 98.6%. Compared with FCP, RTR achieves higher optimal recovery rate with lower computational and transmission overhead.

D. Irrecoverable Test Cases

First, we investigate the percentage of failed routing paths that are irrecoverable. The radius of the failure area increases from 20 to 300 in increments of 20. For each radius, we generate 1,000 failure areas and count failed routing paths. The percentage of irrecoverable routing paths is shown in Fig. 11. Even when the radius is 20 (the failure area takes up about 0.03% of the simulation area), more than 20% of the failed paths are irrecoverable in nine topologies. When the radius is 300 (the failure area takes up about 7.07% of the simulation area), more than 45% of the failed routing paths are irrecoverable in nine topologies. The simulation result illustrates that large-scale failures make many failed routing paths irrecoverable. As a result, the performance of a recovery approach on irrecoverable routing paths is also very important.

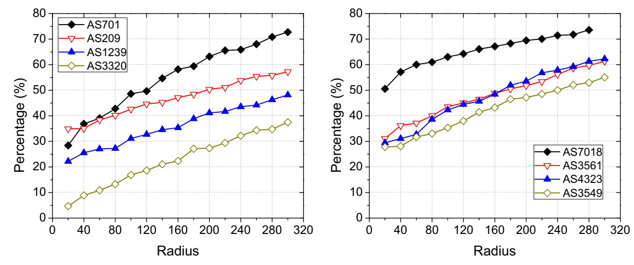


Figure 11. The percentage of failed routing paths that are irrecoverable under the failure area of different radii.

For an irrecoverable failed routing path, all efforts for recovery are wasted, because packets are ultimately discarded. For each topology, we use 10,000 irrecoverable test cases to evaluate the performance of RTR and compare it with FCP in the following two aspects.

- *Wasted computation*: It is defined as the number of shortest path calculations.
- *Wasted transmission*: We assume that the packet size is 1,000 bytes plus the bytes in the packet header used for recovery. Let s denote this value and h denote the hops from the recovery initiator to the node discarding the packet. Thus, the wasted transmission of the packet is defined as $s \times h$.

Fig. 12 shows the cumulative distribution of wasted computation. The wasted computation of RTR is 1 because it needs to calculate the shortest path only once. The wasted computation of FCP is much more than that of RTR. For example, in AS3549 FCP has more than 10 calculations of the shortest path in about 80% test cases.

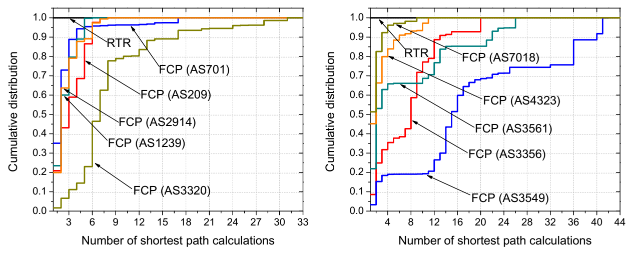


Figure 12. The cumulative distribution of the wasted computation in irrecoverable test cases.

Next, we compare the wasted transmission of RTR with FCP and show the result in Fig. 13. As was the case with the wasted computation, RTR outperforms FCP in every topology on this metric as well. In particular, the wasted transmission of FCP is much more than that of RTR in AS3320, AS3549 and AS3561.

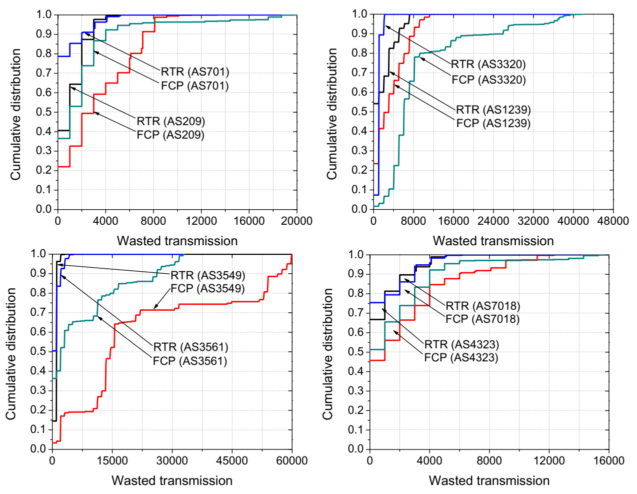


Figure 13. The cumulative distribution of the wasted transmission on irrecoverable test cases.

Finally, Table IV summarizes the wasted computation and wasted transmission of RTR and FCP, including the average and worst result. Compared with FCP, RTR reduces wasted computation by 83.1% and wasted transmission by 75.6%. FCP does not work well because it has to try every possible link to reach the destination before discarding packets. As a result, FCP calculates a recovery path many times and forwards packets for many hops.

V. Related Work

There have been many approaches for intra-domain routing to recover from failures. A straightforward method is to accelerate the IGP convergence via appropriately tuning the parameters related to failure detection, topology update dissemination, and routing path recomputation [10]. However, rapidly triggering the IGP convergence may cause route flapping and make the network more unstable [11]. Proactive recovery methods based on precomputed backup routes, such as [13]–[15], can effectively handle sporadic link failures with low overhead. However, it is hard to apply them to large-scale failures, because a routing path and the corresponding backup routes may fail together when large-scale failures occur.

The first phase of the proposed RTR looks like geographic routing in wireless networks [18], [27], but they are quite different. The only similarity is that both of them use the right-hand rule to select a node to forward packets. The first phase of RTR aims at forwarding packets around the failure area, while geographic routing in wireless networks tries to greedily forward packets to the destination. Additionally, as explained in Section III-C the forwarding disorder problem in RTR is distinct from the permanent loop problem in geographic routing. The solutions to the latter problem cannot solve the forwarding disorder problem in RTR.

There are some works related to large-scale failures in IP networks. Zlatokrilov *et al.* [23] proposed an approach to find paths to bypass certain failure areas in the network using distance-vector protocols. However, their method relies on the assumption that the failure area is a circle whose center and radius are known in advance. Moreover, their approach assumes that a router can query routing information from other routers. Under large-scale failures, however, it is a challenge to deliver query requests to other routers. Neumayer *et al.* [2], [3] analyzed the impact of large-scale failures on network routing, but they did not propose a recovery method. Further, they model the failure area as a circle, while we do not have any assumption on the shape of the failure area. Zheng *et al.* [12], [28] proposed an approach to identify links with abnormal performance, but they did not address how to recover the abnormal links.

Similar to RTR, the proactive recovery approach FCP [22] uses the packet header to collect failure information and calculates recovery paths on-demand. However, FCP needs to modify the mechanism of topology update dissemination in the current IP networks. Moreover, as shown in the simulation, RTR outperforms FCP in both recoverable and irrecoverable test cases. R3 [15] uses multiple backup paths to recover networks from multiple link failures. However, routers are assumed to broadcast failure information immediately after detecting a failure; whereas RTR does not make this assumption.

VI. Conclusions

In this paper, we proposed a Reactive Two-phase Rerouting (RTR) approach for intra-domain routing to quickly recover from large-scale failures with the shortest recovery paths. To be more practical, we model large-scale failures as occurring over a continuous area of arbitrary shape

Table IV
SUMMARY OF THE WASTED COMPUTATION AND WASTED TRANSMISSION OF RTR AND FCP IN IRRECOVERABLE TEST CASES

	Wasted computation				**Wasted transmission**			
	Avg RTR	Avg FCP	Max RTR	Max FCP	Avg RTR	Avg FCP	Max RTR	Max FCP
AS209	1	3.2	1	9	1125.0	3366.4	7098	13364
AS701	1	2.5	1	17	511.7	2049.5	6072	19760
AS1239	1	2.4	1	7	1639.3	2049.5	18720	19760
AS3320	1	8.4	1	31	1071.6	3798.6	7098	20560
AS3549	1	18.8	1	42	929.2	8133.9	7126	30334
AS3561	1	7.0	1	27	790.0	7381.8	13364	32860
AS4323	1	2.7	1	11	716.1	2185.8	7098	12336
AS7018	1	1.9	1	10	676.7	1616.6	6096	18468
Overall	1	5.9	1	42	932.5	3822.8	18720	32860

and at arbitrary location. We prove that RTR is free of permanent loops. Under a single link failure, RTR guarantees to recover all failed routing paths with the shortest recovery paths. Moreover, for any failure area, the recovery paths provided by RTR are guaranteed to be the shortest. Extensive simulations based on ISP topologies show that RTR can find the shortest recovery paths for more than 98.6% of failed routing paths with reachable destinations. Compared with prior works, RTR can find more shortest recovery paths with lower computational and transmission overhead. For failed routing paths with unreachable destinations, RTR can quickly identify them and thus save network resources. Compared with prior works, RTR can save 83.1% of computation and 75.6% of transmission for irrecoverable failed routing paths.

REFERENCES

[1] G. Iannaccone, C. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, “Analysis of link failures in an IP backbone,” in *ACM SIGCOMM Workshop on Internet measurment*, 2002.

[2] S. Neumayer, G. Zussman, R. Cohen, and E. Modiano, “Assessing the vulnerability of the fiber infrastructure to disasters,” in *IEEE INFOCOM*, 2009.

[3] S. Neumayer and E. Modiano, “Network reliability with geographically correlated failures,” in *IEEE INFOCOM*, 2010.

[4] J. Cowie, A. Popescu, and T. Underwood, “Impact of Hurricane Katrina on Internet infrastructure,” http://www.renesys.com/tech/presentations/pdf/Renesys-Katrina-Report-9sep2005.pdf, 2005.

[5] S. LaPerriere, “Taiwan earthquake fiber cuts: a service provider view,” http://www.nanog.org/meetings/nanog39/presentations/laperriere.pdf, 2007.

[6] Y. Ran, “Considerations and suggestions on improvement of communication network disaster countermeasures after the wenchuan earthquake,” *IEEE Communications Magazine*, vol. 49, no. 1, pp. 44–47, 2011.

[7] A. Ogielski and J. Cowie, “Internet routing behavior on 9/11 and in the following weeks,” http://www.renesys.com/tech/presentations/pdf/renesys-030502-NRC-911.pdf, 2002.

[8] S. M. Bellovin and E. R. Gansner, “Using link cuts to attack Internet routing,” AT&T Labs Research, Tech. Rep., 2003.

[9] C. Wilson, “High altitude electromagnetic pulse (HEMP) and high power microwave (HPM) devices: threat assessments,” http://www.fas.org/man/crs/RL32544.pdf, 2004.

[10] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure, “Achieving sub-second IGP convergence in large IP networks,” *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 3, 2005.

[11] A. Basu and J. Riecke, “Stability issues in OSPF routing,” in *ACM SIGCOMM*, 2001.

[12] Q. Zheng and G. Cao, “Minimizing probing cost and achieving identifiability in probe based network link monitoring,” *IEEE Transactions on Computers*, to appear.

[13] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, “Fast IP network recovery using multiple routing configurations,” in *IEEE INFOCOM*, 2006.

[14] S. Kini, S. Ramasubramanian, A. Kvalbein, and A. F. Hansen, “Fast recovery from dual link failures in IP networks,” in *IEEE INFOCOM*, 2009.

[15] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang, “R3: Resilient routing reconfiguration,” in *ACM SIGCOMM*, 2010.

[16] Q. Zheng, G. Cao, T. L. Porta, and A. Swami, “Detecting and localizing large-scale router failures using active probes,” in *IEEE MILCOM*, 2011.

[17] Q. Fang, J. Gao, and L. J. Guibas, “Locating and bypassing routing holes in sensor networks,” in *IEEE INFOCOM*, 2004.

[18] B. Karp and H. T. Kung, “GPSR: Greedy perimeter stateless routing for wireless networks,” in *ACM MOBICOM*, 2000.

[19] P. Narvaez, K.-Y. Siu, and H. yi Tzeng, “New dynamic algorithms for shortest path tree computation,” *IEEE/ACM Transactions on Networking*, vol. 8, no. 6, 2000.

[20] C. Alaettinoglu, V. Jacobson, and H. Yu, “Towards millisecond IGP convergence,” IETF Draft, 2000.

[21] K. P. Gummadi, H. V. Madhyastha, S. D. Gribble, H. M. Levy, and D. Wetherall, “Improving the reliability of Internet paths with one-hop source routing,” in *USENIX OSDI*, 2004.

[22] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, “Achieving convergence-free routing using failure-carrying packets,” in *ACM SIGCOMM*, 2007.

[23] H. Zlatokrilov and H. Levy, “Area avoidance routing in distance-vector networks,” in *IEEE INFOCOM*, 2008.

[24] G. T. K. Nguyen, R. Agarwal, J. Liu, M. Caesar, P. B. Godfrey, and S. Shenker, “Slick packets,” in *ACM SIGMETRICS*, 2011.

[25] R. Sherwood, A. Bender, and N. Spring, “Measuring ISP topologies with Rocketfuel,” in *ACM SIGCOMM*, 2002.

[26] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, and C. Diot, “Measurement and analysis of single-hop delay on an IP backbone network,” *IEEE Journal on Selected Areas in Communications*, vol. 21, no. 6, 2003.

[27] Y.-J. Kim, R. Govindan, B. Karp, and S. Shenker, “Geographic routing made practical,” in *USENIX NSDI*, 2005.

[28] Q. Zheng and G. Cao, “Minimizing probing cost and achieving identifiability in network link monitoring,” in *IEEE ICDCS*, 2010.