

SafeGuard: Safe Forwarding during Route Changes

Ang Li
Dept. of Computer Science
Duke University
angl@cs.duke.edu

Xiaowei Yang
Dept. of Computer Science
Duke University
xwy@cs.duke.edu

David Wetherall
Intel Research Seattle &
University of Washington
djw@cs.washington.edu

TR-2008-04

ABSTRACT

This paper presents the design and evaluation of SafeGuard, an intra-domain routing system that can safely forward packets to their destinations even when routes are changing. SafeGuard is based on the simple idea that packets carry a destination address plus a local estimate of the remaining path cost. We show that this simple design enables routers to detect path inconsistencies during route changes and resolve on a working path for anticipated failure and restoration scenarios. This in turn means that route changes do not disrupt connectivity despite the fact that routing tables are inconsistent over the network. We evaluate the router performance of SafeGuard using a prototype implementation based on NetFPGA and Quagga. We show that SafeGuard is amenable to high-speed hardware implementation with low overhead. We evaluate the network performance of SafeGuard via simulation, and find it reduces periods of disruption to a minimal duration, *i.e.*, the failure detection time.

1. INTRODUCTION

One of the well-known problems in networking is to reduce forwarding disruptions while routes are changing. When a network change occurs (*e.g.*, a link goes down), routers will be informed and recompute their forwarding tables to adapt to the change, a process known as routing convergence. During convergence, routers adjacent to failures may not have valid routes, or routers may have inconsistent next hop sequences that form temporary loops called micro-loops. A measurement study shows that micro-loops formed during intra-domain routing convergence in a tier-1 network account for 90% of the total packet loss [20].

Unfortunately, network changes happen frequently and are part of everyday's operation [21, 29], while real-time applications such as VoIP, online gaming, video conferencing, and IPTV need network services that are free of interruption [5, 9]. Even sub-second periods of packet loss or delay may adversely impact the users of these applications [9, 30], but the present Internet routing system can easily produce noticeable periods of disruption following a change in the network, *e.g.*, a failure or restoration of equipment, an ISP policy change, or a traffic engineering route adaptation.

This long-standing networking problem has brought forth a plethora of solutions both from academia and industry: [8, 13, 14, 15, 16, 17, 22, 24, 26, 32, 37]. However, all of the effective solutions [14, 16, 22, 26] that can minimize the disruption period to its theoretical minimal: the failure detection time, require a burdensome change of the underlying routing protocol; some also require on-demand route computation and variable packet header length. Consensus routing replaces routing convergence with distributed consensus that synchronizes route updates among routers [22]. Other approaches introduce a strict update order [14, 17] that prevents routers from independently updating their routing tables in parallel. The convergence-free routing [26] approach abandons routing convergence, but requires packets to carry all failed components they encounter and routers to compute paths on demand; it also requires an additional centralized protocol to install and update static network maps at routers.

A simple, effective, and efficient solution that minimizes forwarding disruptions while routes are changing remains elusive. By simple, we mean a solution that does not increase the complexity of routing convergence. Link state routing protocols such as OSPF introduce no state dependency among routers and allow each router to dynamically update their routing tables *independently* and *in parallel*. It has been shown to have the least complexity and converge faster than other alternative convergence proposals [12]. By effective, we mean to minimize periods of disruption to the failure detection time, a minimal period that cannot be shortened without physical layer improvement. By efficient, we mean a solution that is amenable for high-speed hardware implementation and retains much of the IP forwarding design: simple table lookups with fixed packet header length. We recognize that satisfying all three goals may not align well with commercial market constraints such as immalleable router hardware, but we believe an intellectual endeavor to explore this unique design point is worthwhile and may influence future network designs.

As a first step, this paper presents a new approach called SafeGuard that is simple, effective, and efficient at minimizing network service interruptions for intra-domain route changes. In SafeGuard, a packet carries the cost of the remaining path to its destination. Routers use this cost to de-

tect route changes and resolve on a valid alternative path. The routing system can rapidly disseminate the news of a failure (or a restoration) to quickly return to its optimal state. It can do this in any update order without the concern of breaking connectivity. We believe that this work is the first design to achieve the effective and efficient goals without increasing the complexity of routing convergence.

A key observation underlying this design is that the remaining path cost succinctly encodes valuable path information that we can leverage. On one hand, it can be embedded into a fixed-length label, similar to a path identifier; on the other hand, it sheds light on what links lie on the remaining path, approximating the effect of a full source route. When a router's local path cost differs from that carried by a packet, it indicates route changes; the cost difference informs a router whether its local path is valid or not, as outdated paths using failed components would have lower costs than a working one. A router can use this information to select a working path from a set of pre-computed alternatives and forward packets to their destinations along the working path. Of course, pre-computation limits protection to anticipated failure or restoration scenarios (such as single links, nodes, or shared risk link groups), but the trade-off is favorable, because multiple independent events rarely occur in complete synchrony, and events that are off by a few hundreds of milliseconds (*i.e.*, the routing converge periods) are fully protected as consecutive anticipated events.

We evaluate SafeGuard in terms of both router and network performance. To assess router performance, we implement SafeGuard in hardware using the NetFPGA platform [18] and in software using the Quagga routing suite [4]. Our experiments show that SafeGuard is amenable to high-speed hardware implementation and introduces memory and computational overhead comparable to other practical solutions that use pre-computed paths to suppress failures [10].

We use simulations over a range of real, inferred, and randomly generated topologies to assess network performance. We find that no packets are trapped in loops with SafeGuard versus micro-loops that can amplify traffic up to 50 folds and occur on average for 16% of the routing transitions we test. Further, after single component failures, SafeGuard fully restores connectivity after the failure detection time.

The rest of the paper is organized as follows. § 2 introduces the problem we tackle. § 3 and § 4 describe SafeGuard and its properties. § 5 evaluates the router and network performance of the SafeGuard design. We compare SafeGuard with related work in § 7 and conclude in § 8.

2. COST AS A SAFEGUARD

The SafeGuard design aims to provide a simple and efficient solution that reduces forwarding disruption during route changes to the failure detection time without altering or increasing the complexity of the routing convergence process. To achieve this goal, we make a packet carry the remaining path cost to its destination and routers pre-compute

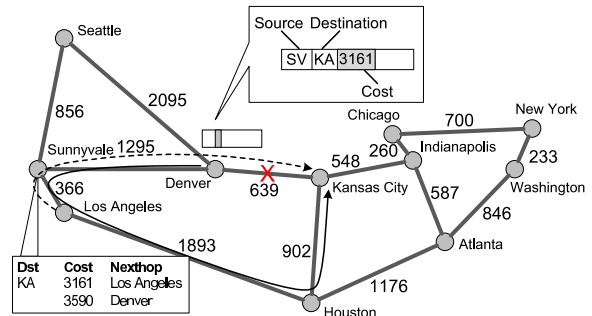


Figure 1: Forwarding using cost-carrying packets on the Abilene network.

alternative paths for anticipated route changes. In this section, we use an example to illustrate how SafeGuard works. We present the detailed design in the following section.

2.1 Using Cost to Resolve Inconsistencies

Figure 1 shows an example using the Abilene network topology. All link weights are taken from the Abilene network configuration [1]. Suppose the link between Denver and Kansas City fails, and the Denver node has updated its forwarding table to use the Sunnyvale node to reach Kansas City. The Sunnyvale node has not learned of the failure nor updated its forwarding table. Without carrying a path cost, packets will loop temporarily between Sunnyvale and Denver, as the Sunnyvale node is still using the outdated next hop Denver to reach Kansas City.

If instead, packets carry the remaining path cost and routers update this cost at each hop using their local estimates, the Sunnyvale node will stamp the remaining path cost to Kansas City (639) into the packets it forwards to Denver. As the Denver node has updated its forwarding table to bypass the failed link to Kansas City, its local path cost (4456) is larger than the packet cost (639). This difference indicates that routes are changing and some node may have incorrect forwarding paths. More importantly, it suggests that Denver's path bypasses a failure that has not been seen by its upstream router Sunnyvale. This is because the updated topology that removes a failed component must have "longer" shortest paths. Therefore, the Denver node can infer that its path is working. It then forwards the packet to its default next hop Sunnyvale and updates the packet cost to the remaining path cost (3161).

Here we emphasize again that what distinguishes path cost from a path identifier (*e.g.*, an MPLS label) is the ability to resolve which path may have included a failed (or a newly restored) component. This in turn assists a router to choose a working path. In contrast, if a packet simply carries a path identifier, a router may detect a path inconsistency, but cannot tell which path, its default path or an upstream router's default path, is a working path.

When the Sunnyvale node receives the packet with a path cost 3161 from the Denver node, it can infer that its default

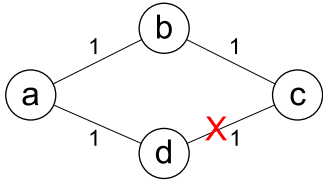


Figure 2: In this simple topology each link has the cost of one. There are two equal cost paths from a to c : $a \rightarrow b \rightarrow c$ and $a \rightarrow d \rightarrow c$.

path has a failure of which it has not learned. This is because its local cost (1934) is lower than the packet cost (3161). Instead of using its default path, the Sunnyvale node will try to find an alternative path that matches the packet cost. By pre-computing all alternative paths that bypass a single component on its default shortest paths, the Sunnyvale node is able to map the path cost (3161) to an alternative path: Sunnyvale \rightarrow Los Angeles \rightarrow Houston \rightarrow Kansas City, and forwards the packet to the correct next hop (Los Angeles) without forming a loop.

2.2 Distinguishing Equal-Cost Paths

As shown in the above example, after detecting a path inconsistency, a router may also use the path cost carried by a packet to select a working path that matches the cost. This would work well if different paths to the same destination have different costs. However, practical network configurations may have equal cost paths for load balancing and backup reasons. When this occurs, a path cost may not reliably identify a working path as shown in Figure 2. There are two equal-cost paths from a to c : $a \rightarrow b \rightarrow c$ and $a \rightarrow d \rightarrow c$. Suppose the node d that detects the link failure to c reroutes the packet to a with a remaining path cost 2. When node a receives this packet, the packet cost matches both paths: one valid and one failed. The node a may erroneously forward the packet along the invalid path $a \rightarrow d \rightarrow c$.

To address this issue, the SafeGuard design adds a random noise to a regular link cost to help distinguish equal-cost paths. With this design, a node is able to select a valid alternative path with high probability. We describe this design in more detail.

3. DESIGN

The SafeGuard design has four main components: enhanced link and path costs, packets that carry enhanced path costs, a pre-computed alternative path database (APD) maintained by routers, and a forwarding algorithm that is provably safe and can forward packets to their destinations without forming loops during anticipated route changes. We describe each design component in turn.

3.1 Enhanced Link and Path Costs

The SafeGuard design embeds a noise into the lower k -bit of a link’s cost metric to help distinguish multiple equal cost paths to the same destination. We refer to this metric

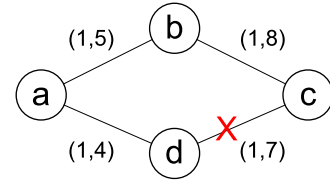


Figure 3: In this simple topology each link has the cost of one and a link noise. The enhanced cost of each link is marked in the format $(cost, noise)$. There are two paths with equal regular costs from a to c : $a \rightarrow b \rightarrow c$ and $a \rightarrow d \rightarrow c$.

as the enhanced link cost metric, and denote it as \overline{cost} . An enhanced link cost can be viewed as a pair of two values: $(cost, noise)$, where the higher-order bits $cost$ is the regular link cost configured by a network operator without considering failure recovery issues, and the lower order k -bit $noise$ is a random value between $[0, 2^k - 1]$.

We further define the addition and comparison functions over the enhanced link cost metric to enable shortest path computation using enhanced costs. If two enhanced link costs: $l_1.\overline{cost} = (l_1.cost, l_1.noise)$ and $l_2.\overline{cost} = (l_2.cost, l_2.noise)$, are added together, each part is added separately: $l_1.\overline{cost} + l_2.\overline{cost} = (l_1.cost + l_2.cost, l_1.noise + l_2.noise)$. Two enhanced link costs are compared lexicographically: $l_1.\overline{cost} > l_2.\overline{cost}$, if $l_1.cost > l_2.cost$, or $l_1.cost \equiv l_2.cost$ and $l_1.noise > l_2.noise$.

An enhanced path cost is the sum of all enhanced link costs on the path p : $p.\overline{cost} = (\sum_{l_i \in p} l_i.cost, \sum_{l_i \in p} l_i.noise)$. We refer to the first term as the regular path cost $p.cost$, and the second as the path noise $p.noise$. When a router stores an enhanced path cost, it only stores the last k -bits of the path noise: $p.\overline{cost} \leftarrow (p.cost, p.noise \bmod 2^k)$ to ensure that an enhanced path cost can be encoded in a fixed-length label with the last k -bit being the path noise.

3.2 Packets that Carry Costs

In the SafeGuard design, a packet carries a fixed-length label $pkt.\overline{cost}$ that encodes the enhanced path cost of its remaining path to the destination. Routers use the regular path cost field $pkt.cost$ to detect path inconsistencies and to resolve whether its default path or an upstream router’s default path is a working path. In the latter case, the router uses the entire packet cost $pkt.\overline{cost}$ to select a working path.

We also use 1-bit in a packet header to indicate whether a packet has been detoured at least once due to path inconsistency. When this bit is off, a packet is referred to as in the *normal* mode; otherwise, it is referred to as in the *escort* mode. Routers can use this bit to detect non-anticipated topology events and abort the effort of selecting a working path. This is because for anticipated events, a packet only needs to be detoured at most once before it reaches its destination (§ 4).

3.3 Alternative Path Database

The SafeGuard design uses an alternative path database (APD) to store alternative paths to reach a destination in case a failure occurs on a router’s current network map. Conceptually, an APD is a table that maps the destination and enhanced path cost pair: (dst, \overline{cost}) , to a valid next hop: $(dst, \overline{cost}) \rightarrow nexthop$.

To avoid on-demand computation, SafeGuard makes a router pre-compute alternative paths by anticipating a future failure on its current network map G . If a router anticipates a network element e (a link, node, or shared risk link group (SRLG) [34]) may fail in the future, it computes an alternative shortest path to reach a destination by removing the element e from the network map G . This computation uses the enhanced link cost metric. The router stores the enhanced path cost and the next hop in its APD. If the same alternative path can bypass multiple elements, the router only stores the path once.

As we will discuss in § 3.5, with randomly chosen link noises, different alternative paths will have different enhanced path costs with high probability. This would warrant that a router selects a working path with high probability. As shown in Figure 3, the enhanced path cost for the path $a \rightarrow b \rightarrow c$ is now (2,13), different from the enhanced path cost (2,11) of $a \rightarrow d \rightarrow c$ (had $d \rightarrow c$ not failed). When the node d reroutes a packet to a , it will carry the enhanced path cost (2,13), which unambiguously identifies the alternative path $a \rightarrow b \rightarrow c$ that bypasses the failed link. If in the unlikely case that a router has different alternative paths with the same enhanced path costs, the router will store the path that bypasses the largest number of failed elements on its default paths to maximize the utility of this path.

A router may re-compute its APD whenever it receives a routing update that changes its current network map. In the case of a failure event, the updated APD is only used when the next topology change occurs. Thus the computation is not urgent and could be done in low priority after a router has updated its forwarding tables. In the case that a topology update results in a better connected topology, *e.g.*, a link up, a router can save computation by swapping the next hops and path costs in its previous forwarding tables to its APD before updating its forwarding tables. As we will soon describe in the next subsection, paths in a router’s previous forwarding tables would be the alternative paths computed without the newly added topology element.

3.4 Forwarding

We now describe how a SafeGuard router forwards packets without loops during routing transitions. We first describe how a router sets up its forwarding tables and then the forwarding algorithm.

3.4.1 Forwarding Table Entries

As in the current intra-domain routing system, an entry in a router n_i ’s forwarding table contains the mapping between a destination prefix to a set of next hops n_{i+1} . These

next hops are computed using the regular link costs without considering link noises. This design ensures that packets are always forwarded along their regular shortest paths, *i.e.*, any of the equal cost multiple paths, when routing is not in transition.

The SafeGuard design also adds path cost information to a router’s forwarding tables. Each forwarding table entry includes not only the next hops n_{i+1} , but also the enhanced path cost from each of the next hop n_{i+1} to the destination: $n_{i+1}.\overline{cost}$. Moreover, it also includes the enhanced path cost from the current router n_i to the destination: $n_i.cost$. These path costs are used by the forwarding algorithm to detect route changes and choose a working path.

A router n_i may efficiently compute the enhanced path costs after it computes its regular shortest paths. Let T denote the shortest path graph generated by the regular shortest path computation. The router can compute the enhanced path cost $n_i.\overline{cost}$ for each destination by running a shortest path computation on T (rather than the full network map) using the enhanced link cost metric. This algorithm is correct because enhanced costs are compared lexicographically, so the shortest path defined over the enhanced cost metric must be one of the shortest paths defined over the regular link costs. Similarly, the enhanced path cost associated with each next hop ($n_{i+1}.\overline{cost}$) can be computed efficiently using the shortest path graph T with each next hop node being the root node of the shortest path computation.

3.4.2 Packets in the Normal Mode

When a packet arrives at a router n_i , the router first checks its forwarding mode. If the packet is in the normal mode, the router neglects the path noise and only compares its local regular cost $n_i.cost$ with the packet cost $pkt.cost$. It uses this comparison to detect and resolve any inconsistency, and turns on the escort mode if necessary. If the packet is already in the escort mode, the router uses the enhanced packet cost $pkt.\overline{cost}$ to identify a working path and forwards the packet along the path. The pseudo-code of the algorithm can be found in the Appendix.

A packet enters the network in the normal mode. When a router n_i receives an incoming packet in the normal mode, it compares the regular costs only. There are three possible outcomes:

Matching Costs ($n_i.cost \equiv pkt.cost$): This indicates that the router n_i and its upstream router have consistent forwarding paths. The router n_i selects a next hop n_{i+1} from its forwarding table, updates the packet’s cost label using the enhanced path cost associated with the next hop n_{i+1} : $n_{i+1}.\overline{cost}$, and forwards the packet to the next hop n_{i+1} .

Higher Local Cost ($n_i.cost > pkt.cost$): This inconsistency shows that a router’s local cost is higher than its upstream router’s cost. The network must be in a routing transition, as the router n_i has computed different paths from other routers. As the packet is in the normal mode, this is the first cost inconsistency the packet encounters. The router

will attempt to resolve the dispute.

Given the local cost is higher, the router n_i resolves the inconsistency by forwarding the packet along its default paths. Its default paths must be valid because n_i must have a less connected topology than other routers, as it has a higher path cost. If it is a failure event, n_i must have already updated its forwarding table according to the event, and its path will bypass the failed component. If it is a restoration event, n_i must have not updated its forwarding table, and its path will not use the newly restored or added component, but can still reach the destination. So as in the matching costs case, the router selects one of its next hops, updates the packet's cost label using its local cost, and forwards the packet to the next hop. The router will also turn on the escort bit to notify the downstream routers that a path inconsistency has been encountered.

A special case of higher local cost inconsistency occurs when a packet reaches a router adjacent to a failure, but the router has not updated its forwarding table yet. The router's next hop is invalid and its local cost $n_i.cost$ is infinite. In this case, a router n_i will immediately start using the alternative path pre-computed to bypass its next hop to forward and update the packet cost. The router will first try to use the alternative path that bypasses its next hop node. If such a path is unavailable, *e.g.*, the next hop is the destination, the router uses the alternative path computed by removing the link to reach the next hop. Let the enhanced cost of the alternative path be $n_i.\overline{cost}'$, and the next hop of it be n'_{i+1} . The router n_i turns on the escort bit, updates the packet's cost to be the remaining enhanced path cost of the alternative path by subtracting the next hop link cost: $pkt.\overline{cost} \leftarrow n_i.\overline{cost}' - l_{n_i \rightarrow n'_{i+1}}.\overline{cost}$. The subtraction is done separately for regular cost and noise and the subtraction of noises is modulo 2^k based as described in § 3.1.

Lower Local Cost ($n_i.cost < pkt.cost$): This inconsistency shows that a router's local cost is lower than its upstream router. Again, the network must be in a transition. As the router has a lower local cost, it must have a network topology with more components. It is no longer safe to forward along the router's default next hops, because it may include a failed component.

To resolve a lower local cost inconsistency, a router uses the packet's enhanced cost $pkt.\overline{cost}$ to look up an alternative path in its APD, because its APD is computed using smaller topologies than its current one, and a path with higher cost may be found in the APD. Suppose this lookup returns a next hop n'_{i+1} . The router n_i updates the packet's cost label using the alternative path's cost, turns on the escort bit, and forwards the packet to n'_{i+1} .

3.4.3 Packets in the Escort Mode

If an incoming packet is in the escort mode, the router will try to forward the packet along the working path chosen by the upstream router who detects the cost inconsistency. It is not enough to only consider the regular costs because they

k	Collision Probability
10	0.0097
16	0.00015
24	6.0×10^{-7}
32	2.3×10^{-9}

Table 1: The probability of having two equal enhanced cost paths between two nodes when there are $c = 5$ regular equal cost paths, given different noise length k .

may not uniquely identify the valid path among equal cost paths. In this case, the router n_i will try to find a path whose enhanced cost matches exactly with the incoming packet's enhanced cost. To do this, the router first compares its local enhanced path cost $n_i.\overline{cost}$ with the packet's cost $pkt.\overline{cost}$. There are two outcomes:

Matching Costs ($n_i.\overline{cost} \equiv pkt.\overline{cost}$): This indicates that the router's shortest path with cost $n_i.\overline{cost}$ is consistent with the chosen working path. In this case n_i will forward the packet using the next hop associated with the enhanced path cost, and update the packet's cost label accordingly.

Unmatching Costs ($n_i.\overline{cost} \neq pkt.\overline{cost}$): In this case, the upstream router that detects a route inconsistency has chosen a working path that is different from the router's current shortest path defined over the enhanced costs. The router uses the packet's cost label $pkt.\overline{cost}$ to look up an alternative path in its APD. If an alternative path is returned, it updates the packet's cost and forwards the packet to the corresponding next hop.

If there is an APD look-up failure, it indicates unexpected topology changes such as multiple concurrent failures or different alternative paths having the same enhanced cost. In this case, not all routers would have a working path. Any further forwarding may risk forming a loop. When this occurs, a router may either demote the packet or discard it. Our design chooses to discard for simplicity.

3.5 Cost Collision Analysis

The link noises are introduced to distinguish equal cost paths. We assign random values to link noises to distinguish any two equal cost paths with high probability. We analyze this probability, and also discuss an alternative mechanism that deterministically generates collision-free link noises for a given network topology.

A link noise is chosen randomly between $[0, 2^k - 1]$. A path noise that is the sum of link noises modulo 2^k is also randomly distributed. If a node has c paths to the same destination with the same regular path costs, the probability that no two such paths have the same enhanced path cost is:

$$1 \cdot \left(1 - \frac{1}{2^k}\right) \dots \left(1 - \frac{c-1}{2^k}\right) = \frac{2^k!}{2^{ck}(2^k - c)!}$$

Table 1 shows the probability of collision when $c = 5$ for various values of k . In practice, c is typically small (< 5)¹,

¹This observation is based on five proprietary ISP topologies with real link costs.

because two backup paths usually suffice.

As can be seen, with practical values of c and k , the probability of collision is low. Our simulations use a 10-bit noise value, and we do not run into any collisions on all simulated topologies, including an inferred tier-1 topology. Therefore, we think that the probability of collision can be practically ignored if we use a 32-bit or longer noise.

Alternatively, we have also designed an algorithm that deterministically generates a collision-free noise configuration for a given network topology. The algorithm runs in polynomial time in terms of the number of equal cost paths (details are in Appendix). One caveat about this algorithm is that the collision-free noise configuration is generated for a given network topology; a future topology change may require a new noise configuration to be collision-free (although this occurs with low probability). For simplicity, we opt to use random noises.

3.6 Practical Considerations

Atomic Update: One practical issue worth mentioning is partial router updates. A topology change involving multiple links (such as a node or SRLG failure) will trigger multiple routing updates in a link-state routing protocol such as OSPF or IS-IS. If a router partially updates its forwarding tables according to a subset of the routing messages, its local cost may not match the alternative paths' costs in other routers' APDs, as an APD only includes paths that bypass a whole component (*e.g.*, a node with all its links removed).

Fortunately, there already exist several practical techniques to enable atomic router updates, because partial updates churn up routers' CPU power and are harmful for routing convergence. These techniques include OSPF and IS-IS's delay timers that batch route update processing, throttling techniques [3] that automatically adjust the delay timer according to the incoming rate of route updates, and the recent proposed LSA correlation technique [19] that identifies concurrent multiple link updates by correlating the information carried by different LSAs. Therefore, in the SafeGuard design, we assume that routers use atomic updates for an anticipated topology update (*i.e.*, a link, node, or SRLG).

Deployment: The SafeGuard design can be deployed by individual ISPs on the Internet without introducing new protocols. Routers need to upgrade to pre-compute alternative paths (§ 3.3), add the enhanced path costs in its forwarding tables (§ 3.4.1), and support the new forwarding algorithm (§ 3.4).

One possible way of deploying SafeGuard on today's hardware routers is to exploit the MPLS infrastructure. A path cost label may be embedded into an MPLS label upon a packet's entrance to an ISP network and swapped at each hop during forwarding. We defer a complete deployment study of SafeGuard as future work.

4. PROPERTIES

We now briefly describe the forwarding properties of the SafeGuard design. We omit formal proofs due to the lack of space, and refer interested readers to our technical report [27]. When stating those properties, we do not consider congestion loss, because it is not caused by violations of forwarding consistency. We also ignore the failure detection time during which routers may forward packets to a failed link without noticing the failure, and the router initialization period during which a newly added router has not obtained any topology information.

Property 1 *Packets will follow regular shortest paths to reach their destinations in the normal mode when the network is in steady state.*

This property holds because routers compute the next hops in their forwarding tables using only the regular link costs. The SafeGuard forwarding algorithm compares only the regular path cost in a packet with a router's local estimate for normal-mode packets. In steady state when routes are not changing, these two costs will always match and packets will reach their destinations without encountering any inconsistency.

Property 2 *If enhanced path costs are distinct, during the routing transition period in which only one network element changes its status and the network is not partitioned, a packet will be forwarded to its destination in either the normal or the escort mode.*

This property holds because when there is only one element changing its status, a router always has a working path in either its forwarding tables or its APD, dependent on whether the router has learned of the change and updated its forwarding tables. Therefore, if any router on a packet's path detects a cost inconsistency, it is able to either use its default path or a valid alternative path in its APD to forward the packet.

Property 3 *A packet will not be trapped in a micro-loop without being discarded.*

By trapped in a micro-loop, we mean that if all routers stop updating their forwarding tables after forming a loop, a packet will not escape the loop until its TTL expires. This property holds because a packet cannot traverse a node twice without a cost inconsistency. After one inconsistency is detected, the packet is marked as in the escort mode, and it will either follow a working path to reach the destination, or be discarded by a router that detects another cost inconsistency and cannot find a cost-matching path in its APD.

5. EVALUATION

In this section, we describe the evaluation of SafeGuard. We have evaluated both the router performance and network performance of SafeGuard.

5.1 Router Performance

The SafeGuard design adds computational and memory

Topology	# of Forwarding Table Entries	# of NotVia Entries		# of APD Entries		APD Computation Time (ms)		NotVia Computation Time (ms)	
Abilene Node:11 Link:28	11	Avg	15.4	Avg	17.3	Avg	0.165	Avg	0.093
		Max	17	Max	26	Max	0.176	Max	0.112
		Min	14	Min	12	Min	0.157	Min	0.073
Sprint Node:315 Link:1944	315	Avg	368.8	Avg	777	Avg	79.4	Avg	49.7
		Max	449	Max	1769	Max	89.2	Max	84.2
		Min	278	Min	534	Min	71.9	Min	36.8
Random Node:100 Link:394	100	Avg	116.6	Avg	276.1	Avg	6.2	Avg	2.7
		Max	140	Max	376	Max	11.9	Max	10.6
		Min	102	Min	149	Min	5.8	Min	2.0

Table 2: Summary of the memory and computational overhead introduced by SafeGuard. For memory overhead the normal forwarding table size and the number of NotVia entries are shown for comparison. For computational overhead the NotVia computation time is shown for comparison.

overhead to a router. A router does an additional cost comparison and a possible APD lookup during packet forwarding time. It also needs to compute the enhanced path costs in a router’s forwarding tables, pre-compute alternative paths that bypass anticipated network failures, and store the alternative paths in its APD. To quantify the overhead, we implement our design using NetFPGA [18] and the Quagga routing suite [4].

NetFPGA provides a hardware-accelerated data plane that emulates the real hardware-based forwarding engine in commodity routers. It comes with a reference implementation of an IPv4 router that leverages TCAMs (Ternary Content-Addressable Memories). Quagga comes with a full-fledged OSPF daemon (`ospfd`), which we use to run the underlying routing protocol. We implement the alternative path database using TCAM, and modify the NetFPGA reference router to support our forwarding algorithm described in § 3.4. We extend `ospfd` to compute the alternative paths and the regular shortest paths needed for the SafeGuard design after each routing update. The total modification includes ~800 lines of Verilog code and ~3000 lines of C/C++ code.

To evaluate the forwarding overhead of SafeGuard, we benchmark its forwarding performance using the NetFPGA implementation. We send small packets with 64 bytes to the NetFPGA data plane using a PC machine, and measure the packet throughput and the per-packet processing time. Our experiments show that SafeGuard’s per-packet forwarding overhead is 120ns. This would translate into 4.3Gbps throughput if per-packet processing is the bottleneck on a SafeGuard router’s forwarding plane.

We then evaluate the computational and memory overhead of SafeGuard. We first feed the modified `ospfd` daemon with different network topologies. To evaluate the overhead of computing shortest paths with the enhanced path costs (§ 3.4.1), we measure the time to compute all forwarding entries with the enhanced path costs, and compare it with the regular shortest path computation time. The results show that the modified shortest path computation using the enhanced costs only increases the computation time by 1%~3% compared to a router’s regular shortest path computation. As we describe in § 3.4.1, this is because the extra computations based on the enhanced link costs can be opti-

mized to use only links on the regular shortest paths, which typically are only a small portion of all links in the network.

We also measure the time it takes to compute the alternative paths and the size of the APDs. In comparison, we also implement the algorithms to compute the backup paths for NotVia [10], a practical IP fast reroute technique that uses pre-computed backup paths to bypass temporary failures, but does not prevent micro-loops during routing convergence. For both mechanisms, we compute the alternative paths to protect all single link and node failures. Our experiments run on a Pentium D 2.4GHz machine with 2GB memory.

Table 2 shows the time it takes to finish computing the alternative paths, and the number of additional entries a router keeps for SafeGuard and NotVia respectively for various topologies. As can be seen, alternative path computation in SafeGuard takes less than 100ms on the largest Sprint topology, and this time is comparable to NotVia. The number of entries in a router’s APD may be 2-8 times larger than a router’s intra-domain forwarding table, but is comparable to that of NotVia’s, suggesting that SafeGuard’s memory overhead is practically affordable. If the memory overhead becomes a practical concern, we can further reduce it by applying optimization. We omit the details for ease of exposition, but a detailed description about the optimization technique can be found in [27].

5.2 Network Performance

We use simulations to further study the network performance of SafeGuard. We have implemented SafeGuard in SSFNet [6], an event-driven simulator that has a complete OSPFv2 implementation. We use the simulator to evaluate whether SafeGuard can enable safe forwarding during route changes without altering the routing convergence process. To do so, we simulate routing convergence caused by various types of topology updates, and measure packet forwarding performance during the convergence periods. Next, we describe the performance metrics we use, the simulation settings, and the results.

5.2.1 Metrics

Flow Amplifying Factor: We measure how many times

Topology	Type	# of Nodes	# of Links
Abilene	Real	11	28
Telstra	Inferred	108	306
Exodus	Inferred	79	294
Sprint	Inferred	315	1944
Random	Random	100	394

Table 3: Summary of the topologies used in our simulations.

Parameter	Value
HelloInterval	50ms
RouterDeadInterval	250ms
SPF Delay	200ms
SPF Computation Time	$(0.00247n^2 + 0.978)$ ms
FIB/RIB Update Time	$rand([0.1, 0.11])$ psms

Table 4: Summary of the simulation settings. n is the number of routers in the network. p is the number of entries in the forwarding table.

a packet passes the same unidirectional link during route changes. We refer to this metric as the flow amplifying factor, because if a traffic flow of t Mb/s passes the same link for K times, then the flow’s peak rate on that link would become $K \times t$ Mb/s. This metric helps answer the question whether SafeGuard prevents forwarding loops during route changes.

Packet Loss Rate: We measure the packet loss rate of the flows that are affected by the updated component. This metric shows whether SafeGuard is effective in minimizing periods of disruption during route changes.

Path Stretch: A path stretch is defined as the ratio of the cost of a path taken by a packet to the shortest path cost in the network. Path stretch shows the forwarding sub-optimality during route changes. This metric measures the quality of the alternative path SafeGuard chooses to resolve a cost inconsistency. The lower the path stretch is, the better the path.

5.2.2 Simulation Setup

Network Topologies: We simulate on real, inferred, and randomly generated topologies summarized in Table 3. The inferred topologies are from the Rocketfuel project [38], and the random topologies are generated using the BRITE topology generator [2]. Real and inferred topologies contain precise or inferred link weights [28]. We use the random topology to test how SafeGuard works on networks with asymmetric link weights. The link weight in each direction is set independently, each using a random integer between 1 and 50.

Link delays of each topology are set according to the geographic proximity of their end nodes. If two routers are in different Point-of-Presences (PoPs), we infer the link delay between them from the geographical distance, and in the generated topology the nodes are randomly spread on a plane that has a similar size as the US continent. If two routers are in the same PoP, we assume the link delay is 0.1ms.

Topology updates: We simulate routing transitions for both

single element update and multiple independent update events. For single element update, we test single link up/down events and node up/down events. For multiple independent updates, we test two concurrent link failures. For each type of update we run 100 experiments with randomly chosen element updates.

After each update event, we send probing packets every 5ms between each pair of nodes. We use the probing packet traces to compute various metrics such as the traffic amplifying factor and packet loss rate. The probing packets’ TTLs are initiated to 128, the default TTL value of the Windows XP operating system. We do not simulate real traffic patterns because it is extremely time-consuming to simulate, and the simulations would not finish in a reasonable time, *i.e.*, a few days.

Mechanisms: We simulate SafeGuard as described in § 3.4 with the OSPF implementation of SSFNet. In comparison we also simulate vanilla IP forwarding with OSPF. We configure SafeGuard to pre-compute alternative paths for all single link and node failures. The various timers and delays of the OSPF implementation are summarized in Table 4. These parameters are set according to the values recommended by various fast convergence techniques [8, 23] and the values observed in commercial production routers [7, 15, 36]. We simulate fast convergence because SafeGuard’s benefits are even more prominent during slow convergence during which micro-loops last longer, but we desire to emphasize only the benefits not replaceable by fast convergence techniques.

5.2.3 Avoiding Forwarding Loops

Figure 4 and 5 compare the distributions of the flow amplifying factors of vanilla IP forwarding with OSPF and SafeGuard with OSPF in a real network topology (Abilene), an inferred tier-1 network topology (Sprint), and a randomly generated network topology (Random). Results from other topologies are similar and omitted. The distributions are drawn from all micro-loops we have observed in the tests. As shown in the figure, SafeGuard’s flow amplifying factor is ≤ 2 . This result shows that SafeGuard prevents packets from being trapped in micro-loops, as packets at most take one detour to reach their destinations.

In contrast, the vanilla IP forwarding with OSPF can have amplifying factors greater than 50. The trapped flows will be amplified for 50 times, which is likely to congest a link and cause congestion loss. One can also note that the Sprint topology suffers from loops with much larger amplifying factors than the other two. This is because routers in Sprint are aggregated in different Point-of-Presences (PoP), and the latencies between routers in the same PoP are small, as they are usually co-located in the same facility. Hence the forwarding loops occurred inside a PoP usually have high amplifying factors.

Table 5 summarizes the characteristics of micro-loops observed under the Sprint topology in our simulations. As can be seen, in case of a failure, more than 15% of the simula-

Update Type	# of Tests Containing Loops	Total # of Micro-loops	Total # of Links Involved	Loop Duration (ms)			
				Avg	Max	Min	Stddev
OSPF							
Link Failure	19	81	132	12.5	44.6	0.32	15.4
Node Failure	17	125	154	11.5	26.3	0.10	26.7
Link Up	4	7	14	11.9	40.7	0.80	21.8
Node Up	11	20	38	6.32	24.8	0.19	6.45
Two Link Failures	38	144	182	9.0	39.7	0.39	11.2

Table 5: Summary of loops during convergence in the Sprint topology. For each update type we run 100 experiments with randomly chosen topology updates.

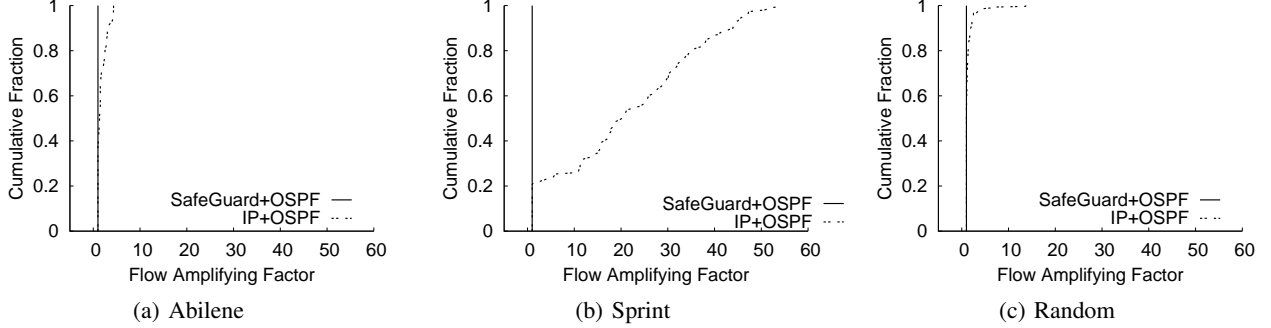


Figure 4: The cumulative distribution of the loops' flow amplifying factor on three topologies for single link failure case.

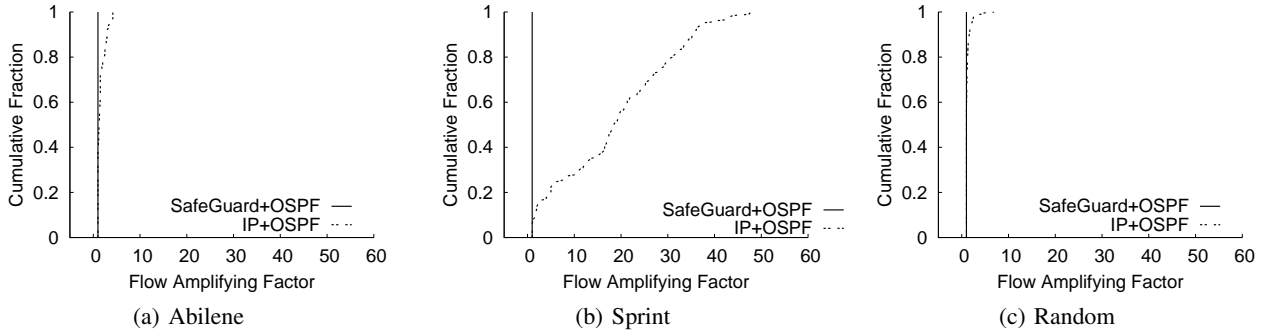


Figure 5: The cumulative distribution of the loops' flow amplifying factor for two link failures case.

tions contain forwarding loops, and the loops can involve a large number of links ($\sim 10\%$ of the total number of links in the topology) and cause congestion on them. Although with fast convergence the loops only last for less than 50ms, given the large amplifying factors shown above they may still cause voluminous flow amplification.

5.2.4 Reducing Packet Loss

Figure 6 and 7 show the average packet loss rates after a link failure and two link failures for each mechanism in three topologies. Note that the packet loss rates do not include congestion loss caused by micro-loops, because we do not simulate real traffic load. Therefore, the packet loss rates we measure are those caused by failed routes. For vanilla IP forwarding, the packet loss rate could be much higher in practice because micro-loops would lead to congestion loss. We measure the packet loss rate at time t by counting how many probe packets sent during the period $[t, t + 10ms]$ are

lost. We average the loss rates over all experiments for each type of update event.

As can be seen in Figure 6 and 7, SafeGuard successfully reduces packet loss rates, while vanilla IP forwarding has much higher packet loss rate, as packets may encounter failed routes until the network has converged, and the forwarding disruption times may last about 600ms. In a single failure case, after the failure is detected (after $\sim 200ms$), SafeGuard is able to rapidly bypass failures through the alternative paths. The time is independent of the size of the network, as SafeGuard only needs the routers adjacent to the failure to detect the failure and does not need the failure to be disseminated over the network. In some of the figures, the packet loss rates do not reach zero because the network is disconnected. In the two link failure case, SafeGuard may also experience packet loss during routing convergence as routers have not pre-computed the alternative paths to bypass them.

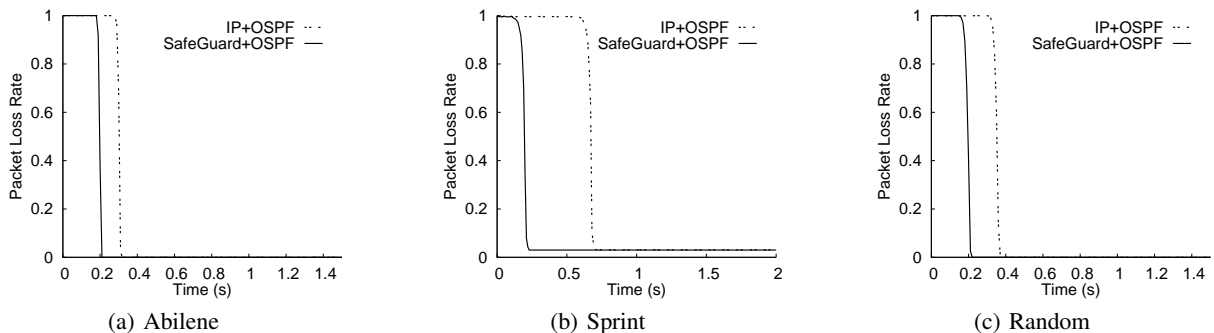


Figure 6: The average packet loss rate after a single link failure. X-axis is the time-line. The failure happens at time 0, and is detected after 200 ~ 250ms. Y-axis is the packet loss rate for all probing flows that use the failed link.

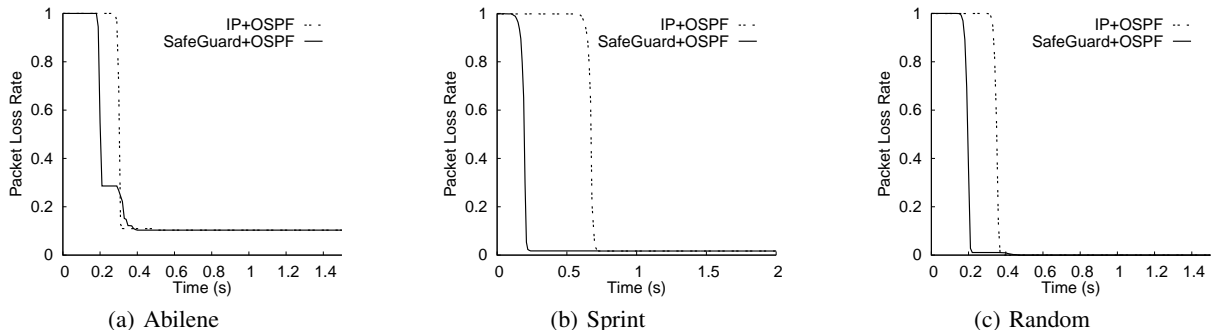


Figure 7: The average packet loss rate after two link failures. Other configurations are the same as in Figure 6.

5.2.5 Path Stretch

Figure 8 shows the average path stretch during routing convergence for a single link failure case. The path stretch is averaged over all source and destination pairs whose default forwarding paths include the failed link. As can be seen, with SafeGuard the path stretch smoothly reduces to 1 during convergence, meaning the forwarding paths gradually shift to the new shortest paths as routers update their states independently. However under vanilla IP forwarding one can observe path stretch surges during routing convergence because of micro-loops: some packets reach their destinations after a micro-loop dissolves but have looped for dozens of times.

5.2.6 Convergence Time

Finally, we measure the routing convergence time with SafeGuard. A main design goal of SafeGuard is not to alter or increase the complexity of routing convergence so that routers can independently update their routing tables without any state dependency or centralized coordination. The convergence time is measured from the time the first router announces a routing update to the time that all routers have finished updating their forwarding tables.

Figure 9 shows the SafeGuard convergence time under the realistic OSPF settings for three topologies. Since SafeGuard does not change the convergence scheme, it converges as fast as OSPF can accomplish. From the figure we can see

that even for the largest Sprint topology, the network can converge within one second, consistent with previous studies [8, 15]. This fast convergence property makes the network resilient and responsive to changes, and also reduces the time a packet follows a suboptimal path, *e.g.*, reaching a failure first before it is rerouted.

6. REDUCE MEMORY OVERHEAD

SafeGuard introduces non-trivial memory overhead as it needs the router to record all alternative paths that bypass one element. Although the actual size of an APD is much smaller than this number (Table 2) and modern routers are capable of handling much larger BGP tables, sometimes it may still be desirable to reduce the size of the database to make it more amenable for old model routers.

We introduce an optimization technique that can significantly reduce the APD size. The key idea is to store an alternative path only when the path’s next hop is different from a node’s regular next hop. Intuitively, if a router is far away from a topology change point, the change is unlikely to affect its next hop to reach a destination, although it may affect its path cost. This is because a failure can often be bypassed locally, using nodes only in the neighborhood of the failure. Therefore, it is likely that most of a router’s alternative paths share the same next hop as its regular next hop. If a router only stores those that have different next hops, it can significantly reduce its APD size. If a router has mul-

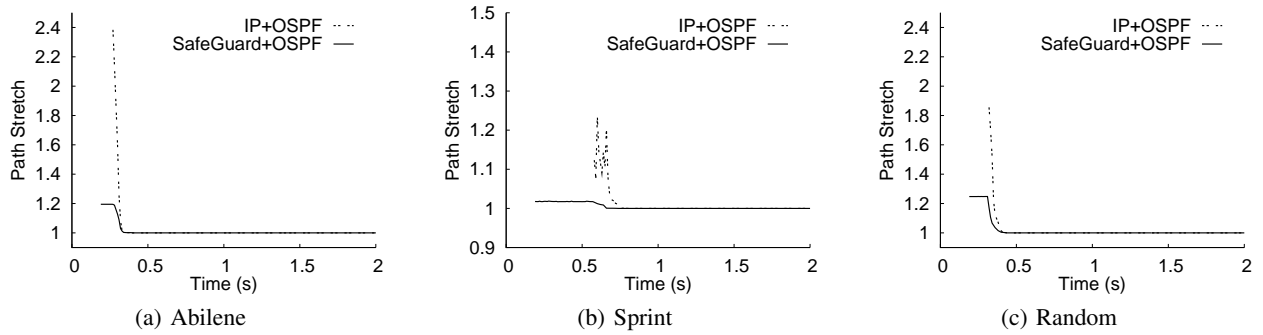


Figure 8: The average path stretch after a link failure. X-axis is the time-line. The failure happens at time 0, and is detected after 200-250ms. Y-axis is the packet stretch for all probing packets that previously pass the failure.

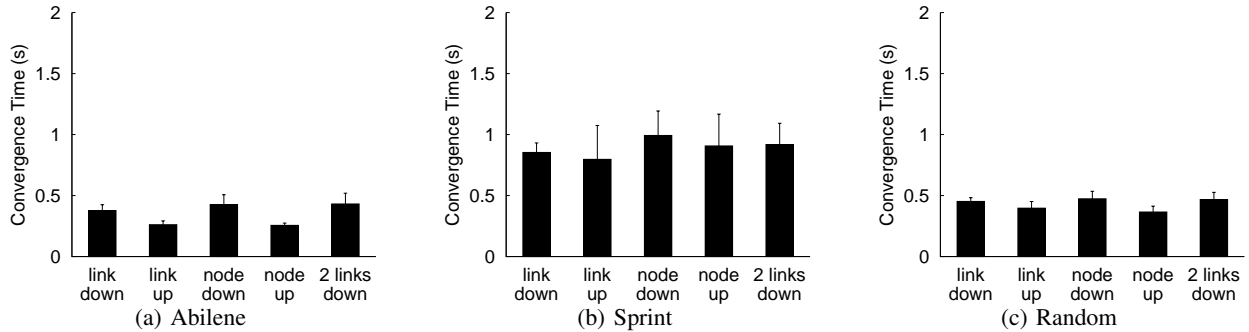


Figure 9: The averaged SafeGuard(OSPF) convergence time after different network changes. The error bars show the standard deviations. SafeGuard essentially achieves the same convergence time as OSPF because it does not modify convergence.

tuple equal cost shortest paths and an alternative path’s next hop matches a router’s shortest path next hop over enhanced costs, the router does not need to store the alternative path either.

The forwarding algorithm described in § 3.4 needs to be modified to utilize the optimization. Previously a packet is dropped if an APD lookup returns no result. With optimization, however, a lookup failure can also mean the corresponding alternative path is “optimized” because its next hop is the same as the regular next hop. Thus we modify the algorithm to keep forwarding a packet along the regular next hop even if an APD lookup returns no result, and update the packet’s enhanced cost by subtracting the link’s enhanced cost. This change will not affect the forwarding in stable state or under single component changes because in those cases an APD lookup failure will never happen with the original forwarding algorithm, and with optimization a lookup failure can only mean the alternative path is optimized. However when multiple concurrent changes happen the optimized algorithm might fail to drop a packet that is not repairable by the pre-computed alternative paths, and hence packets can still be trapped in loops.

Despite the downside, we think the optimization might still serve as a worthy trade-off if memory overhead is really a pressing issue. This is because multiple concurrent updates are much rarer compared to single topology changes,

and even concurrent updates do happen they might still be protected by using alternative paths pre-computed for single changes, especially when the updates happen far from each other.

7. RELATED WORK

In this section, we compare the SafeGuard design with related work. As we will show, SafeGuard is the only system that reduces periods of disruption to the failure detection time without changing the routing convergence process.

Researchers have proposed to redesign the routing convergence process to minimize periods of disruption [14, 17, 22, 26]. These schemes generally require routers to synchronize or to impose a strict order on their routing updates [14, 16, 17, 22]. Although they can effectively reduce disruption periods to the failure detection time as SafeGuard, these schemes introduce state dependency among routers, thereby increasing the complexity of the routing convergence process and slowing down routing convergence [12].

Convergence-free routing [26] eliminates routing convergence and uses a centralized coordinator to distribute static network maps to routers, similar to the Routing Control Platform (RCP) [11]. Packets carry a list of failed components they encountered as a router’s static network map does not have failure information; packets must hit failures before they can be rerouted; and routers employ on-demand compu-

tation when they first learn of the failures to forward packets along a valid path. For non-concurrent failure or restoration events, SafeGuard and convergence-free routing are both able to reduce periods of forwarding disruption to the failure detection time. For concurrent events that happen within a routing convergence period, SafeGuard can still guarantee loop-free forwarding, but may suffer temporary packet loss until routing converges, while convergence-free routing does not. But we think the trade-off is favorable, because such concurrent events are rare, and SafeGuard has a fixed header overhead, does not require on-demand computation, and forwards packets along the shortest failure-free paths except during routing convergence periods.

Another loop-avoidance technique [16] performs multiple rounds of routing convergence. In each round, independent router updates do not cause micro-loops, because the link cost changes are specifically calculated to prevent loops. This technique is suitable for planned failures because multiple convergence rounds must finish before a component can be removed. In contrast, SafeGuard can minimize forwarding disruptions for dynamic unplanned failures.

Exact Hop Count [35] is a proposal to prevent micro-loops in bridge networks by strictly checking the hop count in each packet's header. Hop count can be used to detect path inconsistencies, but unlike path cost, the hop count difference between two paths does not tell which path is valid. This is because practical routing protocols such as OSPF and IS-IS use fine-grained cost metrics to compute paths: a forwarding path computed with a failed component may have either a higher or lower hop count than a working path. In contrast, such a path will always have a lower (or equal) cost than a working path. Furthermore, exact hop count forwarding is incompatible with equal cost multiple path forwarding, as equal cost paths may have different hop counts.

In [41] the authors propose to detect forwarding loops through interface-specific routing. Unlike SafeGuard, this mechanism does not guarantee the detection of forwarding loops in asymmetric networks or during concurrent topology change events. Anomaly-Cognizant Forwarding [13] is a recent proposal that aims to detect and repair forwarding anomalies caused by BGP routing convergence. An ACF packet carries the AS path it has visited. This work focuses on intra-domain routing, and uses the remaining path cost as a safeguard to detect path inconsistency. Carrying cost is more efficient than carrying path as it does not require a variable-length header, and is more effective than interface-specific routing as it can detect loops in any circumstance and help routers to select the valid alternative paths.

There is also much work in enabling routers to rapidly reroute packets using backup paths after failure detection, including R-BGP, IPFRR, MPLSFRR, and Multiple Router Configurations [24, 25, 33, 37]. These proposals provide fast failure recovery, but they do not prevent micro-loops during routing convergence.

Multi-path routing is another approach to improve routing

availability. Work in this area includes routing deflection [40], path splicing [31], and MIRO [39]. Routers or end systems can choose a different path if the default path does not work. SafeGuard aims to enable routers to rapidly detect forwarding anomalies and repair them during routing transitions. It works in both single- and multi-path routing systems.

8. CONCLUSION

An important networking task is to improve network availability to better support real-time and mission critical applications. In this paper, we present the design and evaluation of SafeGuard, an intra-domain routing system that can effectively reduce packet loss and forwarding loops during network changes without increasing the complexity of routing convergence. In the SafeGuard design, a packet carries the remaining path cost to its destination. Routers compare the packet cost with their locally computed costs to detect inconsistent paths and select a working one among pre-computed alternatives. Our NetFPGA implementation of SafeGuard shows that SafeGuard is suitable for high-speed hardware implementation and has low memory and computation overhead. Simulation results show that SafeGuard reduces periods of disruption to the failure detection time during anticipated network changes and greatly reduces packet loss even for non-anticipated changes.

References

- [1] Abilene Observatory. <http://abilene.internet2.edu/observatory>.
- [2] BRITe Topology Generator. <http://www.cs.bu.edu/brite>.
- [3] OSPF Shortest Path First Throttling. http://cisco.com/en/US/docs/ios/12_2s/feature/guide/fs_spftrl.html.
- [4] Quagga Routing Suite. <http://www.quagga.net>.
- [5] Reducing Link Failure Detection Time with BFD. <http://www.networkworld.com/community/node/23380>.
- [6] Scalable Simulation Framework. <http://www.ssfnet.org>.
- [7] SPF Delay Timer. <http://www.juniper.net/techpubs/software/junos/junos74/swconfig74-routing/html/isis-summary53.html#1036104>.
- [8] C. Alaettinoglu, V. Jacobson, and H. Yu. Towards Milli-Second IGP Convergence. Internet draft, draft-alaettinoglu-isis-convergence-00.txt, Nov 2000.
- [9] C. Boutremans, G. Iannaccone, and C. Diot. Impact of link failures on VoIP performance. In *NOSSDAV*, 2002.
- [10] S. Bryant, M. Shand, and S. Previdi. IP Fast Reroute Using Notvia Addresses. Internet draft, draft-ietf-rtgwg-ipfrr-notvia-addresses-00.txt, Dec 2006.
- [11] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. Van der Merwe. Design and implementation of a routing control platform. In *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*, pages 15–28, Berkeley, CA, USA, 2005. USENIX Association.
- [12] B.-G. Chun, S. Ratnasamy, and E. Kohler. Netcomplex: a complexity metric for networked system designs. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 393–406, Berkeley, CA, USA, 2008. USENIX Association.
- [13] A. Ermolinskiy and S. Shenker. Reducing Transient Disconnectivity using Anomaly-Cognizant Forwarding. In *ACM SIGCOMM HotNets VII*, 2008.
- [14] P. Francois and O. Bonaventure. Avoiding transient loops during the convergence of link-state routing protocols. *IEEE/ACM Transactions on Networking*, 15(6):1280–1932, Dec 2007.
- [15] P. Francois, C. Filsfils, J. Evans, and O. Bonaventure. Achieving sub-second IGP convergence in large IP networks. *SIGCOMM Comput. Commun. Rev.*, 35(3):35–44, 2005.
- [16] P. Francois, M. Shand, and O. Bonaventure. Disruption-free topology reconfiguration in OSPF Networks. In *IEEE INFOCOM*, Anchorage, USA, May 2007. INFOCOM 2007 Best Paper Award.
- [17] J. J. Garcia-Luna-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Trans. Netw.*, 1(1):130–141, 1993.

- [18] G. Gibb, J. Lockwood, J. Naous, P. Hartke, and N. McKeown. Netpfgaan open platform for teaching how to build gigabit-rate network switches and routers. *Education, IEEE Transactions on*, 51(3):364–369, Aug 2008.
- [19] M. Goyal, G. Choudhury, A. Shaikh, K. Trivedi, and H. Hosseini. LSA correlation to schedule routing table calculations. Internet draft, draft-goyal-ospf-lsacorr-00.txt, Oct 2008.
- [20] U. Hengartner, S. Moon, R. Mortier, and C. Diot. Detection and Analysis of Routing Loops in Packet Traces. In *SIGCOMM IMW*, 2002.
- [21] G. Iannaccone, C. nee Chuah, R. Mortier, S. Bhattacharyya, and C. Diot. Analysis of link failures in an IP backbone. In *IMW*, 2002.
- [22] J. P. John, E. Katz-Bassett, A. Krishnamurthy, T. Anderson, and A. Venkataramani. Consensus routing: the internet as a distributed system. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 351–364, 2008.
- [23] D. Katz and D. Ward. Bidirectional Forwarding Detection. Internet draft, draft-ietf-bfd-base-07.txt, Jan 2008.
- [24] N. Kushman, S. Kandula, D. Katabi, and B. M. Maggs. R-BGP: Staying connected in a connected world. In *NSDI*, 2007.
- [25] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP Network Recovery using Multiple Routing Configurations. In *INFOCOM*, pages 23–29, 2006.
- [26] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica. Achieving convergence-free routing using failure-carrying packets. In *SIGCOMM*, pages 241–252, 2007.
- [27] A. Li, X. Yang, and D. Wetherall. SafeGuard: Responsive Routing with Consistent Forwarding. Technical Report DUKE-CS-TR-2008-04, Duke, 2008.
- [28] R. Mahajan, N. T. Spring, D. Wetherall, and T. E. Anderson. Inferring link weights using end-to-end measurements. In *Internet Measurement Workshop*, pages 231–236, 2002.
- [29] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, and C. Diot. Characterization of Failures in an IP Backbone Network. In *INFOCOM*, 2004.
- [30] A. P. Markopoulou, F. A. Tobagi, and M. J. Karam. Assessment of VoIP Quality over Internet Backbones. In *INFOCOM*, 2002.
- [31] M. Motiwala, N. Feamster, and S. Vempala. Path Splicing: Reliable Connectivity with Rapid Recovery. In *ACM SIGCOMM HotNets VI*, 2007.
- [32] S. Nelakuditi, S. Lee, Y. Yu, Z.-L. Zhang, and C.-N. Chuah. Fast local rerouting for handling transient link failures. *IEEE/ACM Trans. Netw.*, 15(2):359–372, 2007.
- [33] P. Pan, G. Swallow, and A. Atlas. Fast Reroute Extensions to RSVP-TE for LSP Tunnels. RFC4090, May 2005.
- [34] B. Rajagopalan, J. Luciani, and D. Awduche. IP over Optical Networks: A Framework. RFC3717, Mar 2004.
- [35] M. Seaman. Exact hop count. 802.1aq draft, aq-seaman-exact-hop-count-1206-01.pdf, Dec. 2006.
- [36] A. Shaikh and A. G. Greenberg. Experience in black-box ospf measurement. In *Internet Measurement Workshop*, pages 113–125, 2001.
- [37] M. Shand and S. Bryant. IP Fast Reroute Framework. Internet draft, draft-ietf-rtgwg-ipfrr-framework-08.txt, Feb. 2008.
- [38] N. T. Spring, R. Mahajan, D. Wetherall, and T. E. Anderson. Measuring ISP topologies with rocketfuel. *IEEE/ACM Trans. Netw.*, 12(1):2–16, 2004.
- [39] W. Xu and J. Rexford. Miro: multi-path interdomain routing. In *SIGCOMM '06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 171–182, New York, NY, USA, 2006. ACM.
- [40] X. Yang and D. Wetherall. Source selectable path diversity via routing deflections. In *SIGCOMM*, pages 159–170, 2006.
- [41] Z. Zhong, R. Keralapura, S. Nelakuditi, Y. Yu, J. Wang, C.-N. Chuah, and S. Lee. Avoiding transient loops through interface-specific forwarding. In *IWQoS*, pages 219–232, 2005.

APPENDIX

A. PSEUDO-CODE FOR FORWARDING ALGORITHM

Figure 10 shows the pseudo-code for the forwarding algorithm described in § 3.4. Router n_i receives a packet pkt and makes its forwarding decision based on pkt 's destination $pkt.dst$, its enhanced cost $pkt.\overline{cost}$, and its mode $pkt.mode$.

B. DETERMINISTIC ALGORITHM TO GENERATE LINK NOISES

In this section we present a deterministic algorithm that can generate link noises and guarantee the noises are collision-

```

1: if  $pkt.mode == normal$  then
2:   if  $pkt.cost \neq n_i.cost$  then
3:      $pkt.mode \leftarrow escort$ 
4:   end if
5:   if  $pkt.cost \leq n_i.cost$  then
6:      $n_{i+1} \leftarrow forwarding\_table.lookup(pkt.dst)$ 
7:      $pkt.cost \leftarrow n_{i+1}.cost$ 
8:     forward  $pkt$  to  $n_{i+1}$ .
9:   else
10:     $n'_{i+1} \leftarrow APD.lookup(pkt.dst, pkt.\overline{cost})$ 
11:    if  $n'_{i+1} == null$  then
12:      discard  $pkt$ .
13:    end if
14:     $pkt.\overline{cost} \leftarrow n'_{i+1}.\overline{cost}$ 
15:    forward  $pkt$  to  $n'_{i+1}$ .
16:  end if
17: else if  $pkt.mode == escort$  then
18:   if  $pkt.cost == n_i.cost$  then
19:      $n_{i+1} \leftarrow smallest\_enhanced\_cost(pkt.dst)$ 
20:      $pkt.cost \leftarrow n_{i+1}.cost$ 
21:     forward  $pkt$  to  $n_{i+1}$ .
22:   else
23:     $n'_{i+1} \leftarrow APD.lookup(pkt.dst, pkt.\overline{cost})$ 
24:    if  $n'_{i+1} == null$  then
25:      discard  $pkt$ .
26:    end if
27:     $pkt.cost \leftarrow n'_{i+1}.\overline{cost}$ 
28:    forward  $pkt$  to  $n'_{i+1}$ .
29:  end if
30: end if

```

Figure 10: The forwarding algorithm without optimization

free. The algorithm can be used in place of randomly generating the noises as described in the main part of the paper.

We first formalize the collision-free condition. Assume between a source node s_i and a destination node d_i there lies a set of paths P_i , which includes both the shortest paths and the alternative ones. P_i can be split into several non-overlapping subsets such that the paths in each subset has the same normal cost. The subset with only one path is already uniquely identified by its normal cost and thus we only consider the subsets with more than one path. We refer to each of such subset as an Equal Cost Path Set (ECPS). For each pair of source and destination there might exist multiple ECPS, and over all pairs of nodes we assume there are in total K ECPS: E_1, E_2, \dots, E_K . Our goal is to compute link noises which can discriminate the paths in each set E_i .

We denote the link noise configuration as a vector c where $c(x)$ is the noise configured to link l_x . For each path p , a link usage vector v_p can be defined as follows: $v_p(x) = 1$ if link $l_x \in p$, otherwise $v_p(x) = 0$. Thus path p 's total noise cost is $v_p^T \cdot c$. The collision-free condition can be translated into the following statement: for any $1 \leq i \leq K$, $v_{p_1}^T c \neq v_{p_2}^T c \pmod{2^k}$, $\forall p_1, p_2 \in E_i$ and $p_1 \neq p_2$. For simplicity we drop the notation of modulo in the following description. Each inequality condition can be further converted to a standard form: $e_j^T c \neq 0$, where e_j equals to $v_{p_1} - v_{p_2}$, and j is the index over all p_1 and p_2 pairs and all i . Thus in the end we have N vectors e_j ($1 \leq j \leq N$) and $N = \sum_{i=1}^K |K_i|(|K_i| - 1)/2$. The collision-free condi-

tion is then re-phrased as follows using e_j vectors: for any $1 \leq j \leq N$, $e_j^T c \neq 0$. Our goal is to find a noise configuration c which satisfies the condition.

We show how our algorithm works in an induction fashion. First given e_1 , it is easy to find c_1 such that $e_1^T c_1 \neq 0$. Now assume c_j satisfies that for any $j' \leq j$, $e_{j'}^T c_j \neq 0$, our next step is to find a c_{j+1} which satisfies the inequalities up to the $(j+1)$ th one. If $e_{j+1}^T c_j \neq 0$, we can trivially set $c_{j+1} = c_j$. In case $e_{j+1}^T c_j = 0$ we need to adjust the noise vector in the following way:

First find any non-zero element of e_{j+1} . Assume the index of such element is x . We will then introduce a minor change to $c_j(x)$: $c_{j+1}(x) = c_j(x) + \delta$. All other elements remain the same. It is obvious to see as long as $\delta \neq 0$, $e_{j+1}^T c_{j+1} \neq 0$. Thus our goal is to find the appropriate δ such that it will also satisfy the j' th inequality for all $j' \leq j$. If $e_{j'}(x) = 0$, the change of δ will not affect the inequality, and thus based on the assumption we know the inequality $e_{j'}^T c_{j+1} \neq 0$ holds. For j' that $e_{j'}(x) \neq 0$, we compute $t_{j'} = -e_{j'}(x)e_{j'}^T c_j$, and have:

$$e_{j'}^T c_{j+1} = e_{j'}^T c_j + e_{j'}(x)\delta \quad (1)$$

$$= e_{j'}(x)[\delta + e_{j'}(x)e_{j'}^T c_j] \quad (2)$$

$$= e_{j'}(x)[\delta - t_{j'}] \quad (3)$$

(2) is true because $e_{j'}(x)$ can only be either 1 or -1. Thus we only need to find a $\delta \in [1, 2^k - 1]$ such that $\delta \neq t_{j'}$ for all $j' \leq j$ and $e_{j'}(x) \neq 0$. This ensures all the first j inequalities still hold for c_{j+1} . On the other hand, due to the pigeonhole principle, as long as $2^k - 1 > j$, we can always find a viable δ . The search for δ can be done by first sorting all $t_{j'}$ s in at most $O(j)$ time.

By induction, in the end the algorithm guarantees to find a collision-free noise configuration $c = c_N$ which satisfies all N inequalities, as long as $2^k - 1 \geq N$. The complexity of the algorithm is $O(K^2 D^4 M)$ where M is the number of links. In practice the algorithm can be much more efficient as many elements of e_j are zero. A simple benchmarking study on several real and inferred topologies (details omitted) reveals that the algorithm can finish within reasonable time (a few minutes for the largest tier-1 topology), and k is no larger than 20.

C. PROOFS OF PROPERTIES

Property 1 *Packets will follow regular shortest paths, including equal cost paths, to reach their destinations in the normal mode when the network is in stable state.*

PROOF. We prove by induction. First assume a router n_i receives a packet pkt in normal mode, and the packet's normal cost $pkt.cost$ matches its own cost $n_i.cost$. According to the forwarding algorithm in § 3.4, n_i can forward the packet to the next hops computed using the regular costs only. Suppose n_i chooses the next hop n_{i+1} , whose local

cost is $n_{i+1}.cost$. The packet's normal cost is then updated as $pkt.cost' = n_i.cost - l_{n_i \rightarrow n_{i+1}}.cost$. Since the network is stable, $n_i.cost = n_{i+1}.cost + cost_{n_i \rightarrow n_{i+1}}$. Thus again $n_{i+1}.cost$ matches with $pkt.cost'$, and $pkt.cost' < pkt.cost$. By induction, we know the packet's normal cost decreases monotonically and will arrive at the destination eventually in the normal mode. Further, because each node can forward the packet to any of the regular next hops, the packet can then go through any of the equal cost paths. \square

Property 2 *If enhanced path costs are distinct, during the routing transition period in which only one network element changes its status and the network is not partitioned, a packet will be forwarded to its destination in either the normal or the escort mode.*

PROOF. From the proof of Property 1 we know that if no cost inconsistency is detected, the packet will arrive at its destination in the normal mode. If at least one cost inconsistency is detected, the escort bit of the packet will be turned on by the first router n_i whose local normal cost does not match the packet's normal cost. n_i 's upstream router n_{i-1} must be in a different state as n_i . Given the assumption that only a single network element changes its status, each router can only be in one of two states: either the router has learned the change, or it has not. We first show: no matter which state n_i is in, when it forwards the packet to its next hop n_{i+1} the packet's enhanced cost $pkt.cost$ identifies a valid path that reaches the destination without passing any of the failed or restored component.

If n_i is in the state that the topology contains more components (*i.e.* a "larger" topology), it must discover that its local normal cost is smaller than the packet's normal cost. In this case, the packet's enhanced cost, which is set by n_{i-1} as the remaining path enhanced cost from n_i to the destination, must identify a valid path that does not go through the changed component. This is because n_{i-1} must have the smaller topology which does not include the changed component. According to the algorithm, n_i needs to look up its alternative path database (APD) with the packet's enhanced cost. Such lookup will always succeed because the APD must contain a path bypassing the changed component. n_i then keeps forwarding the packet along the valid path chosen by n_{i-1} , and the packet's cost is updated to identify the remaining path from its next hop n_{i+1} to the destination.

If n_i owns the topology that contains less components, it must discover that its local normal cost is larger than the packet's normal cost. In this case, either n_i is adjacent to a failure (no available local paths), or n_i 's regular shortest paths do not pass the changed component. In either case, the path n_i uses according to the algorithm will not pass the changed component, and the packet's enhanced cost is set to identify such a valid path.

Once the packet leaving n_i in the escort mode has an enhanced cost that identifies a valid path, it only remains to show all the downstream routers starting from n_{i+1} will con-

sistently forward the packet along the same path identified by the cost. According to the algorithm we can see that for an escort mode packet a router can only forward it to a path with exactly the same enhanced cost as the one carried by the packet. Since Different paths have different enhanced costs this ensures the packet can only be forwarded along the same path, unless an APD lookup fails and the packet is dropped. However, lookup failure cannot happen here because if the router's cost does not match the packet's cost, it must own a larger topology and its APD must then contain the path bypassing the changed component. \square

Property 3 *A packet will not be trapped in a micro-loop without being discarded.*

PROOF. By trapped in a micro-loop, we mean that if all routers stop updating their forwarding tables after forming a loop, a packet will not escape the loop until its TTL expires. We prove this property by showing a packet cannot traverse the same node for three times. First we show a packet cannot traverse the same node for twice while still in the normal mode. This is because if no cost inconsistency is detected the packet's normal cost must decrease monotonically, and should always match the local normal cost at each hop. Thus if a packet visits the same node twice, its normal cost cannot match the node's local cost twice, thus for the second time and packet must be already in the escort mode or be turned into the escort mode by the node. Next we show that once a packet is in the escort mode after it leaves a node, it cannot return to the node any more. This is because an escort packet will always follow a consistent path until it reaches the destination or is discarded because of an APD lookup failure (see the last part of the proof of Property 2). A node cannot appear twice on a path, and thus the packet will not loop.

It should be noted that if the optimization (§ 6) technique is in use, the proof only holds when there is a single element change. If single element changes its state, the optimization technique does not change forwarding, and thus Property 2 still holds, which implies Property 3. However, if multiple elements change their status simultaneously, a packet in the escort mode may not follow consistently along a loop-free path because routers will not drop packets upon APD lookup failures. \square