Scalable Multi-Failure Fast Failover via Forwarding Table Compression

Brent Stephens
UW-Madison

Alan L. Cox, Scott Rixner
Rice University

ABSTRACT

In datacenter networks, link and switch failures are a common occurrence. Although most of these failures do not disconnect the underlying topology, they do cause routing failures, disrupting communications between some hosts. Unfortunately, current $1:1$ redundancy groups are only partly effective at reducing the impact of these routing failures. In principle, local fast failover schemes, such as OpenFlow fast failover groups, could reduce the impact by preinstalling backup routes that protect against multiple simultaneous failures. However, providing a sufficient number of backup routes *within the available space* provided by the forwarding tables of datacenter switches is challenging. To solve this problem, we contribute a new forwarding table compression algorithm. Further, we introduce the concept of *compression-aware routing* to improve the achieved compression ratio. Lastly, we have created Plinko, a new forwarding model that is designed to have more easily compressible forwarding tables. All optimizations combined, we often saw compression ratios ranging from 2.10× to 19.29×.

CCS Concepts

•**Networks** → **Routing protocols; Network reliability;** *Network simulations;*

Keywords

Local Fast Failover; Forwarding Table Compression

1. INTRODUCTION

As datacenter networks continue to grow in size, so has the likelihood that at any instant in time one or more switches or links have failed. Even though these failures may not disconnect the underlying topology, they often lead to routing failures, stopping the flow of traffic between some of the hosts. Ideally, datacenter networks would instantly reroute the affected flows, but today's datacenter networks are, in fact, far from this ideal. For example, in a recent study of datacenter networks, Gill *et al.* [11] reported that, even though "current data center networks typically provide $1:1$ redundancy to allow traffic to flow along an alternate route," in the median case the redundancy groups only forward 40% as many bytes after a failure as they did before. In effect, even though the redundancy groups help reduce the impact of failures, they are not entirely effective.

Local fast failover schemes, such as OpenFlow fast failover groups [25], could potentially bridge this effectiveness gap by allowing for a controller to preinstall backup routes at switches that provide high-throughput forwarding in the face of multiple simultaneous failures. Moreover, if a local fast failover scheme is implemented at the hardware level, it can react near-instantaneously to link failures, and if it allows for arbitrary backup routes, then it could be able to build routes that outperform existing redundancy groups. To the best of our knowledge, this paper contains the first description of a method for implementing arbitrary fast failover groups for Ethernet networks in hardware.

In particular, in this paper, we explore the use of local fast failover with backup routes that are t-resilient, *i.e.*, even given t arbitrary link failures, the backup routes guarantee connectivity between all hosts that are connected in the underlying topology while simultaneously ensuring that packets do not enter routing loops [9]. However, the principal challenge in closing the effectiveness gap is providing a sufficient level of resilience within the forwarding table size constraints of datacenter switches/routers, which currently have between ∼1–40Mb of TCAM state [4, 15, 14].

Unfortunately, increasing resilience in the local fast failover schemes considered in this paper causes a rapid increase in the amount of TCAM state consumed by forwarding table entries, which limits the level of resilience that is practical today. Assuming all-to-all routes and that all state is distributed evenly across the switches V, the additional number of forwarding table entries required by these schemes to move from $(t-1)$ to t-resilience is roughly $|D|^2 * ap^{(t+1)}/|V|$, where D is the set of destinations, ap is the average path length, and $|D|^2 * ap/|V|$ is an estimate of the number of 0-resilient (primary) forwarding table entries at each switch. This is because 1) routes are, on average, ap hops long, 2) hop-by-hop forwarding uses a forwarding table entry per hop of each route, and 3) these schemes build a new backup route for each $(t-1)$-resilient forwarding table entry to provide t-resilience.

To solve this problem, we explore different approaches to achieving forwarding table compression given resilient routing. First, we contribute a new forwarding table compression algorithm. Most existing forwarding table compression algorithms are designed for prefix classifiers and firewalls [22], so they are either not applicable to or not effective at compressing resilient forwarding tables.

Next, we observe that only forwarding table entries that share both the same output *and* the same packet modification action can be compressed, which implies that the achievable compression ratio is limited by the number of unique (output, action) pairs in the

SOSR '16, March 14 - 15, 2016, Santa Clara, CA, USA

ISBN 978-1-4503-4211-7/16/03…$15.00

DOI: http://dx.doi.org/10.1145/2890955.2890957

Fast Failover Scheme	Arbitrary Paths	Arbitrary Topos	t-resilient for any t
Fat Tire [27]	✓	✓	×
Packet re-cycling [21]	×	✓	×
ADST-Res [7]	×	✓	×
EDST-Res [37, 7, 32]	×	✓	×
F10 [20]	×	×	×
DDC [19]	×	✓	✓
Borokhovich *et al.* [2]	×	✓	✓
MPLS-FRR [26]	✓	✓	✓∗
FCP [18]	✓	✓	✓∗
Plinko [34]	✓	✓	✓∗

Table 1: A comparison of local fast failover schemes. A $*$ indicates that the state required by the scheme is dependent on topology size, so the resilience of the scheme is limited by forwarding table state.

table. Thus, our next two contributions are explicitly conceived as ways to increase the number of common outputs and actions. First, we introduce the concept of *compression-aware routing*, which increases the number of entries with common forwarding table outputs. Because only backup routes share existing outputs and only a subset of the backup routes can be in use at the same time, aggregate throughput is not impacted. Second, we introduce Plinko, a new forwarding model in which all entries in the forwarding table apply the same action. (A preliminary description of Plinko appeared in a recent workshop [34].)

While many other local fast failover schemes have been introduced, we only compare Plinko against MPLS Fast Re-route (FRR) [26] and FCP [18] because these are the only schemes that also allow for both arbitrary routes and levels of resilience[1], which is shown by Table 1. We believe these properties are important to allow a network to use OpenFlow fast failover groups to outperform existing redundancy groups without conflicting with other network policies. However, in order to provide these properties, forwarding table entries in MPLS-FRR and FCP forwarding tables that protect against different failures need to add different labels to packets to prevent forwarding loops. In contrast, we have deliberately designed Plinko to apply the same action *to every packet*.

In summary, the contributions of this paper are as follows:

New forwarding table compression algorithm: In our experiments, Plinko achieved compression ratios ranging from 2.10× to 19.29× for 4-resilient routes on the full bisection bandwidth fat tree [24] topologies that we evaluated.

Compression-aware routing: By selecting backup routes that are designed to be compressible, we show that it is possible to reduce forwarding table state without impacting resilience or performance.

Plinko: We show that Plinko is often 6–8× more scalable than the other two forwarding models. With all optimizations combined, 4-resilient Plinko is able to scale to networks with about 10K hosts while only requiring 1 Mbits of TCAM state.

Analysis of Resilience: There has been no prior analysis of how the likelihood of routing failures changes given different levels of resilience and topologies. In this paper, we present the first such analysis. We find that even low levels of resilience are highly effective at preventing routing failures as linear increases in resilience reduce the probability of a routing failure exponentially. This is because analysing the resilience of forwarding tables is analogous to analysing the time complexity of Quicksort. Although there exists a set of failures or list of elements that can excite the worst-case behavior, respectively, on average, we would expect t-resilience to protect against more than t failures when averaged over all possible sets of failures just as the average-case time complexity of Quicksort is smaller than the worst case. For example, forwarding tables that are 4-resilient to edge failures were able to provide four 9's of protection (99.99% of routes did not fail) against 16 random edge failures on all of the topologies we evaluated. We provide insight into this result by introducing a closed form approximation for the probability of routing failures that only relies on a single topological property. Lastly, we show that the t-resilient routing schemes in this paper have minimal impact on stretch and throughput.

The remainder of this paper is organized as follows. Section 2 discusses resilience. Section 3 discusses forwarding table compression, and Section 4 discusses the implementation of resilient switches. Section 5 presents our methodology, and Section 6 evaluates the performance and state requirements of resilience given realistic datacenter topologies. We tie up loose ends in Section 7 and discuss related work in Section 8. Finally, we conclude in Section 9.

2. FORWARDING RESILIENCE

OpenFlow fast failover groups, which have been a part of the OpenFlow standard since version 1.1 [25], enable a network controller to preinstall backup routes at switches so that when a link local to a switch fails, the switch can reroute packets using only information known locally and contained in a packet's headers. Ideally, OpenFlow fast failover groups could be used to allow a network controller to install routes in the network that not only satisfy the currently specified high-level security, routing, and performance policies but also allow network operators and applications to specify a desired level of routing fault tolerance as part of a high-level policy. In this paper, we are concerned with how to feasibly build (compile) a fault tolerant routing policy into fault tolerant routes given the limited size of switch forwarding tables.

In particular, we are primarily interested in forwarding table resilience [9] because it is a promising metric for allowing for operators and applications to specify a level of fault tolerance. In this section, we first define t-resilience, then we motivate our interest in t-resilience by presenting an expected-case analysis of the effectiveness of edge resilience. In effect, we show that in addition to providing easy to understand worst-case behavior, t-resilience also provides good expected-case behavior. After that, we present an algorithm for constructing t-resilient routes, and we conclude by discussing three different forwarding functions that can support t-resilient routing for arbitrary routes, which is important for providing backup routes than can satisfy a network's bandwidth and latency requirements despite failures.

2.1 Definition

If a network provides t-resilient routes for a tenant or traffic class, then it guarantees the routes installed in the network satisfy two properties [9]. First, it guarantees that, for any possible set of failures F of size $|F| = t$, there exists a route from any node v to any destination d in the forwarding function if there exists a path in the underlying topology after removing F. Second, a t-resilient forwarding function guarantees that no packets can ever enter an infinite forwarding loop, even if the network is partitioned.

Since the first property quantifies the level of fault tolerance, its purpose is clear. However, the motivation for the second property, which guarantees loop freedom, may not be so obvious. In datacen-

[1]Although they do not meet our requirements, some existing schemes are complementary to our approach. For example, both DDC [19] and Borokhovich *et al.* [2] can provide fast failover that can tolerate any number of failures given a *fixed* amount of state, so they may be useful for high-priority traffic where performance is less important than availability.

ter networks, high-throughput forwarding is important. The second property is included because infinite forwarding loops can significantly impair throughput, potentially leading to network-wide packet loss and congestion, even if looping packets are dropped from the network using a TTL mechanism [8, 5]. As an interesting aside, without the inclusion of the second property, trivial yet impractical solutions such as completely random routing would qualify as ∞-resilient.

While resilience is a metric that can be used to specify the fault tolerance of different types of network elements, including links, switches, and services, we focus on link/edge resilience. This is because we found that link resilience is effective at protecting against switch failures, but switch resilience is not entirely effective given link failures.

2.2 Expected-Case Analysis

Given a t-resilient network that has more than t failures, our expectation is that the network shows a graceful degradation as the number of failures increases because only routes that encounter $t+1$ failures lose connectively, barring a partition. Further, we also expect that increases in t will exponentially decrease the probability of routing failure because they also exponentially increase the number of backup routes. To support these claims, we introduce a new approximation for the probability of a routing failure given a topology, level of resilience, and a number of routing failures, and we use simulations to both validate our approximation and further motivate resilience.

2.2.1 Approximation

In our approximation, we refer to the forwarding pattern, the collection of the forwarding tables of every switch in the network, as fp, a path from a source v to a destination d as p_v^d, the average path length between destinations as ap, the set of edges as E, and the set of edge failures as F_e. The average path length between destinations is the only topological property that we use. Additionally, we assume that failures occur uniformly and that the routes are uniformly spread over the edges and vertices. Also, if shortest path routing is not used, ap should be replaced with the average length of the routes.

Equation 1 presents our approximation:

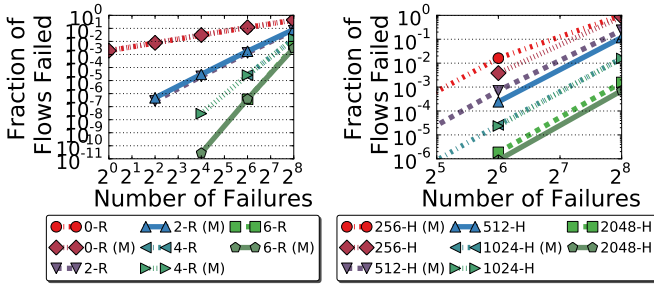
$$Pr(p_v^d \notin fp) = \begin{cases} 0 & \text{if } |F_e| \leq t \\ 1 & \text{if } |E| - |F_e| < ap \\ (1 - \frac{\binom{|E|-|F_e|}{ap}}{\binom{|E|}{ap}})^{t+1} & \text{otherwise} \end{cases} \quad (1)$$

The idea behind our approximation is: if all edges are equally likely to be in a route, then the probability there is not a routing failure is the number of sets of ap edges that do not include a failed edge divided by the number of possible sets of ap edges. This gives the probability of routing failure. Given t-resilience, a total of $t+1$ (backup) routes must all encounter a failure, so the probability that a route fails is raised to the power $t+1$. An interesting implication of this approximation is that the probability of a routing failure decreases exponentially with respect to increases in t.

We have also approximated the probability of flow failures given vertex failures instead of edge failures. However, this approximation is similar to Equation 1, so we omit it.

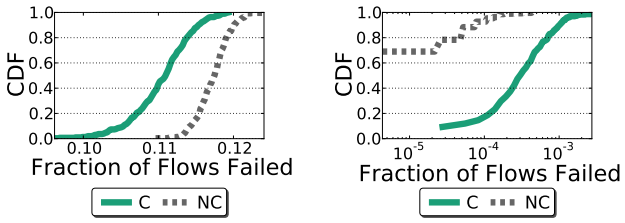
2.2.2 Simulations

To verify the accuracy of Equation 1, we simulated different sizes and modes of routing failures according to the methodology described in Section 5. Figure 1 presents the results of our experiments on full bisection (B1) fat trees, with *-R and *-H representing differing levels of resilience and numbers of hosts, respectively, and (M) marking the output of our approximation. Figure 1a not only shows that our approximation is accurate across a range of levels of resilience on a generalized fat tree (EGFT) [24] with 1024 hosts, but also that linear increases in resilience provide orders of magnitude decreases in the probability of routing failure.



(a) 1024-H EGFT (B1) **(b) 4-R EGFT (B1)**

Figure 1: Expected Effectiveness of Edge Resilience



(a) 0-R Routes **(b) 4-R Routes**

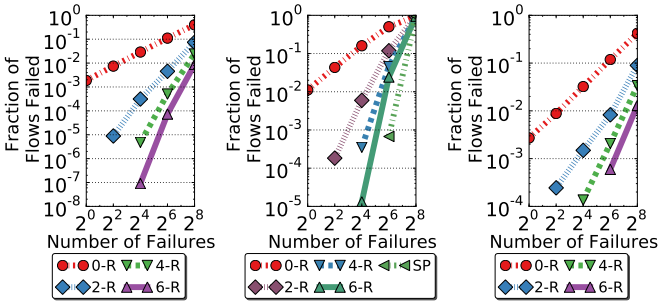
Figure 2: CDF of the fraction of failed routes given 64 edges failures on a 1024 host EGFT (B1)

To show the impact of topology size, Figure 1b presents the probability of routing failure given a varying number of failures and 4-resilience on EGFT topologies. Because the most simultaneous link failures reported in Microsoft data centers was 180 [11], we chose to use the same sizes of edge failures for each topology size. Normalizing the failure sizes to the topology could lead to unrealistic failure sizes for large topologies. Because of this, the probability of routing failure decreases as topology size increases. This figure illustrates the accuracy of our approximation across a range of topology sizes. In the end, the accuracy of our approximation in our experiments is always within an order of magnitude and frequently within a factor of 2–3×.

Correlated Failures.

In practice, we would expect that simultaneous failures are likely to be physically related. To evaluate this case, we randomly selected sets of connected edges for failure. Although we found that correlated failures cause roughly an order of magnitude more host pairs to experience a routing failure, we found that the probability of failure is still low.

To illustrate the effect of correlated and non-correlated failures, Figure 2 shows a CDF of the fraction of routes that failed with both 0-resilience (0-R) and 4-resilience (4-R) given correlated (C) and non-correlated (NC) failures on a 1024 host EGFT topology. Interestingly, correlated failures have less impact than non-correlated failures for the 0-R routes, although both still have a significant number of failed routes. However, with 4-R routes, correlated failures cause more routes to fail than uniform random failures. This is because 0-R routes fail if even a single edge on its path fails and



(a) EGFT (B1) (b) EGFT (B6) (c) 99%tile EGFT (B1)

Figure 3: Resilience and Correlated Failures

Algorithm 1 – Routing for t-resilient forwarding

1. Build primary (non-resilient) routes:
2. Iteratively protect routes against any single additional failure each round: For round i in $\{1, \ldots, t\}$, do:
   (a) Consider every edge e and switch sw that forwards out e in all the paths p built in round $i-1$.
      i. Build a backup path for p assuming that edge e failed, if one exists. This path must not use either e or any edge that p assumes failed.
      ii. Install a route at sw that uses p and modifies the packet to add e to the set of failures identified by the packet.

correlated failures are likely to touch fewer routes. 4-R routes, on the other hand, only fail given multiple failures, so about 70% of all 64 edge failures do not even cause a single routes to fail, while all correlated failures of 64 edges caused some routes to fail, albeit at between two and four orders of magnitude fewer than without resilient routes.

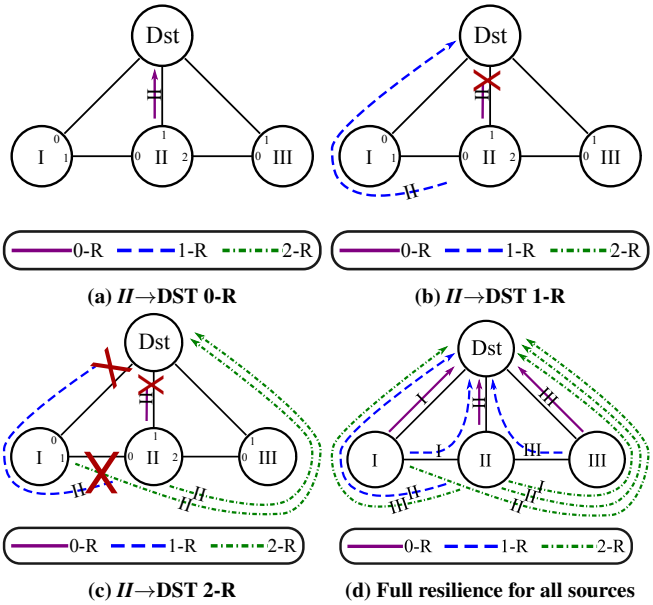
Figure 3 illustrates this by showing the average and 99$^{\text{th}}$ percentile fraction of routes that failed given correlated failures for varying levels of resilience (*-R) on the 1024-host topologies, with B1 and B6 representing 1 : 1 and 1 : 6 bisection bandwidth ratio topologies, respectively. Even though correlated failures impact roughly an order of magnitude more routes, Figure 3a shows that the probability of failure is still low. Although smaller topologies are more likely to be impacted by failures, Figure 3b shows that resilience is still effective on the B6 topology, and 6-resilience is not all that far from reactive shortest path routing (SP). Lastly, Figure 3c, which looks at the 99$^{\text{th}}$ percentile failures, shows that even though the outlying failures impacted roughly an order of magnitude more routes than the average failures, low levels of resilience still prevent far more routing failures than no resilience (0-R).

In conclusion, our approximation and simulations demonstrate the effectiveness of even seemingly low-levels of resilience. Given that we find low levels of resilience to be highly effective in preventing routing failures, we are now primarily concerned with feasibly implementing resilience.

2.3 Forwarding Table Construction

In order to build resilient forwarding tables, we assume a conventional SDN environment that uses a (logically) centralized controller to compute all routes and program each switch's forwarding tables. In addition to building and installing policy-based routes, guaranteeing resilience can also require the controller to install rules at switches that add opaque forwarding labels to packets so that it can also install rules in downstream switches that match on these labels in addition to the status of the ports local to the switch.

To build t-resilient forwarding tables, we use an SDN controller that uses Algorithm 1. This algorithm uses a very simple approach



(a) *II*→DST 0-R (b) *II*→DST 1-R

(c) *II*→DST 2-R (d) Full resilience for all sources

Figure 4: Example Resilient Routes

to route construction where the actual path selection is left up to an arbitrary policy. At the start, Algorithm 1 establishes all-to-all communication by installing a primary route for all source/destination pairs of hosts. Unless one of the edges in the path fails, each switch along the route will use a packet's header to look up an output port in its forwarding table, which is programmed by a controller. The remainder of the routing algorithm iteratively increases resilience by installing new backup routes to protect the paths built in the previous round against the failure of any one additional edge. For every edge in the routes built in the previous round, a backup route that matches the header of packets following the route but requires the original output port to be down is added to the switch local to the edge. Because all possible failures of an additional edge are considered in each round, by the end of round t, paths have been built that are resilient to all possible failures of t edges. While the algorithm is presented in terms of edges, the algorithm can also be used to protect against the failure of vertices, *i.e.* switches.

Figure 4 illustrates the routing algorithm by showing resilient routes, focusing on *II*→*Dst*. In the figure, routes are represented as arrows and the level of resilience is shown by the line style and color, using 0-R, 1-R, and 2-R for forwarding patterns that are resilient to 0, 1, and 2 failures, respectively. Figure 4c is particularly interesting because there are two 2-R entries. This is because failure information is only known locally by the forwarding hardware, *i.e.*, switches only learn of the failure of local links, and the 1-R path [*II-I*, *I-Dst*] can fail at either the *II-I* link or the *I-Dst* link when there is still an operational path to *Dst*. Therefore, if links *II-Dst* and *I-Dst* have failed, switch *II* will still attempt to forward packets along the [*II-I*, *I-Dst*] path based on its local information. Only after packets reach switch *I* will the failed link *I-Dst* be observed, at which point the packet needs to be forwarded along the path [*I-II*, *II-III*, *III-Dst*]. While this leads to path stretch, the stretch tends to be minimal in practice and is unavoidable given that we want to provide local fast failover without any *a priori* failure knowledge.

2.4 Forwarding Models

Because Algorithm 1 allows for arbitrary backup routes, it should be able to build policy compliant routes that achieve high throughput. Thus, the primary difficulty in outperforming existing redundancy groups is in providing a large enough level of re-

silience. In this section, we discuss three forwarding models that are capable of implementing Algorithm 1 given arbitrary levels of resilience, ignoring forwarding table size constraints.

The principal difficulty in forwarding packets along the routes built by Algorithm 1 is in identifying which backup route a packet is currently following and what failures the packet has already encountered so that infinite forwarding loops can be avoided. Two features are sufficient to support t-resilient routing: 1) forwarding tables that not only match on packet headers, but also on the local port status, and 2) packets with a header that identifies the set of failures already encountered by the packet.

Although no datacenter switches currently implement the first feature in hardware, the motivation for it is clear: matching on the current port status allows for near instantaneous use of backup routes in the case of link failure. However, the reason for the second feature may not be so clear. Although Feigenbaum *et al.* [9] proved that it is always possible to protect against any single link failure given a forwarding function that only matches on the destination, the input port, and the current port status, Gill *et al.* found that 41% of failures involve multiple links [11]. Thus, we would like a forwarding function that can protect against the failure of more than one link. However, if an additional header is not used to identify failures, Feigenbaum *et al.* also proved that there exists a topology for which there exists a level of resilience that cannot be provided. The intuition behind this result is that, if a packet encounters multiple failures, a switch cannot determine the exact set of failures encountered by the packet, which can cause backup routes to form forwarding loops. With an additional packet header, it becomes possible to build routes that protect against any possible set of failures on any topology [18, 34].

In this paper, we consider three forwarding models that use additional headers to identify failures. The first of these forwarding models, FCP, accumulates the IDs of the failed links encountered by a packet in a header [18]. More precisely, FCP can either accumulate edge IDs or use a label to identify a set of failed edges, which can save space.

With MPLS-FRR [26], we use a unique ID for every new route to provide t-resilience, with the ID assigned to a packet changing when a failure is encountered. In fact, a given network path may have multiple IDs because an ID also encodes the route that was being used by the packet prior to each encountered failure. Thus, the set of encountered failed edges can be inferred from the ID.

Finally, Plinko is a new model that retains the full path taken by a packet to identify the set of failures encountered by that packet [34]. The failures can be identified because the retained path includes the packet's source at each hop, and, from a given source–be it the original switch or the switch local to a failure–there is only one possible path given the local failures. This is due to the resilient routing algorithm. Because it is deterministic, it causes all forwarding table entries for a packet (reverse path) at a switch to use different paths and protect against a different set of failures. This implies that the path a packet has taken can be used to infer the original source, the forwarding table entries used to forward the packet, and thus the exact set of failures the packet has encountered. (A more detailed proof that Plinko is t-resilient can be found in a tech report [35].)

In Plinko, the full path taken by a packet is retained by having every switch that the packet traverses push the packet's input port number onto a list contained in the packet header, which could even be implemented with switches that support stacked VLAN tags. The primary benefits of Plinko are that the same action is applied to every packet and that packets with common reverse path prefixes may have encountered the same failures and have the same output path, which allows for rules to be compressed.

10GbE Switch	Release Year	TCAM	Exact Match
HP Procurve 5400zl	2006	285 Kbits	918 Kbits
Intel FlexPipe (FM6000)⋆	2011	885 Kbits	5 Mbits
BNT G8264	2011	∼1.15 Mbits	∼5.6 Mbits
Metamorphosis⋆	2013 †	40 Mbits	370 Mbits

Table 2: 10 Gbps TOR Switch Table Sizes. A ⋆ indicates that the switch is reconfigurable, and a † indicates that the switch is academic work and not a full product.

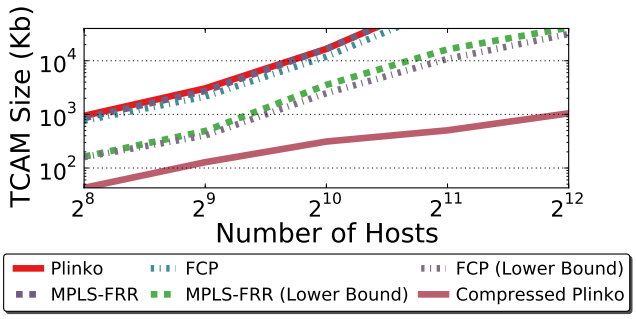


Figure 5: TCAM Sizes for 6-Resilience

3. FORWARDING TABLE COMPRESSION

We are now primarily concerned with implementing useful levels of resilience given limited TCAM state for implementing forwarding tables, which can be thought of as arrays of forwarding table entries that match on the same packet headers (DMAC, DIP, SIP, etc.) and local switch state (port status) and specify an output port and/or packet modification actions to apply. First, this section motivates the need for forwarding table compression for resilience. Next, it describes our new forwarding table compression algorithm. Lastly, it introduces compression-aware routing.

3.1 Motivation

Prior work assumes that state explosion would limit hardware local fast failover to all but the smallest networks or uninteresting levels of resilience [18]. Here, we explore the validity of this assumption. We built 6-resilient routes between all host-pairs on different sized full bisection bandwidth fat tree topologies to understand the exact state requirements of resilience (see Section 5 for the details of our methodology). To see if the state requirements are prohibitive, we compare the results against the TCAM sizes found in existing and proposed datacenter switches, which we present in Table 2. The results of this experiment, shown in Figure 5, confirm the assumptions of previous work. Providing 6-resilience requires over 40 Mbit of TCAM state on a network with just two thousand hosts, more state than is available in any switch. This result clearly motivates forwarding table compression.

Unfortunately, forwarding table compression is challenging because, as was previously mentioned, modifying packets in transit may be necessary for t-resilience. Only forwarding table rules that share the same output and packet modification action can be compressed, which is problematic for FCP and MPLS-FRR, which identify sets of failed edges and routes with a unique id, respectively. Even when looking at only the modifying entries *after* applying previous work that uses network virtualization to reduce forwarding table state [23], which is shown in the "(Lower Bound)" lines in Figure 5, the total state required can be limiting.

On the other hand, Plinko applies the same modification to each packet—pushing the input port onto a list in the packet—so there is no such limitation. Surprisingly, this subtle architectural difference

is crucial to enabling forwarding table compression. We frequently saw compression reduce state by over 95%, which is shown in Figure 5 as the line "Compressed Plinko."

Although MPLS-FRR uses unique IDs for each route, it may seem reasonable to try to adapt FCP so that it applies the same action to each packet. One way to do this would be to have a switch mark a packet with *all* local failures instead of just the failures specific to forwarding the packet, *i.e.*, the specific set of local failures the forwarding table entry protects against. However, t-resilient routes must now be built to match on any of many possible different failures marked in a packet's headers by the switches the packet traverses, causing an explosion in the number of entries. While it is reasonable to try to compress these entries, allowing for each switch to independently mark the failures of all of its edges leads to prohibitively large packet headers.

3.2 Forwarding Table Compression

Compression can be used to reduce the TCAM state consumed by a forwarding table. As long as two forwarding table entries have the same output and packet modification operations, wildcard matching can be used to combine multiple forwarding table entries into a single TCAM entry by masking off the bits in which the entries differ. Such forwarding table compression is particularly powerful as TCAMs allow for overlapping entries, with the priority of an entry determining which entry actually matches a packet. This introduces an optimization problem: given a set of forwarding table entries, find a smaller set of potentially overlapping TCAM entries and priorities that are functionally equivalent. If the state reduction from compressing a resilient forwarding table is greater than the increase in TCAM state from matching on the resilient tag (or reverse path), then it is better to perform matching with a TCAM.

Our work is not the first to discuss this use of TCAMs, and this optimization problem has been well studied in the context of packet classifiers. However, most of the existing work is not applicable to resilient forwarding, because it is designed for prefix classifiers [22], and resilient forwarding table entries do not use prefix rules to match on the local port status. Similarly, XPath [13] requires being able to assign labels to each path, which does not fit with matching on a path in Plinko because assigning a label to each route simply devolves into MPLS-FRR if applied to Plinko. To the best of our knowledge, Bit weaving [22] is the only compression algorithm applicable to Plinko. Unfortunately, we found that it did not result in significant compression when applied to resilient forwarding tables, most likely because Bit weaving was designed for packet classifiers, whose table entries are different than those for resilience. Thus, we developed a new TCAM packing heuristic, which performs well in our experiments. While other effective compression algorithms potentially exist, their existence would further enable resilient forwarding tables.

Specifically, our new TCAM packing heuristic is based on four observations. First, not all reverse paths are possible, and a controller knows all of the possible reverse paths if it knows all of the installed routes. This reduces the restrictions on which forwarding table entries can be safely merged. Second, no entries in an uncompressed resilient forwarding table overlap, so they can be reordered. This is because they all either match on different tags used to identify failures or require the output port of another entry to be failed. The third observation is that higher priority entries in a compressed table have to be finer in granularity so as to avoid matching packets intended for a lower priority entry. Fourth, there is a greater chance for state reduction by choosing to aggregate the largest set of rules that share a common output path first. Based on these observations, our algorithm first sorts the rules in descending order based on the size of the set of rules that share the same output path and action. In other words, for each (output, action) pair in the old table, the algorithm builds the set of entries that use the pair, then considers each entry in each set, starting with the largest set. This order helps ensure that the entries with a larger potential for reducing forwarding table size are considered first.

Dst	Rev Path	Port Status	OP
D	[1, -1, -1, -1]	*0*1	4
D	[1, 2, -1, -1]	*0*1	4
D	[-1, -1, -1, -1]	00*1	4
D	[-1, -1, -1, -1]	10**	1
D	[2, -1, -1, -1]	10**	1
D	[3, -1, -1, -1]	10**	1
D	[-1, -1, -1, -1]	*1**	2
D	[2, -1, -1, -1]	*1**	2
D	[2, 4, 2, -1]	*1**	2
D	[2, 4, 4, -1]	*1**	2
D	[3, -1, -1, -1]	*1**	2
D	[1, -1, -1, -1]	*1**	2
D	[1, 2, -1, -1]	*1**	2

Table 3: Part of a Plinko forwarding table

Dst	Rev Path	Port Status	OP
D	[1, *, -1, -1]	*0*1	4
D	[-1, -1, -1, -1]	00*1	4
D	[*, -1, -1, -1]	10**	1
D	[*, *, *, *]	*1**	2

Table 4: Table 3 compressed

Given this processing order, our algorithm then greedily attempts to merge entries, *i.e.* masking off the bits in which they differ, into rules in a new TCAM, which we initialize as empty. To do so, we maintain a working set of new TCAM entries for each (output, action) pair, which we also initialize as empty. For each old TCAM entry in order, the working set is greedily searched for an entry in the working set to merge with the old entry such that the resulting merged entry *does not overlap* with any committed TCAM entry. In other words, if any of the already considered forwarding table entries with different (output, action) pairs would match the new entry, then the merge is not performed, but, if there is no conflict, then the merge is performed, updating the entry in the working set. If there are multiple entries in the working set that the old entry can be merged with, we pick the one with the minimum Hamming distance from the old entry. However, if all entries in the working set cause conflicts, then the current old entry is added as-is to the working set. Lastly, once all entries for an (output, action) pair are considered, the entries in the working set are committed to the new TCAM at a priority higher than that of any entry currently committed to the new TCAM.

To help illustrate this algorithm, Table 3 and Table 4 show part of a Plinko forwarding table before and after compression. The forwarding tables are for hop-by-hop routing on a four port switch, output ports are labeled as (OP), and reverse paths that are shorter than the longest reverse path are padded with "-1", even though this padding may not be necessary in hardware. Table 3 shows that, although only two entries do not share a reverse path, none of uncompressed entries overlap due to their port status.

When compressing Table 3, our algorithm would first consider all of the entries that use output port 2. Because no rules have been added to the forwarding table yet, all of these entries can be compressed into a single entry. Similarly, none of the entries that use port 1 overlap with the first rule in the table, so they can also be compressed into a single entry. However, if all of the entries for output port 4 were compressed into a single entry, the compressed entry would match some of the packets intended for output port 1, which would lead to an incorrect forwarding table. Instead, only two of the entries for output port 4 can be merged.

3.3 Compression-Aware Routing

Forwarding table compression will always be constrained by the

Table	Input Fields	Match Actions	Modified Fields	Explanation
Island In Table	InPort	Drop	∅	Distinguish between packets based on whether the input port is internal or not
Security Table	*	Drop	Security Tag	Perform arbitrary packet matching to either drop packets or add a security tag
Encap/MPath Table	Dst	∅	VDst, ECMP ID, ResTag, RevHopCount RevHops	Add the necessary default packet headers to external packets. Optionally converts overlay/end-host addresses into underlay addresses, selecting among multiple possible paths if applicable
SrcFwd/Local Table	((Dst × VDst) | FwdHop[0]) × bm	OutPort	FwdHopCount, FwdHops	Source routing: checks to see if the current next hop is operational. If not, discard the forward source route Network Virtualization: Checks to see if the virtual destination is local and the port of the physical destination is up
ResFwd Table	(VDst | Dst) × (RevHops | ResTag) × bm × SecTag	OutPort, Drop	FwdHopCount, FwdHops, **ResTag**	Choose an output edge or path as a function of the destination, the resilience tag or reverse path, and the port status bitmask
SrcUpdate Table	∅	∅	Fwd/Rev Source Route	Pop off the current FwdHop and Push on the new RevHop
Island Out Table	OutPort	∅	Entire new header	Decapsulate packets that leave the island

Table 5: Description of the tables in Figure 7.

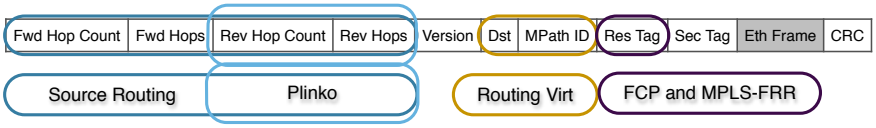


Figure 6: A Packet Header for Resilient Forwarding

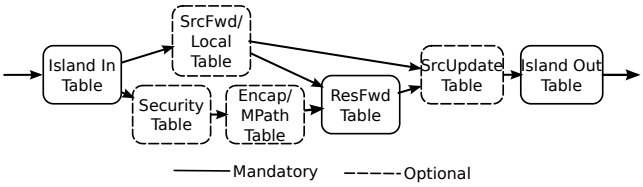


Figure 7: An Example Resilient Forwarding Pipeline

number of unique output paths in the forwarding table. However, our resilient routing algorithm allows for arbitrary paths, and, on datacenter topologies, the algorithm frequently chooses between multiple equivalent paths. To exploit this property to improve compressibility, we propose *compression-aware routing*, which attempts to choose routes that increase the compressibility of the forwarding tables.

The compression-aware routing heuristic first checks if any of the existing routes for the current destination avoid the failures that the current route being built protects against. If so, the most common of such paths is chosen. If no such path exists, non-compression-aware routing continues and chooses a new path that avoids the necessary failures.

Because each backup route depends on the paths used by previous routes, the order in which the routes are chosen can have an impact on compression. There are two reasonable orderings, which are akin to bread-first search (BFS) and depth-first search (DFS) graph traversal: the BFS ordering builds all possible t-resilient routes before building any $(t+1)$-resilient routes, while the DFS ordering recursively protects the first unprotected hop of the most recent route until the desired level of resilience is achieved.

We chose to use the BFS ordering for two reasons. First, lower-level resilient routes are more important for performance. Given, t failures, there are likely to be more routes that hit $i < t$ failures than those that hit exactly t failures. Further, the 0-resilient routes are especially important to network performance because these are the primary routes. Second, there are likely to be more $(t+1)$-resilient routes than there are t-resilient routes, so considering them later may open up more opportunities for reusing existing routes.

4. IMPLEMENTATION

Because implementing resilience may require new packet formats and the ability to match on the local port status in hardware, we now discuss how to implement our resilient forwarding models. First, we describe the implementation of a simple t-resilient switch for the MPLS-FRR and FCP forwarding models. Next, we discuss a resilient forwarding pipeline that includes more features and then discuss the implementation of source routing and network virtualization, network features that, as a side effect, reduce forwarding state [23]. To the best of our knowledge, this section is the first description of a method for implementing arbitrary fast failover groups for Ethernet networks in hardware.

4.1 A Simple Resilient Switch

As discussed in Section 2, to implement local fast failover at the hardware level requires a forwarding table that can match on the current port status of the local switch. However, we are not aware of a switch that currently allows for matching on the port status as a p-bit value, given a p-port switch. Fortunately, we believe that this change is easily implementable. In particular, this port bitfield would be maintained based on PHY information and used as an input to the forwarding table pipeline, where it is then best suited for matching with a TCAM.

While matching on the current port status could also use exact match memory, doing so would cause a prohibitive explosion in state. Most forwarding table entries protect against a handful of failures and do so regardless of the state of the other ports on the switch. If we have a single 1-resilient backup route that requires one edge to be up and one to be down on a 64-port switch, ignoring the other 62 ports, using a bitmask with wildcards can specify this match in a single TCAM entry, while using exact match would require 2^{62} separate entries to cover all possibilities.

Given a port bitmask for matching, it is simple to implement resilient FCP and MPLS-FRR. In its basic form, the switch pipeline would consists of a single table that uses exact match memory to match on the destination and resilience tag and a TCAM for matching on the port bitfield. Each entry would then specify an output port and, if a failure was encountered, write the new resilience tag to the packet. Similarly, Plinko could be implemented in the same way, except that instead of writing a new resilience tag over the old one, the input port would be pushed onto the packet header similar to how VLAN stacking is done on switches today.

4.2 Resilient Logical Forwarding Pipeline

Additional switch features can reduce forwarding table state. To illustrate this, Figure 6 presents an example packet header, Figure 7 presents an example resilient packet processing pipeline, and Table 5 describes the functionality of each table in the pipeline, referencing fields in the packet header, a packet's input port InPort, its output port OutPort, and the port bitmask of the switch *bm*. Together, Figure 7 and Table 5 are similar to P4's Table Dependency Graphs (TDGs) [3].

Although all of these features may not be currently available, recent developments have led to switches with both reconfigurable packet parsers and reconfigurable match tables that support a multitude of generic packet matching and modification actions (RMT [4] and FlexPipe [15]). Given such switches, adding these features should be simple. However, these features are not limited to reconfigurable switches and could also be implemented on an FPGA or ASIC.

4.3 Source Routing

Source routing, where packets contain a full path in a header, reduces forwarding table state in proportion to the average path length of the network topology because forwarding table state is no longer stored at the intermediate switches along a packet's path. However, this reduction in state comes at the cost of increased packet header overhead. To reduce this overhead, we reuse an existing architecture for source-routed Ethernet that only uses a single byte per hop of a source route. Typically, this only incurs an overhead of 1–2% or less [30]. Specifically, an Axon source route is a list of switch port numbers, one for each switch along the path. Because Plinko matches on the reverse path of a packet to provide resilience, Plinko benefits from the Axon's compact source route not just from reduced packet header overhead but also from reduced forwarding table state.

Although Axons were originally implemented in an FPGA, implementing the Axon protocol with a reconfigurable switch is possible by using two small additional logical tables, which are labeled as the SrcFwd and SrcUpdate tables in Figure 7. These tables attempt to forward via the source route and update the source route, respectively. The SrcFwd table matches on the next forward hop in the source route and the current port status. This table is small, containing just one entry per switch port. Each entry simply checks to see if the output port specified by the source route is operational. If it is, then it is used. Otherwise, the forward source route is discarded, and packet matching continues in the resilient forwarding table to find an alternate route. The SrcUpdate table is even simpler as it applies the same increment, decrement, push, and pop operations required to modify the forward and reverse source routes to all packets.

4.4 Network Virtualization

Lastly, network virtualization can also be used to reduce state [23]. Most hosts are only attached to one or two switches, while top-of-rack (TOR) switches connect to many hosts and many switches. This leads to the switch level topology being smaller than the host level topology. However, using multiple paths is especially important when forwarding on the switch topology so as to prevent all hosts on a switch from using the same path. Thus, encapsulating packets from (virtual) end-hosts and routing on the switch topology reduces state proportional to the number of (virtual) hosts per switch divided by the degree of multipathing.

The forwarding table pipeline in Figure 7 includes two tables to support network virtualization: the Encap/MPath table and the Local table. The Local table is responsible for checking whether the virtual destination (VDst) is the local address and, if it is, forwarding the packet to the correct local port. The Encap/MPath table is responsible for the other half of virtualization, encapsulation. The table matches on an unencapsulated packet's physical destination and converts it into a virtual destination (VDst), optionally adding a tag for multipathing (MPath Tag). Although this table can be implemented as part of the TOR switch, we expect that it would commonly be implemented as part of a virtual switch.

Although we have added a table and tag for multipathing, this table could do more than simply perform ECMP. For example, this table would be most effective if used in conjunction with edge-based load balancing like Presto [12], which could maintain well-balanced traffic given failures that cause some resilient paths to be longer but not others.

5. METHODOLOGY

In evaluating resilience, we are primarily concerned with two key properties: the state required to implement the routes and the effectiveness of the resilient routes, both in terms of preventing routing failures and the performance of the routes. To understand these properties, we simulated routing on realistic datacenter topologies[2]. First, the simulations compute all-to-all routes for each destination, building the forwarding tables for each switch, and, in the case of e-way multipathing, this is repeated e times. These forwarding tables identify the state requirements. Further, we use the forwarding tables in conjunction with a workload to compute three metrics: the fraction of active routes that experienced failure given the level of resilient routing, the stretch of active routes that avoided failures, and the throughput achieved by all of the flows, which is computed using Algorithm 2 from DevoFlow [6].

We assume that all-to-all routes are installed in the network because this provides the worst case state for routing. In networks where not all hosts are allowed to communicate or routes are installed reactively, we expect that the state would be reduced proportional to the number of routes that are actually installed.

Additionally, we make assumptions about packet header fields and forwarding table state. We assume that Plinko, MPLS, and FCP all require a 72-bit wide exact match entry (MAC + VXLAN). Because we assume 64-port switches and we do not restrict the ports that may be used for backup routes, each MPLS and FCP entry requires 64 bits of TCAM state for the port bitmask, and each Plinko entry requires 64 bits for the port bitmask plus 8 bits ($> \log_2 64$) per hop in the reverse path. As part of future work, we are investigating relaxing these assumptions to reduce the number of TCAM bits used per entry for the port bitmask. Further, to remain independent from a single specific switch implementation, we assume that forwarding table entries require no overhead. In practice, internal fragmentation leads to additional state overhead, but prior work has pointed out that the additional cost is small [4].

When considering state, we report only the maximum state required by any switch in the network. Current datacenter topologies, including the two that we consider, are designed to be implemented with (close to) identical TOR switches, and reporting the maximum captures the required state given identical switches.

In the performance evaluation, we repeat the computation of our metrics 300 times for each number of failed edges. Also, we only present the stretch of the routes that encountered a failed network element yet still had a valid forwarding pattern route because routes that did not avoid a failure are guaranteed to have a stretch of 1.0 and would unfairly bias the results. The throughput results that we present are normalized to the maximum aggregate load on the topology, which is the number of hosts multiplied by the line rate.

[2]The simulation environment may be found at https://github.com/bestephe/res-sim

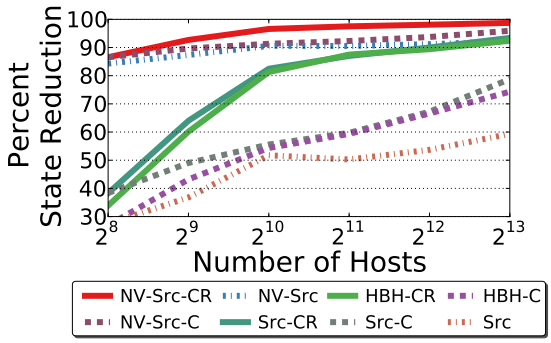


Figure 8: 4-R Jellyfish (B6) Compression Ratio

Multiple simultaneous network failures may be correlated, so we have implemented two different ways of selecting the set of failed network elements. The first way randomly chooses edges or vertices to fail. The second model, which attempts to mimic correlated failure, builds a set of failed edges or vertices by first selecting a single random element and then choosing elements that are neighbors of the already selected edges or vertices.

During the performance evaluation, we use a uniform random (uRand) workload to select the set of active flows. When computing the effect of failure, we use a uRand workload of degree 36, which is where each host connects as a source to 36 random destinations, to match the median degree of communication measured in a production datacenter [1]. Not all of the connections between servers carry bulk data, so we change the degree of communication to four when computing the throughput results.

We use two topologies in our evaluation to demonstrate that the results hold across different realistic datacenter topologies: the EGFT (extended generalized fat tree) [24] and the Jellyfish [31]. Although some network operators do not consider Jellyfish topologies to be practical due to their randomness, we use Jellyfish topologies to illustrate that our approach is truely topology independent. All of the topologies are built using 64-port switches and are sized for either a 1 : 1 (B1) or a more realistic 1 : 6 (B6) bisection bandwidth ratio, and we utilize prior work to minimize the number of switches in the network [33].

6. EVALUATION

There are a three important questions regarding the resilient forwarding models that we intend to answer. What is the cost of resilience? How effective are the optimizations (source routing, network virtualization, forwarding table compression, and compression-aware routing) at reducing the cost of resilience? Do any of the optimizations hurt performance, either by reducing throughput or increasing the probability of routing failure?

We find that the cost of naively implementing resilience, *e.g.* hop-by-hop routed FCP, may be prohibitively high. For example, providing 4-resilience on a 2048-host EGFT or a 4096-host Jellyfish given this model requires roughly 10Mbits of TCAM state. On the other hand, the optimizations to reduce forwarding table state for MPLS and FCP are effective, achieving an 84% or greater reduction in forwarding table state. However, Plinko significantly outperforms both of them due to the added benefits of forwarding table compression and compression-aware routing. With all optimizations combined, Plinko frequently achieves over a 95% reduction in forwarding table state, requiring only 1Mbits of TCAM state to implement 4-resilience on all of the 8192-host topologies. We discuss this further in Section 6.1.

	512-H (S/H)	1024-H (S/H)	2048-H (S/H)	4096-H (S/H)	8192-H (S/H)
0-R	1.00/1.25	1.00/1.63	1.00/2.32	1.00/1.25	1.00/1.16
1-R	1.17/1.56	1.11/1.95	1.19/3.85	1.08/2.14	1.05/2.21
2-R	2.25/2.00	2.14/2.77	2.83/6.34	3.09/4.16	3.28/4.51
4-R	2.67/2.10	4.60/7.14	12.08/22.07	17.08/16.21	15.78/19.29
6-R	3.13/3.33	10.32/16.38	34.94/73.75	56.47/63.23	

Table 6: EGFT (B1) Compression Ratio

	512-H (S/H)	1024-H (S/H)	2048-H (S/H)	4096-H (S/H)	8192-H (S/H)
0-R	1.02/1.02	1.00/1.27	1.00/1.94	1.00/2.02	1.00/2.12
1-R	1.02/1.39	1.15/1.93	1.41/2.75	1.53/3.15	1.56/3.41
2-R	1.22/1.84	1.60/3.03	2.01/4.36	2.24/4.74	2.34/4.95
4-R	2.03/3.45	3.59/7.03	4.99/10.26	5.32/11.67	5.58/12.09
6-R	3.70/6.10	7.69/16.42	11.21/24.98	13.38/28.86	14.18/

Table 7: Jellyfish (B1) Compression Ratio

We also found that none of the optimizations had any noticeable impact on the probability of routing failure or stretch. Further, only network virtualization impacted forwarding throughput, and this impact disappeared as long as 8-way or larger multipathing was used, which we discuss in Section 6.2. This implies that compression-aware routing significantly reduces forwarding table state without compromising on the goals of effectively protecting against failures and maintaining high network throughput.

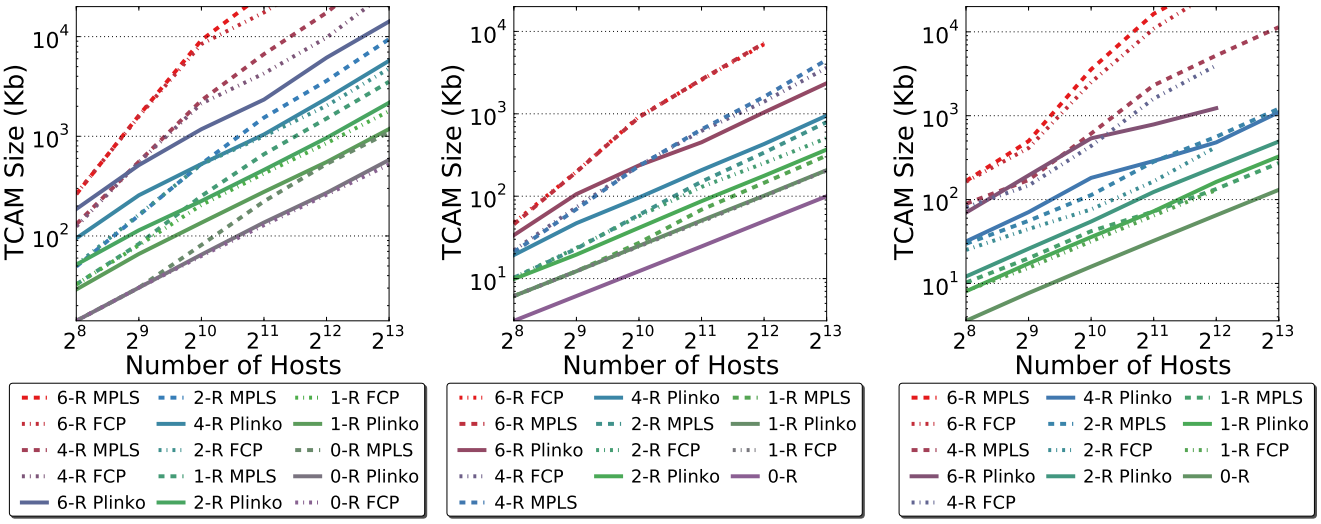
As part of this project, we have also evaluated protecting against both edge failures (edge-resilience) and vertex failures (vertex-resilience). However, we only present the results from edge-resilience given edge failures for two reasons. First, Gill *et al.* found that multiple switches failing at the same time in a datacenter is "extremely rare" [11], so providing edge-resilience is more desirable than vertex-resilience. Second, we found that low levels of both edge and vertex resilience (2-R) were as effective as reactive routing given vertex failures, but vertex-resilience did not provide any significant protection against edge failures. This result is particularly interesting because it challenges assumptions made in previous work on fault tolerance [20].

6.1 State

While we have seen the effectiveness of resilience in Section 2.2, if the state necessary to implement the forwarding tables is too large, the applicability of these results is limited. First we present results on the proportional usefulness of the different optimizations for reducing state, then we focus specifically on forwarding table compression. Lastly, we present the specific state requirements of the different forwarding models.

Figure 8 presents the percent reduction in forwarding table state over naive hop-by-hop routing (HBH) achieved by the different implementation variants, including 8-way multipathing network virtualization (NV), source routing (Src), and forwarding table compression with (CR) and without (C) compression-aware routing. Although source routing reduces state without compression, HBH routing surprisingly matches the performance of source routing with compression due to a proportional increase in compression for HBH routing. Another interesting point is that network virtualization and source routing on their own achieve a compression ratio between 84% and 92%. However, the addition of compression and compression-aware routing achieves up to a 99% reduction in state.

Although source routing and network virtualization are largely independent of the level of resilience, forwarding table compression is dependent on the level of resilience, a dynamic that is not captured in Figure 8. To illustrate this effect, we present Tables 6 and 7, which show the compression ratio achieved given Plinko with compression-aware routing and varying levels of resilience (*-R) on EGFT and Jellyfish topologies with a varying number of



(a) Hop-by-hop Routing (B6) **(b) Source Routing with Virtualization (B6)**

Figure 9: Jellyfish TCAM Sizes

Figure 10: Source Routing with Virtualization (EGFT B1)

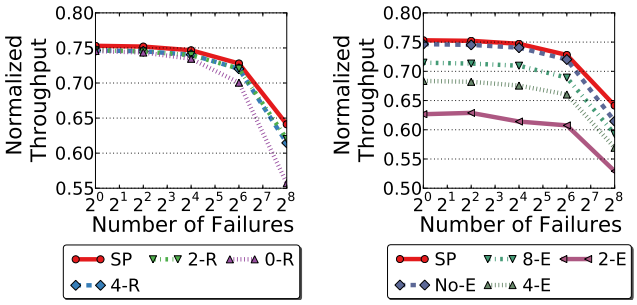
hosts (*-H). Besides showing that forwarding table compression is effective, these tables show two important trends: forwarding table compression increases with both increases in resilience and topology size. These trends are important because state is more likely to be a limiting factor given either larger networks or applications that desire increased resilience.

Figures 9 and 10 present the total state required to implement the three forwarding models for varying levels of resilience (*-R) on EGFT and Jellyfish topologies of differing size, respectively. Although these figures have many lines, we maintain two invariants to simplify interpretation. First, only Plinko results use solid lines, while FCP and MPLS, which perform very similarly, both use different styles of dashed lines. Second, the legend is sorted in decreasing order of the state required by each variant. Further, we omit the results from the Jellyfish (B1) and EGFT (B6) topologies because, surprisingly, the state requirements were almost the same as the other bisection bandwidth variant of the topology. Although we would expect state to increase due to the increase in the average path lengths of the B1 topologies, this increase in state is balanced by an increase in the number of switches in the network over which the state is distributed.

The most important trend that is visible in Figures 9 and 10 is that Plinko requires significantly less forwarding table state than FCP and MPLS, which require roughly similar forwarding table state, although FCP tends to perform better than MPLS as topology size increases. For example, 6-R Plinko consistently required less state than 4-R FCP or MPLS, and 4-R Plinko requires roughly the same amount of state as 2-R FCP and MPLS. We have previously seen that increasing resilience significantly reduces the probability of a routing failure (Figure 1), so this implies that Plinko is either able to provide significantly more routing protection given the same amount of state or the same level of protection on far larger topologies. Combining all optimizations, we expect that Plinko would be able to easily support 4-R routing on networks with tens of thousands of hosts within the 40 Mbits of TCAM available in Metamorphosis [4].

6.2 Performance Impact

Because compression-aware routing and network virtualization



(a) Resilience **(b) NV Multipathing**

Figure 11: EGFT (B1) Throughput Impact

can potentially hurt performance, we look at two metrics to evaluate the performance of resilient routes: stretch and throughput. Note that this section elides performance results for source routing and forwarding table compression because they do not impact path choice.

First, we found that compression-aware routing did not have any significant impact on throughput or resilience, despite significantly improving compression. Because of this, we omit figures on the impact of compression-aware routing.

Next, we found that resilience incurs little stretch. In all of the cases we evaluated, the median stretch was 1.0, and the tail of the stretch distribution is similarly small. This implies that most backup routes are the exact same length as the minimal length primary routes. Further, even when we considered the 99.9th percentile stretch given both random and correlated failures, the stretch only ranged from 1.0–2.5, with stretch increasing with topology size and resilience.

Because of these stretch results, we would expect that the throughput impact of resilience is also small. Figure 11a shows the normalized aggregate throughput given a uniform random workload on a 1024 host EGFT for both no-latency reactive shortest path routing (SP) and varying levels of hardware resilience (*-R). This figure validates our expectation. We see that even low levels of resilience achieve almost the same throughput as no-latency reactive routing.

However, unlike varying levels of resilience, moving to varying degrees of multipathing between endpoints given switch-level network virtualization can have a noticeable impact on throughput. Figure 11b shows the effect of varying degrees of random multipath (*-E) routing on the normalized throughput of a 1024 host EGFT topology, with the results holding for all of the forwarding models. In this figure, *SP* stands for reactive shortest path routing, and *No-E* refers to routing independently for each host as was performed in Figure 11a. From the throughput results, we see that 8-way ECMP reduces throughput by just under 5%, 4-way ECMP reduces throughput by under 10%, and 2-way ECMP reduces throughput by about 15%.

7. DISCUSSION

Although our resilient routing may appear to be an offline algorithm, implementing an online algorithm is simple. When a host joins the network, a controller only needs to compute and install 0-resilient routes for the host before connectivity is established, after which the routes for additional resilience can be computed and installed lazily. Computing 0-resilient routes is fundamental to all networks, so resilient routing does not incur unnecessary latency. Further, handling changes in the switch topology can be done safely and efficiently without the need for packet loss during updates through the addition a version field to packet headers (Figure 6), which can be used for consistent network updates [28]. We expect that, as soon as a controller detects a failure, it would use consistent updates to compute and install new routes that replace the existing operational but potentially non-optimal backup routes.

So far, we have only discussed forwarding table resilience assuming a single Ethernet segment, but it is also reasonable to implement resilience at each layer of a hierarchical network, *e.g.*, Ethernet segments connected by IP routers. In such a scenario, different forwarding functions could be used at each layer, *e.g.*, source-routed Plinko on the Ethernet network and hop-by-hop FCP at the IP layer.

Although most link-flapping events do not impact network connectivity [11], link-flapping can complicate resilience. A link that is persistently and constantly flapping could cause packet loss as routing also flaps between a usable backup route and the flaky link. Although this problem is beyond the scope of this work, which builds a system on top of the ability to correctly detect the existence of a failure in hardware, we do not believe this problem to be intractible. At a small cost to detection latency, the PHY or a hardware layer between the PHY and the forwarding table could add hysteresis when reporting a link as failed. Similarly, bidirectional failure detection [16] could also fix this problem. Regardless of the specific implementation, we believe that accurate and fast hardware failure detection is an important topic for future research.

8. RELATED WORK

While we have mainly discussed MPLS-FRR, FCP, and the work of Feigenbaum *et al.* [9] that formalized resilience, there is other significant work that should be discussed.

First, XPath [13] introduces a new forwarding table compression algorithm so as to allow for all desired paths to be preinstalled in a network. XPath's algorithm operates by first grouping paths into path sets, then it assigns labels to path sets so that, considering all forwarding tables, entries that share outputs at switches have labels that share prefixes so they can be compressed. Because MPLS-FRR and FCP have assignable labels, XPath can compress them, subject to the previously discussed lower bound. Because Plinko uses a path instead of a label, XPath is not applicable to Plinko.

Next, we have yet to consider some related work on routing failures for a variety for reasons. For example, ECMP, IP Fast Reroute [10], and Fat Tire [27] offer limited resilience. Packet recycling [21] and Borokhovich *et al.* [2] use inefficient paths. R-BGP [17] and F10 [20] rely on graph-specific properties. DDC [19] guarantees connectivity but at least temporarily incurs significant stretch and can suffer from forwarding loops, although an IP TTL may terminate packet forwarding. KF [36] also allows loops.

Although these projects do not meet of our goal of implementing efficient OpenFlow fast failover, many of them are complementary. For example, DDC [19], or data-driven connectivity, is a complementary project that provides *ideal forwarding-connectivity*, which guarantees that a packet will reach its destination as long as the network remains physically connected. DDC achieves this by performing provably safe link-reversals until the forwarding DAG converges, which, for n switches, is guaranteed to occur after $O(n^2)$ reversal operations. On one hand, DDC can temporarily incur significant stretch, and, in the case of a partition, packets will be looped until a TTL expires, so Plinko is preferable for routing failures it can prevent. On the other hand, DDC will always converge to a route given the destination is not partitioned, so if Plinko experiences a routing failure, it may be desirable to fall back on DDC for important traffic.

Similarly, Borokhovich *et al.* [2] describe an OpenFlow fast failover algorithm that guarantees delivery without looping packets by treating failover as a maze traversal problem. However, paths in their algorithm can be inefficient. Like DDC, their algorithm would also be effective as a secondary failover scheme if Plinko fails.

Fat Tire [27] introduces a new language for specifying fault tolerant routing that would be useful for programming a resilient network. However, as described, Fat Tire cannot provide t-resilience for all values of t, so Fat Tire benefits from our work.

Some recent work relies on disjoint spanning trees to provide resilience. For example, Elhourani *et al.* [7] introduced an algorithm that uses arc-disjoint spanning trees to provide up to $(k-1)$-resilient forwarding in a network that is k-connected. Similarly, Stephens and Cox [32] introduced DF-EDST resilience, which uses edge-disjoint spanning trees to provide deadlock-free local fast failover. While these approaches are promising, we did not consider them because they do not allow for arbitrary routes.

Lastly, Schiff *et al.* [29] have introduced a number of useful functions that rely on hardware fast failover groups and are complementary to the forwarding functions in this paper.

9. CONCLUSIONS

In this paper, we explore the feasibility of implementing local fast failover groups in hardware, even though prior work assumes that state explosion would limit hardware resilience to all but the smallest networks or uninteresting levels of resilience [18]. Specifically, this paper presents a number of practical advances that increase the applicability of hardware resilience. First, we have introduced a new forwarding table compression algorithm. Because forwarding table compression is limited by the number of unique (output, action) pairs in the table, we also introduced two ways to lower this bound. In order to increase the number of common output paths in a forwarding table, we introduce the concept of compression-aware routing, and we find that it is highly effective when combined with our compression algorithm, achieving compression ratios ranging from 2.10× to 19.29× given 4-resilient routes on EGFT topologies. In order to reduce the number of unique actions, which limits compression in both MPLS-FRR and FCP, we introduce Plinko, a new forwarding model that applies the same action to every packet. Finally, we have also considered us-

ing source routing and network virtualization to reduce forwarding table state. While source routing and network virtualization are effective on their own, reducing forwarding table state by as much as 92% on one topology, adding in forwarding table compression and compression-aware routing leads to a reduction of up to 99% on the same topology (*i.e.*, with forwarding table compression, the state required is over 8× smaller). Putting this all together, we expect that 4-resilient and 6-resilient Plinko will easily scale to networks with tens of thousands of hosts. In contrast, we expect that fully optimized FCP and MPLS-FRR could provide 4-resilience for topologies with 8192 hosts. Lastly, we find that even seemingly low-levels of resilience are highly effective at preventing routing failures, with 4-resilience providing four 9's of protection against 16 random edge failures on all of the evaluated topologies.

10. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *SIGCOMM*, 2010.

[2] M. Borokhovich, L. Schiff, and S. Schmid. Provable data plane connectivity with local fast failover: Introducing OpenFlow graph algorithms. In *HotSDN*, 2014.

[3] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 2014.

[4] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. A. Mujica, and M. Horowitz. Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN. In *SIGCOMM*, 2013.

[5] S. Casner. A fine-grained view of high-performance networking. In *Presented at NANOG22*, 2001.

[6] A. R. Curtis, J. C. Mogul, J. Tourrilhes, and P. Yalagandula. DevoFlow: Scaling flow management for high-performance networks. In *SIGCOMM*, 2011.

[7] T. Elhourani, A. Gopalan, and S. Ramasubramanian. IP fast rerouting for multi-link failures. In *INFOCOM*, 2014.

[8] K. Elmeleegy, A. L. Cox, and T. S. E. Ng. Etherfuse: An ethernet watchdog. In *SIGCOMM*, 2007.

[9] J. Feigenbaum, P. B. Godfrey, A. Panda, M. Schapira, S. Shenker, and A. Singla. On the resilience of routing tables. In *Brief announcement, 31st Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC)*, July 2012.

[10] P. Francois and O. Bonaventure. An evaluation of IP-based fast reroute techniques. In *CoNext*, 2005.

[11] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *SIGCOMM*, 2011.

[12] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella. Presto: Edge-based load balancing for fast datacenter networks. In *SIGCOMM*. ACM, 2015.

[13] S. Hu, K. Chen, H. Wu, W. Bai, C. Lan, H. Wang, H. Zhao, and C. Guo. Explicit path control in commodity data centers: Design and applications. In *NSDI*. USENIX Association, 2015.

[14] IBM BNT RackSwitch G8264. http://www.redbooks.ibm.com/abstracts/tips0815.html.

[15] Intel Ethernet Switch FM6000 Series - Software Defined Networking. http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ethernet-switch-fm6000-sdn-paper.pdf.

[16] D. Katz and D. Ward. RFC 5880 Bidirectional Forwarding Detection (BFD), June 2010.

[17] N. Kushman, S. Kandula, D. Katabi, and B. M. Maggs. R-BGP: staying connected in a connected world. In *NSDI*, 2007.

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

[19] J. Liu, A. Panda, A. Singla, P. B. Godfrey, M. Schapira, and S. Shenker. Ensuring connectivity via data plane mechanisms. In *NSDI*, April 2013.

[20] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson. F10: A fault-tolerant engineered network. In *NSDI*, 2013.

[21] S. S. Lor, R. Landa, and M. Rio. Packet re-cycling: eliminating packet losses due to network failures. In *HotNets*, 2010.

[22] C. R. Meiners, A. X. Liu, and E. Torng. Bit weaving: a non-prefix approach to compressing packet classifiers in TCAMs. *IEEE/ACM Trans. Netw.*, 20(2), Apr. 2012.

[23] J. Mudigonda, P. Yalagandula, J. C. Mogul, B. Stiekes, and Y. Pouffary. NetLord: a scalable multi-tenant network architecture for virtualized datacenters. In *SIGCOMM*, 2011.

[24] S. Ohring, M. Ibel, S. Das, and M. Kumar. On generalized fat trees. *Parallel Processing Symposium, International*, 0:37, 1995.

[25] OpenFlow switch specification, version 1.1.0. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf.

[26] P. Pan, G. Swallow, and A. Atlas. RFC 4090 Fast Reroute Extensions to RSVP-TE for LSP Tunnels, May 2005.

[27] M. Reitblatt, M. Canini, A. Guha, and N. Foster. FatTire: Declarative fault tolerance for software defined networks. In *HotSDN*, 2013.

[28] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker. Abstractions for network update. SIGCOMM, 2012.

[29] L. Schiff, M. Borokhovich, and S. Schmid. Reclaiming the brain: Useful OpenFlow functions in the data plane. In *HotNets*, 2014.

[30] J. Shafer, B. Stephens, M. Foss, S. Rixner, and A. L. Cox. Axon: A flexible substrate for source-routed Ethernet. In *ANCS*, 2010.

[31] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *NSDI*, April 2012.

[32] B. Stephens and A. L. Cox. Deadlock-free local fast failover for arbitrary data center networks. In *INFOCOM*, 2016.

[33] B. Stephens, A. L. Cox, W. Felter, C. Dixon, and J. Carter. PAST: Scalable ethernet for data centers. In *CoNext*, 2012.

[34] B. Stephens, A. L. Cox, and S. Rixner. Plinko: building provably resilient forwarding tables. In *HotNets*, 2013.

[35] B. Stephens, A. L. Cox, and S. Rixner. Plinko: Building provably resilient forwarding tables. Technical Report TR13-06, Department of Computer Science, Rice University, October 2013.

[36] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng. Keep forwarding: Towards k-link failure resilient routing. In *INFOCOM*, 2014.

[37] B. Yener, Y. Ofek, and M. Yung. Convergence routing on disjoint spanning trees. *Computer Networks*, 31(5):429 – 443, 1999.