On the Resilience of Routing Tables[1]

Joan Feigenbaum[2] Brighten Godfrey[3] Aurojit Panda[4]
Michael Schapira[5] Scott Shenker[4] Ankit Singla[3]

Technical Report YALEU/DCS/TR-1454
August 2012

Abstract: Many modern network designs incorporate ``failover'' paths into routers' forwarding tables. While such resilient routing tables are widely used in practice (*e.g.,* ECMP), there has been little theoretical work on their inherent power and limitations. We initiate the theoretical study of the conditions under which such *resilient routing tables* can guarantee delivery of packets and take the first steps in this research direction. We prove that routing tables can provide guaranteed resilience against single failures, as long as the network remains connected. We show, in contrast, that perfect resilience is not achievable in general.

Keywords: Internet routing, fault tolerance

[1] Brief announcement appeared in *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing* (PODC).

[2] Yale University, Computer Science Department, New Haven, CT 06520 USA, joan.feigenbaum@yale.edu, supported in part by NSF grant CCF 0728443.

[3] University of Illinois, Computer Science Department, Urbana, IL 61801 USA, {pbg, singla2}@illinois.edu, supported in part by NSF grant CNS 1017069.

[4] University of California, EECS Department, Berkeley, CA 94720 USA, {apanda, shenker}@eecs.berkeley.edu, supported in part by NSF grants CNS 1040838 and CNS 1117161.

[5] Hebrew University, School of Computer Science and Engineering, 91904 Jerusalem, Israel, schapiram@huji.ac.il. Work done in part as a postdoc at Yale and UC Berkeley, supported by NSF grant CNS 0331548.

On the Resilience of Routing Tables

Joan Feigenbaum (Yale), Brighten Godfrey (UIUC), Aurojit Panda (UC Berkeley),
Michael Schapira (Hebrew University), Scott Shenker (UC Berkeley), Ankit Singla (UIUC)

Abstract

Many modern network designs incorporate "failover" paths into routers' forwarding tables. In this paper, we formally address the conditions under which such *resilient routing tables* can guarantee delivery of packets.

1 Introduction

The core mission of computer networks is delivering packets from one point to another. To accomplish this, the typical network architecture uses a set of forwarding tables (that dictate the outgoing link at each router for each packet) and a routing algorithm that establishes those forwarding tables, recomputing them as needed in response to link failures or other topology changes. While this approach provides the ability to *recover* from an arbitrary set of failures, it does not provide sufficient *resiliency* to failures because these routing algorithms take substantial time to reconverge after each link failure. As a result, for periods of time ranging from 10s of milliseconds to seconds (depending on the network), the network may not be able to deliver packets to certain destinations. In comparison, packet forwarding is several orders of magnitude faster: a 10 Gbps link, for example, sends a 1500 byte packet in 1.2 μsec.

In order to provide higher availability we must design networks that are more resilient to failures. To this end, many modern network designs incorporate various forms of "backup" or "failover" paths into the forwarding tables that enable a router (or switch), when it detects that one of its attached links is down, to use an alternate outgoing link. We call these *resilient routing tables* since they embed failover information into the routing table itself and do not entail changes in packet headers (and so require no change in the low-level packet forwarding hardware). Because these failover decisions are purely local — based only on the packet's destination, the packet's incoming link, and the set of active incident links — they occur much more rapidly than the global recovery algorithms used in traditional routing protocols and thus result in many fewer packet losses.

While such resilient routing tables are widely used in practice (*e.g.,* ECMP), there has been little theoretical work on their inherent power and limitations. In this paper, we prove that starting with arbitrary loop-free routing tables, we can add forwarding rules to provide resilience against single failures in all scenarios (so long as the network remains topologically connected). We show, in contrast, that perfect resilience is not achievable in general (*i.e.,* there are cases in which no set of routing tables can guarantee packet delivery even when the graph remains connected). We leave open the question of closing the large gap between our positive and negative results. Other interesting open questions include exploring resilient routing tables in the context of specific families of graphs, randomized forwarding rules, and more.

The prior work closest to ours is Failure Insensitive Routing (FIR) [6]. FIR is also able to guarantee resilience to a single link failure, but is restricted to starting with shortest path routing tables. Our result on resilience to a single failure is more general, allowing the use of arbitrary (loop-free) routing tables in the absence of failure; and adding rules for tolerating one failure. In addition, we also demonstrate the impossibility of perfect resilience. FIR does not discuss a negative result of this nature.

While there is other significant past research on how to make routing more resilient, these efforts differ from our discussion here in one or more important respects. For instance, the literature discusses approaches that: (a) use bits in the packet headers to determine when to switch from primary to backup paths (this includes MPLS Fast Reroute) [1, 4, 9]; (b) encode failure information in packet headers to allow nodes to make failure-aware forwarding decisions [5, 8, 2] (work on fault-tolerant compact routing [10] also fits in this category); and (c) use graph-specific properties to achieve resilience [3]. Our own recent work [7] provides full resilience (*i.e.,* guaranteed packet delivery as long as the network remains connected), but modifies routing tables on the fly.

2 Model

The network is modeled as an undirected graph $G = (V, E)$, in which the vertex set consists of source nodes $\{1, 2, \dots, n\}$ and a *unique* destination node $d \notin [n]$. Each node $i \in [n]$ has a *forwarding function* $f_i^d : E_i \times 2^{E_i} \to E_i$, where E_i is the set of node i's incident edges. f_i^d maps incoming edges to outgoing edges as a function of which incident edges are up. We call an n-tuple of forwarding functions $f^d = (f_1^d, \dots, f_n^d)$ a *forwarding pattern.*

Consider the scenario that a set of edges $F \subseteq E$ fails. A *forwarding path* in this scenario is a route in the graph $H^F = (V, E \setminus F)$ such that for every two consecutive edges e_1, e_2 on the route which share a mutual node i it holds that $f_i^d(e_1, E_i \setminus F) = e_2$.

Intuitively, our aim is to guarantee that whenever a node is connected to the destination d, it also has a forwarding path to the destination. Formally, we say that a forwarding pattern f is *t-resilient* if for every failure scenario $F \subseteq E$ such that $|F| \leq t$, (1) if there exists some route from a node i to d in H^F then there also exists a forwarding path from i to d in H^F; and (2) all forwarding paths in H^F are loop-free. (Observe that the combination of these two conditions implies, intuitively, that a packet never enters loop en route to the destination or, alternatively, "gets stuck" at an intermediate node.)

3 Positive Result

3.1 High-Level Overview

We now present our main result, which establishes that for every given network it is possible to efficiently compute a 1-resilient forwarding pattern.

Theorem 3.1. *For every network there exists a 1-resilient forwarding pattern and, moreover, such a forwarding pattern can be computed in polynomial time.*

We prove Theorem 3.1 constructively; we present an algorithm that efficiently computes a 1-resilient forwarding pattern. We now give an intuitive exposition of our algorithm. We first orient the edges in G so as to compute a directed acyclic graph (DAG) D in which each edge in E is utilized. Our results hold regardless of how the DAG D is computed. An example network and corresponding DAG appear in figures 1(a) and 1(b), respectively. The DAG D naturally induces forwarding rules at source nodes; each node's incoming edge in D is mapped to its first active outgoing edge in D, given some arbitrary order over the node's outgoing edges (*e.g.*, node 4 in the figure forwards traffic from node 5 to node 2 if the edge to 2 is up, and to node 3 otherwise).

Intuitively, the next step is to identify a "problematic" node, that is, a node that is bi-connected to the destination in G but not in the partial forwarding pattern computed thus far, and add forwarding rules so as to "fix" this situation. Once this is achieved, another problematic node is

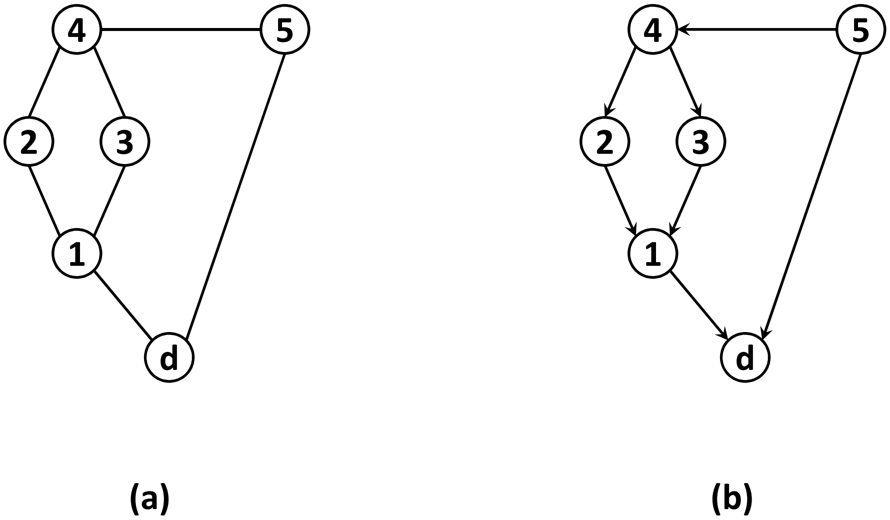

Figure 1: Illustration of high-level idea

identified and fixed, and so on. Observe that nodes 1-4 in the figure are all problematic. Observe also that adding the two following forwarding rules fixes node 4 (*i.e.*, makes node 4 bi-connected to the destination in the forwarding pattern): (a) when both of node 4's outgoing edges in D are down, traffic reaching 4 from node 5 is sent back to 5; and (b) when node 5's direct edge to the destination is up, traffic reaching node 5 from node 4 is sent along this edge. Thus, the algorithm builds the forwarding functions at nodes gradually, as more and more forwarding rules are added to better the resilience of the forwarding pattern.

Implementing the above approach, though, requires care; the order in which problematic nodes are chosen, and the exact manner in which forwarding rules are fixed, are important. Intuitively, our algorithm goes over problematic nodes in the topological order $<_D$ induced by the DAG D (visiting problematic nodes closer to the destination in D first), and when fixing a problematic node i, forwarding rules are added until a minimal node in $<_D$ whose entire sub-DAG in D does not traverse i is reached. We prove that this scheme outputs the desired forwarding pattern in a computationally-efficient manner.

3.2 Algorithm and Correctness

3.2.1 Algorithm

1. **Initialize.** $\forall e = (i,j) \in E, \forall T \subseteq E$, set $f_j^d(e,T) := \emptyset$.

2. **Construct DAG.** Construct a DAG $D = (V, E_D)$ (*e.g.*, using BFS/DFS) that is rooted in d and such that $\forall (i,j) \in E$, $(i,j) \in E_D$ or $(j,i) \in E_D$. D induces the following partial order $<_D$ over V: $\forall i,j \in V$, $i <_D j$ iff there is a route from j to i in D.

3. **Install DAG-based forwarding rules.** $\forall i \in V$, let E_D^i denote the set of i's outgoing edges in D. Choose an order over every E_D^i in some arbitrary manner. $\forall j \in V$ such that $e = (j,i) \in E$ and $\forall T \subseteq E$ such that $T \cap E_D^i \neq \emptyset$ set $f_i(e,T)$ to be the highest element in E_D^i that is not in T.

4. **Install additional forwarding rules.** While there exists a node q that is bi-connected to

d in G but not in $f^d = (f_1^d, \ldots, f_n^d)$ (that is, for which there do not yet exist at least two edge-disjoint forwarding paths to the destination in f^d) do:

(a) Choose i to be a minimal node (under $<_D$) that is bi-connected to d in G but not in $f^d = (f_1^d, \ldots, f_n^d)$.

(b) Choose j to be a minimal node (under $<_D$) such that (1) $i <_D j$ and (2) $\exists x \in V$ such that $(j, x) \in D$ and $i \nleq_D x$.

(c) Choose a simple route $R = (j = v_1, v_2, \ldots, v_k = i)$ from j to i in D.

(d) Set $c := k - 1$.

(e) While $(c > 1)$ and $(f_{v_c}^d(v_{c+1}, v_c) = \emptyset)$ do:

- $f_{v_c}^d(v_{c+1}, v_c) := (v_c, v_{c-1})$
- $c := c - 1$

(f) If $c = 1$, then $f_j^d(v_2, v_1) := (j, x)$.

3.2.2 Proof of Theorem 3.1

We now show that the algorithm outputs a forwarding pattern f^d as in the statement of Theorem 3.1. Consider a node i chosen in Step 4b of the algorithm.

Claim 3.2. *For every node i that is bi-connected to d in G but not in f^d there exists a node j such that (1) $i <_D j$; and (2) j has a directed edge in D to some node x such that $i \nleq_D x$.*

Proof. D spans all nodes in G and so there must exist a route R_1 from i to d in D. i is bi-connected to d in G and so there must also exist another route R_2 that is edge-disjoint from R_1 and is not in D (otherwise i would be bi-connected to d in D). Let j be a node on R_2 that has a route R_3 to d in D that does not go through i. We can now go over the nodes in R_3 (from j to d) one by one until we reach a node as in the statement of the claim. □

Consider an iteration of Step 4 of the algorithm. Recall that the node i chosen at that iteration is a node that (at that point in time) is bi-connected to d in G but not in f^d, and node j is a minimal node such that $i <_D j$ and that has a child x in D for which $i \nleq_D$.

We now show that following the execution of Step 4 the chosen node i becomes bi-connected to d in f^d and thus ceases to be "problematic". We handle two cases.

- **Case I:** In the execution of Step 4, c is decreased until $c = 1$. Observe that in this case i (that already has a route to d in D) has (at the end of that iteration) two edge-disjoint forwarding paths to d in f^d.

- **Case II:** c is decreased until a non-empty "entry" in f^d is reached. We now show that in this case, too, i has two edge-disjoint forwarding paths to d in f^d at the end of that iteration.

We now handle Case II above. For ease of exposition we illustrate our arguments on the specific (sub)network described in Figure 2. Recall that in Step 2 of the algorithm we construct a DAG D. The nodes and the red directed edges in the figure are some subgraph of D (the destination node d does not appear in the figure). Let i_1 and j_1 be the nodes i and j, respectively, chosen at some iteration q_1 of Step 2 of the algorithm, and let $R_1 = (j_1, \alpha, \beta, i_1)$ be the route R selected at iteration q. The blue directed edges in Figure 2 represent the changes to the forwarding functions made in the q_1'th iteration (along the route R_1). Let i_2 and j_2 be the nodes i and j, respectively, selected

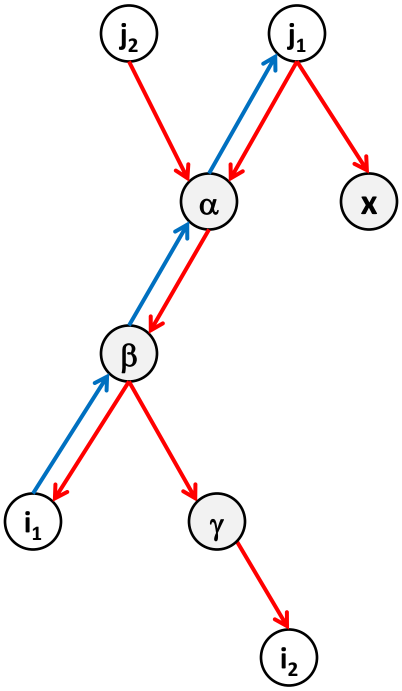

Figure 2: Illustration of proof idea

as some later iteration $q_2 > q_1$ of Step 2, and let $R_2 = (j_2, \alpha, \beta, \gamma, i_2)$ be the route R selected at iteration q_2.

Now, suppose that at the end of iteration q_1 node i_1 is not only bi-connected to d in G but also in f^d. We now show that at the end of the q_2'th iteration, i_2 too shall be bi-connected to d in both G and f^d. Consider the q_2'th iteration of Step 2. Observe that at the q_2'th iteration c is decreased until it reached the node α as, at that point, a non-empty entry in the forwarding function is reached. Hence, after the q_2'th iteration the route $(i_2, \gamma, \beta, \alpha, j_1, x)$ exists in the network. We now show that $i_2 \not\leq_D x$ and so there exists a route from i_2 to d that does not intersect its routes to d in D.

By contradiction. Suppose that $i_2 \leq_D x$. Recall that j_1 was chosen at iteration q_1 because it was a minimal node such that $i_1 <_D j_1$ and has a child x in D such that $i \not\leq_D x$. Hence, it must be that $i_1 <_D \gamma$ because otherwise β would have been chosen instead of j_1. Similarly, $i_1 <_D i_2$ because otherwise γ would have been chosen instead of j_1. This, combined with our assumption that $i_2 \leq_D x$ implies that $i_1 \leq_D x$ — a contradiction! The proof of the theorem follows.

4 Negative Result

We say that a forwarding pattern f is *perfectly resilient* if it is ∞-resilient — so that regardless of the failure scenario $F \subseteq E$, if there exists some route from a node i to the destination d in H^F then there also exists a forwarding path from i to d in H^F. To prove that forwarding patterns cannot always achieve perfect resilience, we first prove two properties of perfectly resilient forwarding patterns.

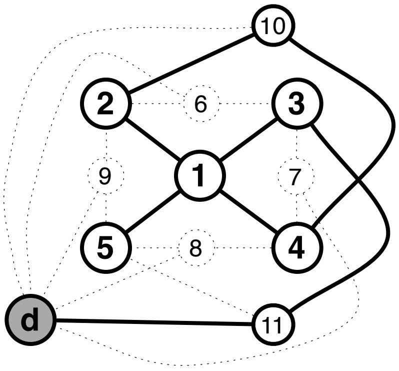

Figure 3: A failure scenario where perfect resilience is impossible.

Lemma 4.1. *For any edge e_{uv}, if v has any working path to the destination which does not use the edge e_{vu}, then v must not send a packet traveling $u \to v$ back to u.*

Proof. Assume the contrary, *i.e.*, there is a perfectly resilient forwarding pattern f with $f_v^d(e_{uv}, E_v) = e_{vu}$ and $\exists e_{vw} \in E_v, w \neq u$ such that w has a working path to d. Now, consider a scenario where all edges at u other than e_{uv} fail while v is connected to d through e_{vw}. A packet from u must be sent to v along e_{uv}. Then $f_v^d(e_{uv}, E_v) = e_{vu}$ implies v sends the packet back to u. u having no other live edges, sends it back to i, and we have a forwarding loop, even though there is a route to d. This contradicts the claim of f being perfectly resilient. □

Lemma 4.2. *A node i in the destination's connected component must route in some cyclic ordering of $E_i \backslash F$, i.e., an ordering of its edges with its neighbors $v_1, \dots, v_m$ such that $\forall j < m : f_i(v_j, E_i \backslash F) = v_{j+1}$ and $f_i(v_m, E_i \setminus F) = v_1$. For example, in figure* **??**, *node 1 may route packets from 2 to 3, packets from 3 to 4, from 4 to 5, and from 5 to 2.*

Proof. Let $nbrs(i)$ be the set of neighbors of node i. Assume the lemma is false, *i.e.*, there is a perfectly resilient forwarding pattern f such that f_i does not use such a cyclic ordering over $nbrs(i)$. Then f_i must have a smaller cyclic ordering which skips some neighbors $S \subset nbrs(i)$. Consider a scenario where $u \in S$ has a route to d, but all edges from nodes in $nbrs(i) \setminus S$ have failed, except those to i. The cyclic ordering in f over $nbrs(i) \setminus S$ ensures that packets loop over these nodes: packets starting at any node in $nbrs(i) \setminus S$ are sent to i which forwards them to some other node in the set (per the cyclic ordering). Any such node has no other connectivity except i, so the process repeats *ad infinitum*. However, each node in $nbrs(i) \setminus S$ does have a route to d through u. This contradicts the claim of f being perfectly resilient. □

Theorem 4.3. *There exists a network for which no perfectly resilient forwarding pattern exists.*

Proof. Consider the example network in figure (c). We show that after certain failures, no forwarding pattern on the original graph allows each surviving node in the destination's connected component to reach the destination. In figure (c), the surviving links are shown in bold; all other links fail.

By Lemma 4.2 above, node 1 has to route packets in some cyclic ordering of its neighbors. By the topology's symmetry, we can suppose w.l.o.g. that this ordering is $2, 3, 4, 5, 2$, *i.e.*, f^d is defined such that 1 forwards packets from 2 to 3, packets from 3 to 4, *etc.* Note that a forwarding loop is formed when a packet repeats a directed edge in its path (rather than just a node). To show that this occurs, consider the path taken by packets sent by 5 after the failures. By Lemma 4.1, packets sent 1 → 2 must not loop back, and so must travel 2 → 10 → 4 → 1. As a result the packet travels 5 → 1 → 2 → 10 → 4 → 1 → 5 → 1 which is a loop since the edge 5 → 1 is repeated. □

References

[1] S. Cho, T. Elhourani, and S. Ramasubramanian. Resilient multipath routing with independent directed acyclic graphs. In *ICC*, 2010.

[2] A. Cvetkovski and M. Crovella. Hyperbolic embedding and routing for dynamic graphs. In *INFOCOM*, 2009.

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

[4] A. Kvalbein, A. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP network recovery using multiple routing configurations. In *INFOCOM*, 2007.

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

[6] S. Lee, Y. Yu, S. Nelakuditi, Z. Zhang, and C. Chuah. Proactive vs Reactive Approaches to Failure Resilient Routing. In *INFOCOM*, 2004.

[7] J. Liu, B. Yan, S. Shenker, and M. Schapira. Data-driven network connectivity. In *HotNets*, 2011.

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

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

[10] K. Wada and K. Kawaguchi. Efficient fault-tolerant fixed routings on (k+1)-connected digraphs. *Discrete Applied Mathematics*, 1992.