Title
Routing Along DAGs

Author
Liu, Junda

Publication Date
2011

Peer reviewed|Thesis/dissertation

Routing Along DAGs

by

Junda Liu

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Scott J. Shenker, Chair
Professor Ion Stoica
Professor John Chuang

Fall 2011

Routing Along DAGs

Abstract

Routing Along DAGs

by

Junda Liu

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Scott J. Shenker, Chair

Since the invention of packet switching networks, routing is an important, if not the most fundamental, component of networking. Over decades, both academia and industry put huge efforts to improve routing in various scenarios. However, with the explosive growth of Internet, wide adoption for critical business and services, and new environments like data center networks, routing has more difficulties to meet the increasingly stringent requirements.

We examine current routing schemes and discover that the problem is more fundamental, because they rely on many assumptions and trade-offs that are no longer applicable for today's networks. Believing that more radical changes are necessary to effectively address the challenge, we start the design process from scratch, with assumptions, trade-offs and design philosophies which better reflect the networks of today and tomorrow. The outcome of the design process is a unified routing framework which utilizes directed acyclic graph (DAG) as routing topology. And we name it Routing Along DAGs (RAD).

RAD separates route optimization and connectivity maintenance, handles both failures and congestions, and recovers fast and locally. We explain the details of RAD design and test its performance on various real world topologies. Then we improve RAD to achieve even better performance and cover more use cases. With these extensions, RAD becomes both complete and practical.

To my family

Contents

List of Tables

List of Figures

Acknowledgments

I am grateful to my professors, family and friends for their help and support during my entire PhD career. My time at Berkeley with the RAD Lab and NetSys Lab was truly an amazing and invaluable experience. Thanks to Professor Scott Shenker, my advisor, for guiding me through every aspect of my research. From identifying challenging problems in practice, to understanding the fundamental limitations, from brainstorming creative ideas, to proposing innovative approaches with strong reasoning support, Professor Shenker gave me tremendous guidance and help that I will treasure in my entire life.

My special thanks go to other members of my dissertation committee, Professor Ion Stoica and John Chuang. Their suggestions and feedbacks help improving my research and dissertation significantly.

Thank everyone in the RAD Lab and NetSys Lab. I couldn't have asked for better friends and collaborators. I enjoyed the discussions and stimulating environment, and learned a lot from them.

I am also fortunate to collaborate with talented researchers from other universities and research institutions. Thanks to Professor Jennifer Rexford from Princeton University, Michael Schapira from Hebrew University of Jerusalem, Brighten Godfrey from UIUC. They generously shared their experience and knowledge, and I appreciate all their help.

I also received great feedbacks from people with industrial relations. Thank Martin Casado and Teemu Koponen from Nicira Networks, and Amin Vahdat from Google Inc. They encouraged me to consider more practical requirements and constraints, and helped shaping the goals of my research.

Finally I would like to thank my parents and wife, for all their support and help in my life. And thank my daughter, for all the happiness she brought to the family.

Chapter 1

Introduction

1.1 Motivation

Routing is arguably the most fundamental aspect of networking, since it answers the basic question: *how do you place the appropriate state in routers or switches so that packets will travel from source to destination?* In the early days, it may be good enough to simply deliver the packet. But as networks become popular across the world and carry more and more communications, how to forward packets efficiently becomes another, maybe more important, task for routing. So routing needs to provide a result that is both correct and optimized. Seeing its direct impact to the correctness and performance of networking, both academia and industry put tremendous efforts to improve routing. There is a huge academic literature on routing, and the commercial world has been honing their routing implementations for many years. We have seen new routing algorithms, modified schemes for new networks, improved mechanisms for new requirements, etc. After all this academic and commercial routing work, one might expect that there would be little new to say about routing.[1]

1.1.1 New Challenges

However, there are two trends that are driving a new round of routing research. First, networks are increasingly used for critical services, so network reliability requirements are now more stringent (*i.e.,*"five nines" of reliability is often quoted as an aspirational goal, but even significantly less ambitious targets are beyond the reach of today's routing protocols). Routing, broadly defined, provides reliability in three ways: failure resilience (the ability to tolerate link failures without having to recompute routes), failure recovery (timely recomputation of routes when required), and load distribution (distributing load across the network

[1]In this thesis, we focus on intradomain routing. Interdomain routing, with its requirement for policy autonomy and flexibility, has significantly different routing requirements. We will briefly discuss the potential of our approach in interdomain routing and point to some pioneer work in Improve RAD chapter.

so that no congestion hotspots degrade performance). All three of these mechanisms must be improved if networks are to meet these demanding performance requirements.

Second, networks are growing at a rapid pace, and a new class of networks — datacenters, which can have hundreds of thousands of hosts and millions of VMs — are pushing the scaling limits as never before. Routing algorithms are essentially distributed consistency algorithms, so their convergence times (for responding to failures and hotspots) and/or the routing overhead (in terms of the number and size of routing messages) necessarily increase with size. Beyond being big, datacenters also combine high path diversity with volatile traffic demands, making multipath routing crucial in achieving full bisection bandwidth.

These two trends demand that routing protocols do a better job (*i.e.,*be more reliable, with fewer and shorter outages) of a harder task (*i.e.,*routing on larger and more complex networks). This challenge is not limited to a particular networking scenario or layer: it applies to wide-area networks, enterprise networks, and datacenter networks alike, and at both layer 2 and layer 3.

1.2 Previous Efforts

As a result, both the commercial world and the academic community have been actively investigating new routing paradigms. Various rerouting methods, such as Fast Reroute in Multi-Protocol Label Switching (MPLS), have been available in commercial products for years; these provide fast failover, improving the reliability of a given path, but are generally not used for load distribution.[2] In the last decade the academic community has produced myriad multipath routing methods (see [50, 59, 49, 44, 41, 14] for a sampling). These multipath mechanisms make it easier to spread load, and they can be used by the endpoints for failure recovery. However, because these new paths are requested by the ends, not by internal nodes detecting failures, these multipath methods do not provide fast in-network failover.

These proposed mechanisms significantly improve the reliability of networking, but are still far from meeting the required reliability goals. This is largely because these recent efforts are retrofitted on top of the traditional approach to routing, which builds a single path from the source to the destination, and do not incorporate more effective failure resilience, failure recovery, and load distribution into the routing's algorithmic foundations.

More recently, in a series of pioneering publications (see [60, 45, 46, 43, 61, 62]), the research community has begun to explore an entirely new routing paradigm, one that changes the basic output of routing for a given source-destination pair from a single path to a *directed acyclic graph* (DAG). DAGs provide multiple paths between the given source and destination that can be used for both fast failover (thereby improving failure resilience) and load distribution. DAG-based approaches are hardly new; they were first investigated in the

[2] An exception, for the datacenter, is ECMP, which does both fast failover and load distribution, but has limited flexibility.

wireless literature in 1983, when a prescient paper by Gafni and Bertsekas [21] proposed an algorithm to restore connectivity using local recovery methods. This was followed by several other papers in the wireless literature, though this family of algorithms seems to be out-of-favor currently. Even in the wired literature, while there was little explicit attention to DAGs, they were implicitly used in an earlier generation of routing protocols [7, 24, 23, 68]. Despite this earlier interest, it was only recently that the community focused more generally and intensely on DAG-based schemes for wired intradomain routing.

1.3 Unified Framework

We believe the DAG approach has more potential to explore. One obvious advantage it has is to unify the failure and congestion handling into one framework. We think this is of value because, today, routing designs typically achieve failure resilience and load distribution with two separate and reasonably complicated mechanisms (such as backup paths for resilience and computing link weights for load distribution), and each are separately overlaid on top of a basic routing protocol that provides failure recovery. Our RAD approach addresses all of these issues in a single mechanisms; given how hard it is to manage modern networks, simplifying routing would be a major improvement. Our performance analysis suggests that we can achieve this unification while preserving, and in many cases improving, routing performance.

In achieving this unification, we argue for a different separation of concerns in route computation that is not possible with shortest-path approaches. We focus on loop-prevention as the highest priority, followed by connectivity, followed by path optimization. Some canonical routing algorithms like distance-vector invert this order, guaranteeing loop-freeness only when the shortest paths have been computed.

1.3.1 Handling Dynamics

Routing is easy when networks are static and lightly loaded; one can compute the routing tables once (without tight time constraints on the computation) and then leave this routing state unchanged. However, when links fail or become overloaded, a more dynamic response is required. In fact, the performance of a routing protocol can largely be described in terms of how quickly and how well it responds to failed or overloaded links.

The quality of the response to failures can be measured by the degree to which connectivity is (or is not) guaranteed. That is, *if the failure does not disrupt the connectivity of the resulting graph, does the routing algorithm ensure that packets will be delivered?*

The quality of the response to overloaded links is measured by how effectively the load is distributed: *how large a load can the network carry without a link becoming overloaded?* Note that we are not equating load distribution with load balancing: the goal is to carry

more traffic without overload, not to minimize the maximal load on the links.[3] While the latter is mathematically cleaner, the former more directly measures the performance from a provider's point of view.

Thus, our goal is to develop routing algorithms that guarantee connectivity in the presence of failures and maximize the amount of traffic the network can carry without overload. We also, as noted above, want routing to respond to failures and overloads quickly. When defining "quickly", rather than discussing absolute numbers we compare against two different timescales: local responses (when the network event can be handled locally within the router or switch detecting the problem) and global recomputations (where the routing algorithm must be re-run in order to recompute the state in routers). We would ideally like routing algorithms to guarantee connectivity and maximize the traffic carried without overload, all while using only local responses. Thus, we want an approach that is both *unified* and *local*.

1.4 Summary

Despite decades of research and engineering efforts, routing still has difficulties to meet the scalability and availability requirements from today's networks. We argue that more radical changes are inevitable because many assumptions and trade-offs of current routing schemes are no longer applicable for the networks of today and tomorrow.

We propose RAD to address this challenge. Built from revised assumptions and design philosophies, RAD is a unified routing framework that handles both failures and congestion, reacts locally and scales well. With appealing features and modern design, RAD can be applied to many different network scenarios and has more potential to meet even more restrict requirements.

The rest of the dissertation proceeds as follows: Chapter 2 details the design process of RAD and gives an overview through one example. Chapter 3 reviews related routing efforts and compares them by different metrics. Chapter 4 explains the details of RAD and shows the performance evaluation. Chapter 5 discusses improvements and extensions to RAD, and Chapter 6 concludes with future direction suggestions.

[3]Overload means when the link can no longer meet a reasonable SLA; depending on the historical burstiness of the load, and the desired SLA, the target utilization could vary widely.

Chapter 2

RAD Overview

At first glance, choosing DAG as the routing topology is simply a change of graph concept. However, the departure is more fundamental. To better explain the differences, we first detail the design process of RAD in Section 2.1. We start the process by reviewing routing in packet-switched networks and its three goals: connectivity, optimization and resilience. For each goal, we analyze its underlying requirements and examine how it is reached by conventional approaches. Then we explore the design space by carefully reexamining the assumptions and trade-offs, and making them more applicable for the networks of today and future. Several high level ideas include:

- Separate optimization and connectivity
- Build explicit loop-free topology
- Allow temporary suboptimal forwarding
- Prefer fast and local recovery

Naturally, the process leads to RAD, a complete and coherent routing framework. We highlight the advantages of RAD, and provide a simple example in Section 2.5 to illustrate major components of RAD and describe how they work together to achieve the three goals of routing.

2.1 Review Routing

In packet-switched networks, every packet carries a hint (bits in packet) for its own delivery. Common choices of delivery hint include destination MAC address in layer 2 switching networks, destination IP address in IP networks and pre-defined label in virtual circuit networks like MPLS. Extra information may also be used to enable finer granularity control of forwarding and extensions to routing such as source routes, so long as it is supported

by the network. By checking the delivery hint in the packet, each router, or Forwarding Equipment (FE) in general, decides how to do the forwarding on a per-packet basis. These decisions come from routing. Or more precisely, the role of routing is to provide choices and make packet forwarding decisions. The choices may be a set of next hop routers and the decision is simply the chosen one.

2.1.1 Choice and Decision

However, the distinction between choice and decision is often neglected, because conventional routing provides only one choice[1] and the decision is always the same as the choice. Yet the distinction is important. By separating the two concepts, one can view routing as two parts: a method to compute choices and a set of rules to pick one, which will enable greater flexibility and more potential of routing. But there is a challenge. Researchers have been familiar with the term "topology" and there are no well acknowledged terms to differentiate choice and decision.

2.1.2 Three Topology Abstracts

To be compatible with conventional terms and avoid future confusion, we now introduce three key topology concepts about routing.

- Network Topology (NT): underlying physical network, often represented by an undirected graph with nodes and links.
- Routing Topology (RT): forwarding choices, the output of routing algorithm and usually independent per delivery hint.
- Forwarding Topology (FT): forwarding decisions following the selection rules, a subset of Routing Topology (may also be the same), also delivery hint specific.

The two parts of routing can also be described as how to get RT from NT and how to decide FT from RT. Take Spanning Tree Protocol as an example. Its Routing Topology is one single tree, and Forwarding Topology is the same tree. For most IP routing protocols, the RT is one tree per destination IP address, and FT is also the same as RT. In contrast, one can imagine a more resilient routing design would create a rich connected RT, not a single tree, but keep FT as a tree to avoid loops.

[1] Any multipath methods including ECMP are separately considered and will be discussed in more details later

2.1.3 Routing's Goals

Most of the time both the users and network operators only care about FT, which directly impacts the performance of packet delivery. But since FT is a subset of RT, meaning the decisions are limited by choices, a good RT is crucial to begin with. Therefore routing should be evaluated by both the choices and decisions it will provide, with specific criteria listed as follows.

- Correctness: packets should be successfully delivered following the decision on every forwarding equipment.
- Performance: when packets traverse the network, the time and resources taken should be small.
- Availability: when choices become invalid due to failures, routing should provide alternate choices so packets can still be forwarded.

Corresponding to the three criteria, routing has three goals: connectivity, optimization and resilience, respectively. Understanding these goals is fundamental to the design of RAD. It will also become clear that the design choices made by RAD are a natural outcome of our insights about how to achieve these goals. In the following sections, we will discuss each goal in details and highlight the differences and contributions of RAD.

2.2 Connectivity

The goal of connectivity has two folds. First, if node A and B are connected in NT, they should also be connected in RT. Second, packets sent from A can reach B, and vice versa[2]. However, it is not easy to directly measure connectivity by above definition, because it is often expensive to test for every source-destination pair in the network. To solve this problem, we first present one key observation of connectivity: "loop-free and keep forwarding guarantee connectivity".

- Loop-free: No loops in FT. So any packet that follows the forwarding decision on each router, hop by hop, will not follows a decision on the same router more than once. It is possible that the packet visits a router more than once, but as long as the decision is not the same as before, it is not considered a forwarding loop.
- Keep forwarding: Every router in RT has at least one choice to forward the packet, so the packet will be sent to next hop router or delivered to destination, and will not be dropped on router because of no available routes.

The observation is more precisely stated as the following theorem.

[2]Unless it is physically impossible due to uni-direction links or policy constraints

Theorem 1 *For any network with finite size, packets will be delivered when both loop-free and keep forwarding requirements are met.*

The theorem shows that loop-free and keep forwarding are sufficient conditions for connectivity, and it is easy to prove that by contradiction.[3] In other words, if one routing scheme has both conditions met, its correctness is guaranteed, even though the performance may be suboptimal.

2.2.1 Separate Optimization

Separating optimization from connectivity is one significant yet logical change we make in RAD. Conventional path-based routing does not distinguish the two goals, and actually achieves connectivity through optimization. More specifically, the loop-free requirement is implicitly enforced by requiring every router only uses shortest or minimum-cost path, which is essentially the result of optimization. In other words, path-based routing can only achieve the connectivity goal and guarantee correctness *after* optimization process is finished, i.e. the shortest path is available to routers. If any router violates this and sends packets along a non-shortest path, it may receive the packets again and create a forwarding loop.

Because both RT and FT are limited to shortest paths, there are several obvious shortcomings and limitations:

- Only some links are included, which limits routing choices and performance potential.
- All routers should agree on the shortest paths, which requires global computation and synchronization.
- If a metric can not be represented on a per-link basis, such as the resilience of RT, it can not be included for optimization.
- When link goes up and down, all routers need to be aware of that and re-compute RT to avoid potential loops.

As we have shown earlier, optimal path is not necessary for connectivity. It just happens to be the case for conventional routing because it combines two tasks into one. There are many other means to achieve the loop-free goal. For example, MPLS and various multi-path routing ideas solve this problem by introducing per-path labels or tags. Typically, the packet will carry extra bits which can provide more information to make the forwarding decision. Routers may also update those bits to reflect topology changes, so other routers can learn that when they receive the packets. For more details, we compare different mechanisms and discuss advantages and problems in the related work section.

[3] In almost all practical cases, loop-free and keep forwarding are also necessary conditions for connectivity. But one might argue if FT has transient loops, some packets may be successfully delivered. Detailed discussion of such scenarios is not of our current interests, and is left open for future work.

2.2.2 Loop-free Topology

We decide in RAD that to achieve the connectivity goal, we will make loop-free explicit and keep it across topology updates. In other words, loop-free becomes the first and top priority task throughout the design. Note that we only require loop-free in FT, not RT, which leaves more design space for sophisticated routing schemes. So a routing may allow loops in RT, but carefully choses which links to be included in FT to make sure FT is loop-free. One example would be that RT contains multiple pre-computed back-up links, and the selection rule is to only use back-up link when primary one fails. More details about this scenario and our decision of RAD are discussed in the resilience and improve RAD section. To simplify the discussion, by default we discuss the design which also has a loop-free RT, unless otherwise specified.

But for undirected graphs, a tree is already the best loop-free topology one can get. How can RAD improve that? The answer is to use links with directions, so the topology is essentially a Directed Acyclic Graph (DAG). The DAG is also delivery hint specific, and in one DAG, each link is used in one direction.[4] Using links in one direction per DAG does not reduce the potential throughput, because the other direction can be used to carry traffic in other DAGs. In fact, traditional IP routing has per-destination tree, and for packets to the same destination, any link in the tree is only used in one direction (towards the root). It would be counter-intuitive and cause loops if both directions are used to send same packets. The situation is similar for RAD, except that RAD can use more links while keeping loop-free.

Note the correctness does not rely on any specific DAG structure, instead, there are many different ways to generate the DAG, and the DAGs can be very different. In Section 2.5 RAD by Example, we will show a simple algorithm to create a DAG per destination that utilizes all links of the network topology, and every router will have at least one choice for forwarding (most routers have multiple choices).

2.3 Optimization

With connectivity, packets will eventually be delivered, but in practice, there are more factors to consider. For example, users would want low and consistent in-network latency among packets, in-order packet delivery and more available bandwidth, so their traffic can achieve high and stable throughput. On the other hand, network operators would prefer the packets take less network resources and leave their networks as soon as possible, avoid local congestion or hot-spot, make sure Service Level Agreement (SLA) is not violated and forwarding is compliant with policy, and many more economical concerns. Quite often both network users and operators have similar and compatible preferences, so there is little to none conflict. But there are also cases where network operators provide tiered services, and

[4]We will discuss advanced techniques about using links in both directions while keeping loop-free in the Improve RAD section.

push lower priority traffic through longer path to guarantee the performance of high priority traffic.

Given the nature of optimization, i.e. improving the performance of packet forwarding, we believe that the following properties are desired:

- Continuous process: Optimization is not a one-time result, but an evolving process with new inputs and outputs.
- Non-disruptive update: Packet forwarding should not be interrupted when adopting better choices and decisions.
- Agile reaction: An optimum may become a bad choice when conditions change, so a fast reaction will reduce the cost of suboptimal.
- Scalable scope: When a local optimization is good enough, there is no need to trigger a global recomputation.
- Flexible metrics: Support customized metrics and cost functions.

2.3.1 Common Practices

As we discussed earlier, currently the path optimization is done by incorporating many metrics into a quantitative value, the link weight, and the optimizing process is simply to find the shortest or min-cost path. This method works because the metrics can include many factors such as bandwidth, delay and load. The optimization of traffic distribution for QoS purposes is done in a similar way. A cost function is associated with the utilization of every link, and the goal is to minimize the total cost.

However, because of its tight integration with connectivity, conventional optimization has difficulties to provide desired properties mentioned above. Updating new optimal choices typically requires global action and forwarding is impacted until all routers have the same agreement of topology. Thus network operators have to make a trade-off between running network at a suboptimal state and the potential cost of adopting new optimization results, leading to lower computing frequency and slower event reaction. There are other limitations as well. For example, the overall resilience of routing topology does not fit into the optimization model, because it is inherently a global metric, and can not be broken down to per-link numbers.

And due to the unavoidable delay of information propagation, the calculated optimal may already become suboptimal even before it is disseminated to routers, because the traffic and other factors of the network already changed. Therefore a common assumption of the approach is the changes are not significant within certain time frame. For ISP networks, it works without unexpected traffic bursts. But for data center networks, where traffic varies dramatically and quickly, this approach simply can not provide an efficient result fast enough.

2.3.2 Optimization in RAD

In RAD, optimization is separated from connectivity computation and runs in parallel without interference. There are two components included, corresponding to our choice and decision distinction. First is a global DAG optimizing process which runs periodically to generate a better DAG, e.g. more link capacity is available and shortest paths are included in the DAG. This process usually creates a new optimal result when network topology is changed by operators, so the time scale is relatively slow. When new choices need to be deployed on routers, packet forwarding is not affected as long as the update order follows the router ordering in the new DAG. Details will be explained in Section 4.5.

Another optimization component is how to make the best decision from choices, which happens locally on every router and on a much faster time scale. One intuitive optimization is simply only using the shortest path, so a tree will be formed towards the destination. But because the routing topology is a DAG, every router can freely re-distribute traffic across outgoing links without worrying about forwarding loops. This enables instant reaction to traffic bursts, and greatly reduces the chances of local congestion.

Changes of traffic are typically more volatile and unpredictable than those of topology, therefore the second optimization takes effect when traffic changes, effectively keeping the network running close to optimal. Combining both global and local optimization, RAD successfully provides the desired properties of optimization and give network operators more choices to run the network in an optimal state.

When optimal choices are no longer available due to failures or congestions, RAD also makes it clear that temporary suboptimal packet forwarding is preferred. It means that RAD will not stop forwarding and wait until a new optimal choice becomes available, instead, RAD will forward the packets along suboptimal but available choices, and try not to drop the packet. The reasoning behind this decision is simple. Since dropping a packet means an infinite forwarding latency, we assume that compare to forwarding along suboptimal choices, dropping a packet incurs more cost and penalty for both users and network operators. This assumption holds in most practical scenarios, where the extra latency is comparable.[5]

One might argue that if the latency exceeds some threshold, the communication ends may believe the packet is lost and the sender will resend the packet. The users will definitely benefit from the duplicated packets, but more network resources will be utilized. We believe that as long as the network capacity allows, network operators would avoid dropping packets to fulfill the best-effort service. Besides, RAD can run separately by traffic classes. So the high priority traffic will take all available network capacity for a guaranteed performance, while low priority takes what is remaining.

[5]We will discuss how RAD can be easily extended to further reduce the latency in Chapter 5.

2.4 Resilience

Network is dynamic, and failure happens. Even though packet-switching networks are designed to handle the dynamics and generally believed to recover well after failures, the resilience of routing has great room for improvement, because the scale and complexity of today's networks have changed dramatically. Besides, as we mentioned in the introduction section, the uptime and availability requirements are also higher than ever, and will only be more stringent as networks become a critical infrastructure.

2.4.1 Timescales

In short, routing should react to failures as quickly as possible, and the interruption to packet forwarding should be minimized. To better analyze the problem, we first discuss four different timescales of routing actions. The timescales are mostly decided by the scope of the behavior, and what resources it requires, e.g. ASIC forwarding chip or control CPU. Given the nature of these scopes and resources, the timescales vary significantly, with several orders of magnitude difference. Therefore, understanding the different timescales is the first step to improve routing's availability.

- Local hardware: if the network event can be handled locally within hardware of the router or switch detecting the problem. Modern hardware implementations like ASIC and FPGA can change or update states within microseconds or even lower.

- Local software: if the network event can be handled locally within the router or switch detecting the problem, but the response requires software processing from the router or switch. Depending on CPU capacity and computation complexity, the time needed varies with environments and equipment settings. Based on discussions with vendors, a common estimate is a couple of milliseconds. So it may already be 1000 times slower than local hardware reaction.

- Peer response: if the router or switch detecting the problem must contact some peer entity in order to evoke a response to the problem. We can think of this as a sum of two local software delays and round trip time between the two routers. The time can be on the order of 10 to 100 milliseconds. The number will be much smaller in data center networks, where end to end latency is usually less than one millisecond.

- Global recomputation: if the distributed state must be recomputed when a problem is detected. Global recomputations take longer as the network grows in size, but in terms of order of magnitude we can think of this as being on the order of 10s of seconds (or more).

We also need to add to these timescales the time it takes to detect a problem, which can vary from low-level signal detection to high-level congestion detection. In some cases,

the time to detect a problem will be longer than all but the global recomputation time. In others, it will be on the same order as a local hardware response. But we argue that no matter which failure detection case applies, routing itself should never become a bottleneck, and advances in technologies, like fiber optical, have potential to provide close to real time failure detection. Therefore we focus on the reaction times of routing itself, assuming failure has been detected in a timely manner.

The four timescales listed above provide a better design framework for routing resilience. Following the clear guidance, a resilient routing design should prioritize different actions, and avoid time consuming global recomputation as much as possible.

Unfortunately, most conventional routing schemes do not have the notion of dramatically different timescales, and involve global recomputation frequently when reacting to network dynamics. Such decision works fine in the early days, when the size of network is relatively small, and the time difference between hardware and software processing is not that significant. But as networks grow in size and complexity, and new generations of hardware are deployed, routing should take into account the changes, and evolve with them with better reacting mechanisms.

2.4.2 Global Recomputation

As discussed in previous section, global recomputation is very time consuming. Since it significantly impacts routing resilience, one would want to reduce the time it takes. However, due to the inevitable latency of state propagation, aggressively reducing timers usually leads to instable routing results, and the computation may have oscillation and never reach a final consistent result.

Therefore to improve routing availability, the frequency of global recomputation, not the duration of it, should be reduced first. In other words, the routing design should avoid invoking this mechanism unless it becomes necessary, i.e. no other reaction method can handle the situation with acceptable performance. But shortest-path routing generally does not support this, because if current shortest path is no longer available, the algorithm will find a new shortest path, which is essentially an optimal result that requires global synchronization. Same analysis applies to any design that always requires optimal states to operate on.

A common trade-off in various resilient routing proposals is to quickly adopt a suboptimal (non-shortest) path and restore connectivity as soon as possible. This can improve the availability because intuitively, finding an optimal choice is more time consuming and resource expensive than simply finding a usable one. This trade-off is also widely accepted by industry and appears in many proposed standards like IP Fast Reroute, MPLS backup path, etc. As discussed in the connectivity section, because they are still path-based approach, using non-shortest path requires explicit labeling or signaling to make sure all downstream routers are aware of the change, and avoid forwarding loops. Schemes that use packet labeling generally store extra bits into packets, and rewrite or remove them when necessary.

Although they may provide good resilient results, the change to data plane implies high cost for practical deployment.

Although forwarding a packet along a longer path introduces extra latency, it should be a better choice than dropping the packet, which can be considered as infinite forwarding delay. But if the extra delay is significant enough that the packet has been resent by the sender, the benefit of not dropping it becomes negligible. Therefore to make sure the sub-optimal choices are still good enough for forwarding, path stretch, the ratio between new path and previous shortest path, is an important criteria to evaluate multi-path proposals.

However, an often neglected metric may have more impact in the overall routing availability. Consider a multi-path algorithm that computes two forwarding choices on one router. Obviously it can withstand any one of it fails. But if both of them fail, the algorithm has to rerun the computation to generate another two choices. Due to the added complexity in the algorithm, the computation may take five times more time than a normal shortest path computation. So this particular multi-path proposal has two fewer recomputations than shortest path approach, at the cost of much longer down time during recomputation. In practice, its overall routing availability may be even worse. Therefore, simply pre-compute multiple choices and reduce the frequency of recomputation only solves part of the problem, the added complexity and potentially longer computation time should also be considered in the design.

2.4.3 Local Repair

In RAD, we explore another option to improve resilience: local repair. The first step is the same as multi-path ideas: computing multiple choices for forwarding. But when all precomputed choices fail, instead of recomputing new choices, RAD will try to find available choices locally, i.e. without global message exchanges and computation.

But is this possible?

When we review the connectivity requirement, we learned that loop-free and keep-forwarding will guarantee packet delivery. Although loop can not be detected locally, if we start from a loop-free topology, like a DAG, we can devise a mechanism to modify the topology while keeping its acyclic property. And keep-forwarding basically requires every router has a choice to forward, which is locally available to the router. When a router no longer has any choice, the routing topology can be modified in a loop-free way to provide new choices.

We call the scheme local repair, in that it fixes "stopped-working" topologies without the requirement for global agreement. Following our timescales discussion, local repair shows great potential in improving routing availability, because it avoids global recomputation and can even be implemented in hardware. The comparison with shortest path routing and multi-path proposals is listed below:

- Shortest path routing:

  - Initial choices: only one, the shortest path.[6]
  - Failure reaction: always recompute a new shortest path.

- Multi-path proposals:
  - Initial choices: multiple, may include non-shortest paths.
  - Failure reaction: recompute when no choice is available.

- RAD:
  - Initial choices: multiple, may include non-shortest paths.
  - Failure reaction: Pick next choice if there is any available. Local repair when no choice is available.

The philosophy of local repair is surprisingly simple: try other neighbor nodes if all precomputed choices are not available. Thinking in a DAG way, if one node loses all outgoing links, it will try to use its incoming links for forwarding. So the direction of the link is reversed. Observe that if recomputation is triggered, the new computed choice must be one of those previously incoming links, the local repair mechanism follows the intuition that one of the remaining neighbor should be able to keep forwarding the packets.

The challenge is how to make sure the process keeps the loop-free property of routing topology. We will discuss more details and provide proofs in Chapter 4.

2.5 RAD by An Example

In this section we describe how RAD works with a simple example. The description here is not meant to be comprehensive and detailed, but instead captures the basic mechanisms of RAD and demonstrate its effectiveness. We first introduce the example topology, then explain the components of RAD and how they work together to handle failures and congestions. We focus on basic mechanism and leave many design details and proofs for Chapter 4.

2.5.1 Simple Topology

We consider a network topology as shown in Figure 2.1a. There are total five nodes including the destination and seven links. d is the forwarding destination, and nodes (routers) 1, 2 and 3 are directly connected to d, while node 4 is two hops away and directly connects to nodes 2 and 3. All links between nodes are undirected, meaning they can carry traffic in either direction. The link weight can be arbitrary, but for simplicity, we assume it is unit weight. Therefore the distance can be measured by hop counts.

[6]ECMP is considered as a multi-path proposal.

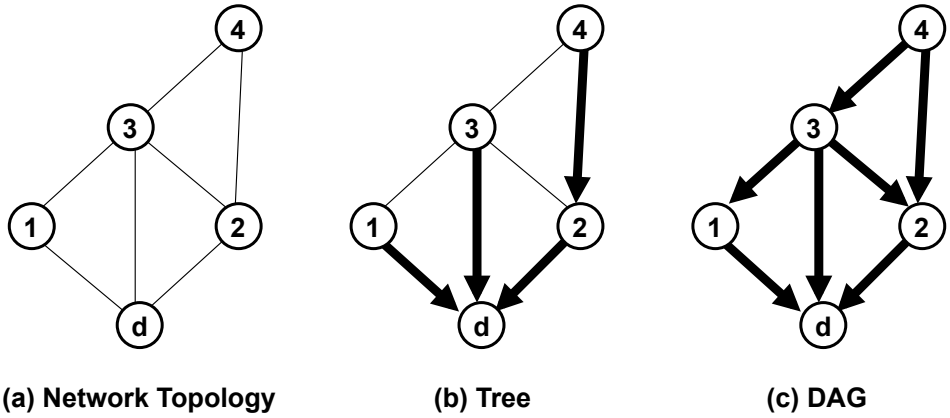

Figure 2.1: A tree and a DAG on a network graph. In the tree, the only path from 4 to d is 4-2-1. In the DAG, there are four paths: 4-2-d, 4-3-d, 4-3-1-d, and 4-3-2-d.

Figure 2.1b shows how one shortest path tree rooted at d can be constructed. [7] Because only links along shortest path can be included in the topology, the number of utilized links is always the number of nodes minus one, so it is four in this case. Only 57% percent of links are used, and every non-destination node only has one forwarding choice.

But if we compute a DAG, as shown in Figure 2.1c, the situation is different. First, all links, including those along non-shortest path, can be included in the routing topology, which means 100% coverage of network links. Given the added links in routing topology, many nodes have more choices to forward packet. For example, from node 4 to d, there is only path 4-2-1 in the tree, while in the DAG, there are four of them: 4-2-d, 4-3-d, 4-3-1-d and 4-3-2-d.

2.5.2 Computing DAG

The first task of RAD is to compute the DAG as its routing topology. We will present formalized problem statement, general algorithm and proofs in Chapter 4. But here we show a simplified algorithm that produces good enough result for our topology.

The algorithm start from destination node d, and iterates all its direct neighbor nodes, 1, 2 and 3 in our example. Apparently links between d and its neighbors should have the direction towards d. Then it iterates over neighbors of d and assigns direction to links by comparing the id of two ends, pointing to the smaller node. For example, the link between 3 and 1 points to 1. Then nodes that are two hops away, node 4, are considered. Since 4 has

[7] In this example, node 4 has two shortest paths 4-2-d and 4-3-d, and without ECMP it just picks one of them: 4-2-d.

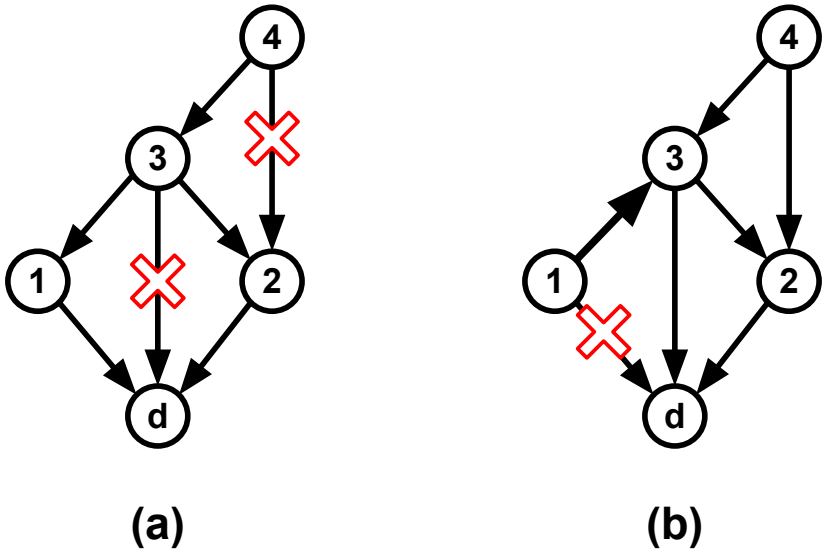

Figure 2.2: RAD failure handling mechanism. (a) When link $3 \rightarrow d$ fails, node 3 can instantly start using $3 \rightarrow 1$ or $3 \rightarrow 2$. (b) When link $1 \rightarrow d$ fails, node 1 will reverse link $3 \rightarrow 1$ to $1 \rightarrow 3$.

two neighbors 2 and 3, and both of them are directly connected to d, the link goes from 4 to 2 or 3. After this computation, we get the DAG as shown in Figure 2.1c.

2.5.3 Handling Failures

When link fails, the first resilient mechanism of RAD is to switch to another choice. For example, when link $3 \rightarrow d$ fails, node 3 can instantly start using link $3 \rightarrow 1$ and path 3-1-d for packet forwarding, as shown in Figure 2.2(a). Similarly, node 4 can tolerate either link $4 \rightarrow 2$ or $4 \rightarrow 3$ failure, and pick another one immediately. The advantage of RAD is that such decision changes can be completely local, without having to notify other nodes the path has been changed. The reason is that the routing topology is loop-free (a DAG), therefore choosing any subset of the topology will remain acyclic. And as long as there is no forwarding loop and every node has a choice, the packet will be delivered. Recall that local hardware reaction can be one million times faster than global recomputation, there is no doubt that local repair can greatly improve routing resiliency.

But what if a failure that removes all choices? For example, in our example topology, node 1 only has one choice $1 \rightarrow d$, because it is directly connected to d and has the smallest id number. If link $1 \rightarrow d$ fails, most multi-path proposals will require a new round of global computation, but RAD has a different mechanism, local repair. Instead of recomputing a new DAG from the changed network topology, RAD will reverse the link $3 \rightarrow 1$ to $1 \rightarrow 3$,

then node 1 has a usable choice, although may be not optimal, to forward packets, as shown in Figure 2.2(b). The basic rule is when a node has no outgoing links, it will reverse all incoming links to outgoing. In our simple example, the connectivity for node 1 is quickly restored by reversing link $3 \rightarrow 1$ to $1 \rightarrow 3$. And node 3 still has link $3 \rightarrow d$ and $3 \rightarrow 2$ to forward packets. Therefore the connectivity of routing is repaired by local action among node 1 and 3, with no global message exchanges or route recomputation.

2.5.4 Handling Congestions

Traffic in the network may change dramatically. RAD adopts a responsive load distributing method to deal with unexpected traffic bursts. For instance, if link $3 \rightarrow d$ becomes congested, node 3 can offload excessive traffic to link $3 \rightarrow 2$ and $3 \rightarrow 1$. Unlike conventional traffic engineering approaches, RAD does not try to minimize a cost function on every router. Instead, the load distributing rule is simply:

- Keep sending along a link if its utilization is below certain threshold
- Offload to next available link which has capacity
- Notify upstream nodes if all links are congested

We will show later that such simple and local load distributing method can provide close to optimal traffic engineering results.

2.6 Summary

Current routing protocols typically use a *path-oriented* approach. That is, for each destination they compute a tree of paths spanning all the nodes and converging on the destination, providing a path between each source and destination that can be realized with destination-based routing tables. In the case of L2 technologies, these trees are often a single spanning tree. It is the path-based nature of current routing that creates the need for global recomputation, because every time a link fails or becomes overloaded the paths traversing that link must be recomputed. Most of the techniques described above that improve reliability do so by giving routers/switches[8] (or edge devices) access to more than one path, but they do so in a limited manner (in that they typically compute a small number of alternatives, often using *ad hoc* techniques). Here, we propose a routing paradigm that makes multiple paths an inherent part of the paradigm, which allows us to reason about it formally and implement it cleanly.

[8]Our discussion applies to both L2 and L3, but for convenience in what follows we will refer only to routers, not switches.

Simply put, we propose that the basic computational output of routing algorithms not be a tree of paths but instead be a spanning directed acyclic graph (DAG). That is, for each destination the routing algorithm computes a DAG that spans all nodes and provides each source at least one path (and typically more) towards the destination. This approach of Routing Along DAGs (RAD) intrinsically provides a *set* of paths, rather than a single path, from each source to each destination. And switching between paths is a local decision.

At the end of the RAD routing computation, each router has one or more outgoing ports for each destination. These routers can unilaterally choose which outgoing port to use, based on local failure or load information, since there is no chance that these independent choices will result in loops. Because these choices can be made locally by each router, RAD does not require signaling bits in the packet header, or participation from the network edge, to select paths. ECMP has this property as well, but is restricted to choosing among equal cost paths. For example, unlike the DAG of Fig. 2.1(c), ECMP would not use links $3 \rightarrow 1$ and $3 \rightarrow 2$. This aspect of RAD can therefore be seen as a generalization of ECMP that, as we shall see, has substantially better resilience and load distribution than ECMP in most networks. In addition, if the DAG becomes disconnected due to failures, RAD can modify the DAG structure using *local* link-reversing responses (which we describe more details later) to restore connectivity (if possible; and detect disconnection if not); this feature of RAD is not foreshadowed by ECMP. All of this is provided by an algorithm that is closer in complexity and communication overhead to distance-vector than link-state or path-vector.

More conceptually, RAD separates the task of responding to failures and congestion, which RAD does locally, from the task of responding to managed topology changes and route optimization (i.e., the addition or deletion of links), which requires a DAG recomputation to fully accommodate. This is appropriate, since these two categories of events, failures and topology changes, occur on very different time scales. Thus, RAD does what needs to be done rapidly (failure recovery and load balancing) on a purely local basis, and does what needs to be done globally (computing the DAG) on much slower time scales. The slow rate of global recomputations of the DAG, enabled by the use of local recovery, gives RAD much lower control overhead than standard approaches.

Aside from intellectual contributions, RAD also has several features that are appealing in practical deployment:

- No proprietary packet labeling or signaling. Therefore its implementation does not require expensive changes to data plane. RAD also has greater deployment flexibility because it does not depend on specific layer or packet header format.

- Handle bursty traffic locally and effectively. Currently networks usually run with low link utilization, to reserve enough capacity absorbing bursty traffic. But with RAD, network links can have a higher utilization, with much efficient use of resources, and still be able to avoid link congestion.

- Modular components. Just like modern operating systems, the design of RAD is mod-

ular. It enables great flexibility and can adapt to many different usage scenarios with ease.

Chapter 3

Related Work

Since our efforts improve routing in many fundamental aspects, we do not intend to give a comprehensive overview of all the routing work that is related to what we describe here. Instead, we briefly explain traditional routing methods, then review three threads of work: current practice, recent non-DAG research, and DAG-based approaches. These three categories cover routing improvements over time, research efforts that share similar motivations with us, and proposals that also leverage DAG to solve problems.

3.1 Traditional routing:

How does the traditional routing paradigm deal with failed and overloaded links? Every time a link fails, all paths traversing this link must be recomputed (via a global route recomputation), leading to a temporary service outage and a burst of control traffic. Moreover, in cases such as spanning tree protocols the recomputation breaks paths that were previously working.

Early attempts to distribute load via load-dependent routing led to unfortunate oscillations [39], so traditional routing algorithms are not load-dependent. Instead, load distribution is traditionally done by adjusting link weights and then doing a global route recomputation (often the link weights are adjusted with the goal of minimizing the maximal link utilization) [19].

Thus, for both failures and overloaded links, traditional routing methods lead to global route recomputations. Global recomputation scales (either in terms of overhead or latency) with the size of the system (in some cases nonlinearly). Thus, if we are to help routing meet the dual demands of larger networks and shorter outages, we must find a way for routing to react to failures and congestion *without global route recomputation.*[1]

[1]Designs like the Rapid Spanning Tree Protocol (RSTP) [2] provide much faster convergence than standard Spanning Tree Protocols, but the resulting outages are still unacceptable for large networks with stringent performance requirements.

3.2 Current Practices

Both the commercial world and the academic community have been actively investigating new routing paradigms. There have been many routing enhancements that deal with failure-recovery and load-distribution, and have been applied to production networks. Various rerouting methods (such as MPLS Fast Reroute) have been available in commercial products for years; these provide fast failover and improve the reliability of a given path.

- MPLS: MPLS Fast Reroute [56] avoids recomputation by providing a rapid local response to failures wherever there is an alternate route set up at the router that detects the failure (and a non-local response when the backup path is remote from the failure site). These backup paths provide resilience to single failures, but cannot guarantee connectivity with multiple failures. They are also somewhat cumbersome to manage, in that all but the simplest backup functionality (backing up a single link or node) requires manual configuration.

  MPLS Fast Reroute is currently the most widely deployed method for traffic protection in the WAN. It provides fast failover (sub-50ms failure recovery), but not load balancing or error recovery (that is, if the backup path also fails, Fast Reroute cannot by itself restore connectivity).

  MPLS also allows operators to set up multiple paths and then split the load over those paths. This gives only a limited ability to direct traffic (since it must be over one of the preconfigured paths).

- ECMP: Equal-Cost Multi-Path (ECMP) allows a switch or router to split traffic among multiple outgoing shortest paths.[2] This supports both failure recovery and load-distribution, but the alternate paths used must be shortest paths (and thus must be exactly the same length), which is limiting in irregular topologies (but fine in standard CLOS topologies). By adjusting link weights one can effectively use ECMP in more general topologies, but this requires global route recomputation.

  ECMP is a multipath routing scheme that bears some similarity to RAD, in that it provides quick failover and load distribution. However, it is limited to cases where paths have exactly the same cost, and even then can only split traffic evenly over these paths; this works well for very symmetric datacenter topologies but is impractical for WANs and other irregular topologies. Moreover, ECMP itself does not provide error recovery, it must be paired with a standard routing mechanism.

- Multiple Spanning Trees: This approach computes multiple spanning trees and gives switches a choice of which tree to use. This can be reasonably effective against single link failures, but current implementations (which use only a few trees) do not guarantee connectivity (when the graph remains connected).

[2] We consider Link Aggregation (LA) to be just a special case of ECMP.

Thus, both ECMP and Fast Reroute are similar to portions of the RAD approach, but neither represents a comprehensive algorithm like RAD. First, it requires careful planning and deployment, which involves strategically selecting routers that will act as Points of Local Repair (PLR). This is obviously dependent on the given topology, and thus there can be no universally correct way to select PLRs, which makes deployment perilous as it leaves the intricate task of optimization up to the network operator. Then, FRR's "one-to-one backup" scheme requires impractically much state on the PLRs, as they need to store a backup information for each Label Switched Path (LSP). On the other hand, the alternative "facility backup" sacrifices load distribution flexibility for state, as it reroutes all the protected traffic the same way. Moreover, FRR is only a local protection mechanism, and a separate mechanism is required to compute primary paths.

3.3 Recent non-DAG research

In the last decade the academic community has produced myriad multipath routing methods (see [50, 59, 49, 44, 41, 14] for a sampling) to improve routing. These multipath mechanisms make it easier to spread load, and they can be used by the endpoints for failure recovery.

- Multipath routing: This approach precomputes several paths and allows sources (or edge routers/switches) to change paths; in terms of precomputing paths, it is a generalization of MPLS Fast Reroute. Several multipath computation algorithms have been proposed (in contrast to the manual configuration approach often used for MPLS), but most do not guarantee connectivity under all failure scenarios that leave the graph connected [50, 73, 74].

- Packet marking: There have been some schemes (such as [47]) that, by altering the forwarding behavior of switches and putting marks in the packet, can guarantee connectivity in the face of arbitrary failures (as long as the graph remains connected). However, these require a major change to IP, and do not address load distribution.

- Better load distribution: there are some recent proposals, such as TeXCP, that greatly improve load distribution, but do not give rapid response (shorter than round-trip time) to failures [38, 34].

The work in [66] proposes a technique for joint failure recovery and traffic engineering, by splitting traffic over multiple precomputed paths between each pair of edge routers. However, the solution relies on path-level failure detection (slower than the link-level mechanisms used in RAD) and the routers must store state for each path (rather than the more compact DAG representation in RAD). SPAIN [51] supports multipath routing in data centers through multiple precomputed spanning trees, with end hosts splitting traffic over the multiple paths. However, the state in the switches grows linearly with the number of paths, with one Virtual

LAN (VLAN) per spanning tree. In addition, SPAIN relies on the end hosts to balance load and detect failures in an end-to-end fashion.

3.4 Other DAG-based research

The most relevant work here is that of Gafni and Bertsekas, who were first to prove the failure-recovery properties of the node-reversal algorithms. Similar reversal techniques were explored in LMR [15] and TORA [57]. These papers focused on the error-recovery aspects of the approach, but did not study load-distribution nor examine the failure-resilience properties in any depth.

More recently there has been a mini-surge of DAG-based research. Protection Routing [45] optimizes "protectability" (fraction of nodes that will not be disconnected by any single link failure) by carefully constructing an appropriate DAG (with some nondirectional links to be used only for backup paths). The work also explores the load distribution properties of routing along the DAG. O2 [61] was an earlier work that chose a DAG that optimized the probability of finding a working path between any source-destination pair at any given time. Several related pieces of work have somewhat different optimality goals: DIV-R [60] attempted to equalize the number of output ports on all nodes, MRC [44] and MARA [55] optimized the number of paths provided for all source-destination pairs. All these proposals shared the same general goal of maximizing robustness while guaranteeing loop-freeness. In most cases, the optimization boils down to a careful ordering of the nodes to produce an appropriate DAG. Some of this research also looked at load distribution.

Our approach differs in that we don't optimize the DAG itself but instead construct a DAG that performs adequately well under normal conditions and rely on the rapid reversal process to restore connectivity when needed. That is, we prefer to keep the DAG construction algorithm simple and rely on our local failure recovery method as needed. In addition, we devote more focus to the load distribution properties, comparing against the optimal load balancing scheme in a variety of conditions and extending it to multiple classes of service.

3.5 Summary

There is no doubt that these innovations have greatly improved routing. In fact, one might say that, put together, they have made the reliability of current networks acceptable for all but the most stringent requirements. However, there are three problems with accepting these developments as the "final word" in routing:

First, most of these routing improvements have been proposed for a particular context (*i.e.,*multiple spanning trees for L2, MPLS for WANs, etc.), whereas the twin challenges of failure recovery and load distribution hold across all of these contexts. It would be far better to have a unified framework for dealing with these challenges rather than a collection

of narrowly targeted solutions.[3]

Second, these techniques typically provide a limited degree of failure resilience (*i.e.,*they can protect against most single failures, but not multiple failures) which might be sufficient today, but would not scale to more stringent requirements. There isn't a clear "upgrade path" for making these techniques more general (*i.e.,*it isn't clear how to scalably improve Fast Reroute to provide strict connectivity guarantees).

Third, the reason why we don't know how to "upgrade" most of these techniques is that we don't have an intellectual framework for attacking these problems in their full generality. Backup paths are an extremely useful technique, but they don't provide a different way of thinking about routing that leads to more general solutions.

[3]However, proposals like TRILL [4] suggest a trend where the techniques at L2 and L3 are becoming more similar.

Chapter 4

RAD Design

In this chapter we describe the details of RAD design. More specifically, RAD has the following components:

- Build initial DAG: compute a DAG for one destination and make sure every router other than the destination has at least one forwarding choice. A preferred property is to include all shortest paths.
- Repair DAG upon failures: when DAG can not provide connectivity for packet forwarding due to failures, i.e. routing topology is disconnected, locally modify the topology to restore connectivity.
- Optimize DAG: continuous and slower timescale optimization of the DAG topology, to include new shortest paths, increase overall capacity etc.
- Distribute load: spread traffic across multiple choices to avoid congested link.

We start from the model for network and define terms for RAD. Then for each component, we first introduce the intuition and a skeleton of design, then we fill in more details and theory proofs to complete the picture. Various cases are also discussed to ensure the design is valid and robust. Since the DAG is independent per every destination, we focus on one destination in the following text, unless specified otherwise.

4.1 Network Model

Consider a network with nodes $r \in R$ (either routers or switches, but in what follows we will use the term router), connected by symmetric links (although our approach can be generalized to asymmetric links), which provide routing for some set of destinations. The

routing tables for each destination v are computed independently, so in what follows we focus on a particular destination v.[1] We denote the neighbors of node r by $N(r)$.

Network topology is modeled by undirected graph G, while routing topology is a DAG D. Because v is the only node in D that has no outgoing links, we name such D rooted at v, or v oriented. In RAD, the DAG always contains all functioning links, so routing merely consists of choosing the direction of each link. Each node has an identifier i that is invariant, and a routing metric d that is adjusted by the routing algorithm. The direction of a link connecting two nodes is determined by the relative values of the routing metrics: links point from the node with a higher d towards the node with the lower d, and if the d's are the same then the tie is broken by the identifier. In short, every node associates with a value $d.i$ value, and comparing the $d.i$ of a link's two endpoints determines its direction. The fact that these comparisons are transitive guarantees the directed graph is acyclic.

A node r has a set of outgoing links (or ports) O_r and a set of incoming links (or ports) I_r, with every one of r's links/ports in one of these two sets. When the value of d is changed, then the membership of these set would be altered if some of the links are "reversed".

4.2 Build DAG

The problem statement is listed below:

- Given undirected graph G and one of its nodes v
- Find a direction assign for every link in G
- So that the resulting graph is a DAG, and v is the only node that has no outgoing links

4.2.1 General Algorithm

Define a strict order $<$ on nodes, so that node v has the lowest order, while every other node has at least one neighbor with a lower order than itself. In graph terms below:

- For any node $d \neq v$, $v < d$
- For any node $d \neq v$, $min(N(d)) < d$

[1] Destinations could be an individual host (as in L2) or a prefix; in ISP networks where IP prefixes are disseminated using a different protocol — such as iBGP — and the IGP is only used to compute routes between the various routers, the destination can be a destination IP prefix or a set of prefixes reachable from one edge router. RAD could even use an approach similar to TRILL [4] and LISP [18] in which the core routing design provides switch-to-switch delivery, and an auxiliary mapping function determines which switch a host is attached to.

For link (i, j), assign direction following node ordering, i.e. link points from higher order node to lower order one.

- If $i < j$, link points to i, $j \rightarrow i$
- If $j < i$, link points to j, $i \rightarrow j$

Now we explain why above algorithm produces a DAG rooted at v. First, all neighbors of v will have the link points toward it. So v only has incoming links. Second, for any other node, because at least one neighbor has a smaller order than itself, there is at least one outgoing link towards that neighbor.

4.3 Three Rules

A simplified version of our algorithm, call it SRAD, is defined by three rules (the first two of which are simplified versions of the Gafni-Bertsekas algorithm [21]):

- Connectivity (Node Reversal): if a link fails, leaving a node without any outgoing ports, the node then immediately resets its metric as follows:

  $$d_r^{new} = 1 + MAX_{s \in N(r)}[d_s]$$

  Note that this causes all of its links (which must be incoming for node reversal to be invoked) to reverse to become outgoing.

- Optimization (Healing): at periodic intervals, a node will adjust its label as follows:

  $$d_r^{new} = 1 + MIN_{s \in N(r)}[d_s]$$

  This brings the routing label into harmony with those of its neighbors. After sufficient iterations, the metrics will reflect the shortest path distances in the network. Think of this as a slow-motion distance-vector routing algorithm, with lazy updating of routes.[2]

- Load distribution (Local Congestion Avoidance): if all of the outgoing links are congested, the node sends a *congestion* message across its incoming links. The upstream nodes then attempt to redirect some of their traffic to their other outgoing links, thereby lightening the load on the congested node. This definition applies to traffic across all destinations (*i.e.,*the load distribution algorithm does not just consider traffic to a particular destination, it considers all traffic flowing out on each link).

We now explain how these three simple rules define a routing algorithm that provides failure resilience, failure recovery, and load distribution.

[2]One could generalize this step to arbitrary link metrics, but for convenience here and in the rest of the paper we focus on fewest-hop-count paths.

Failure resilience SRAD constructs a DAG which uses every link in the graph, each only used in one direction. This DAG contains all shortest paths; that is, for a particular node, the DAG contains all shortest paths from that node to the destination. It also contains many nonshortest paths.

The fact that the DAG is acyclic means that each node can independently choose which of its outgoing links to use for forwarding a packet; any set of these independent choices will reliably deliver the packet to the destination. Thus, for any node with multiple outgoing ports for a particular destination, it can withstand a failure to any single one of them by merely using the other outgoing links. Later we show, for various network topologies, how many nodes have greater than one outgoing link.

Failure recovery Whenever a failure removes the last outgoing link from a node, the node reversal procedure will, when iteratively applied, restore connectivity (unless that portion of the graph has been disconnected from the destination). By iteratively applied, we mean that when one node does a node reversal (NR), it may cause another node to do so, and so on. The result is the following theorem taken from [21]. (We sketch the proof of this result in Section 4.4, and direct the reader to [21] for the complete proof):

Theorem 2 *Given an arbitrary set of failures, and an arbitrary set of initial metric values d_r, the node reversal process in the component of the network still connected to the destination will terminate in a finite number of iterations and all nodes in that component will have paths to the destination.*

This shows that no matter what the set of failures, or the initial conditions of the metric values, the node reversal process will reconnect the network. This is why RAD does not need to be based on a more traditional global route computation algorithm; instead, the node reversal process is sufficient to establish connectivity. As we shall see in Section 4.7, this allows RAD to recover many failures locally, while still guaranteeing connectivity.

More generally, connectivity will be re-established in a minimal number of steps. That is, among the many other paths being set up by the node reversal process, some of which are not shortest paths, the shortest path to connectivity for the nodes reversing themselves is created by a series of node reversals.

Load distribution While the load distribution algorithm is defined for all traffic, irrespective of destination, we can only theoretically analyze it when all traffic is destined for a particular destination. For the case where all traffic is destined for the same destination v, we define a local congestion-avoiding algorithm as one in which each router, upon detecting an overloaded outgoing link, diverts excess traffic to other uncongested outgoing links, if such uncongested links exist. If all outgoing links are congested, the node sends a congestion message upstream through all of its incoming links, telling these nodes to redirect some of

their traffic elsewhere. We can show that merely avoiding congestion locally leads to optimal throughput (see Section 4.6 for the proof):

Theorem 3 *In our simple model, any local congestion-avoiding algorithm will maximize throughput.*

4.4 Repair DAG

Here we improve the SRAD algorithm in a few small ways to enhance its behavior. First, we note that (as observed in [21]) the naive node reversal algorithm can cause repeated link reversals: if node r reverses, and this causes node s to reverse, then the link between r and s will be reversed twice in a short time. This is because in SRAD, a node always try all its neighbors to re-establish connectivity, including nodes lost their own connectivity and did the reversal. In order to suppress these spurious link reversals, we introduce "generation" to give a global maximum when a node starts reversal. The amended link reversal algorithm is as follows. For each node r, we assign both a metric d_r and a generation g_r. So the value associates with every node is defined as $g.d.i$, where g means the topology generation identifier, d and i are distance and node id respectively. The directions of links are determined by a comparison of $g.d.i$, and the adjustment algorithm is:

- Node Reversal: Whenever a node is left without any outgoing ports, the node then immediately resets its metric as follows:

$$
\begin{aligned}
g_r^{new} &= 1 + MIN_{s \in N(r)}[g_s] \\
d_r^{new} &= \begin{cases} MIN_{s \in N(r),\, g_s = g_r^{new}}\ [d_s] - 1 \\ \quad if there is s \in N(r)\ s.t.\ g_s = g_r^{new} \\ D, otherwise \end{cases}
\end{aligned}
$$

  where D should be a small multiple of the longest path in the network.

- Healing: at periodic intervals, a node that has a neighbor with generation 0, will adjust its label as follows:

$$
\begin{aligned}
g_r^{new} &= 0 \\
d_r^{new} &= 1 + MIN_{s \in N(r),\, g_s = 0}\ [d_s]
\end{aligned}
$$

Theorem 2 still holds, so this slight modfication changes nothing essential about the algorithm but reduces the number of link reversals. The node reversal step is very similar to "partial reversal" of [21] except that we set the metric to D. This value does not affect the behavior of node reversals (as all metrics in a new generation are computed relative to the first metric in that generation), however larger values result in a faster healing process.

Intuitively, this is analogous to the behavior of distance vector - it takes more iterations if several neighboring nodes have too small a metric than if they have too high a metric. The intuition behind the healing step is that we bring all generations back to zero and follow the basic healing process within generation zero. Note that because a destination's generation is always zero, there will always be a node that can heal.

Second, if the network becomes disconnected, the RAD algorithm will continue node reversals indefinitely, trying to re-establish connectivity. The TORA design [57] proposes a complicated disconnection detection algorithm (whose proof of correctness has not been published), but here we opt for the much simpler approach of placing a TTL on link reversals. The node initiating the reversal (*i.e.,*a node that is reversing because of a failure, not because a neighboring node reversed its links) creates a unique ID for the reversal, and this ID is carried along as other nodes have to reverse because of this original reversal. Every time this reversal is passed on, the TTL counter is incremented. By choosing a TTL that is several multiples of the diameter of the network, the node reversal process will be terminated only after any possible connectivity has been found.

Third, if a node only has one physical link, then when that link fails the node is obviously disconnected. To prevent RAD from trying to establish connectivity in vain, nodes that are singly connected (in the underlying topology) signal this to their neighbor, so that when that link goes down no node reversal process is initiated.

Lastly, in the definition of the algorithm we implicitly assumed that reversing a link was an atomic action. In reality, it will take some time for a router to change its forwarding tables; this could cause the case where two nodes disagree on the direction of a link. To prevent this, whenever a node signals a link reversal to its neighboring node, it does so through the following handshake procedure: node r informs node s that the link between them should be reversed (by sending it its updated $g.d.i$ value) but does not forward packets to s yet (instead, packets are dropped until at least one of the link reversals is completed); node s removes the link to r as an outgoing link for that destination, and then confirms the link reversal to r; r then considers the link to s to be an outgoing link to the destination and can send packets to it.

All of these changes are straightforward, and do not change the basic nature of the algorithm but remove some practical problems.

4.4.1 Proof of Ideal Connectivity

Below we prove that given an arbitrary set of failures, the link reversal process in the component of the network still connected to the destination will terminate in a finite number of iterations and all nodes in that component will have paths to the destination.

Consider $G = (V, \overrightarrow{E})$ as the DAG rooted in a destination dst. Remove a set of edges (due to link failures), and denote the resulting graph $G' = (V, \overrightarrow{E'})$. For each node v, let $f(v)$ be the hopcount from v to dst in the undirected graph G'. If v is not in the same component

with dst, $f(v)$ is undefined.

We now show, using an inductive argument, that each node v that is in the same component with dst in G', will be connected to the destination following a finite number of iterations.

Induction base case: Neighbors of dst in $G' = (V, \overrightarrow{E'})$ have $f(v) = 1$, and they all have valid paths to dst. So, neighbors of dst in G' are trivially connected to dst after a finite number of link reversals.

Induction step: Observe that if any node with $f(v) = i$ has valid paths to dst in G', then any node with $f(v) = i + 1$ needs at most one link reversal to connect to d.

So, by induction, all nodes in the same component with dst connect to d after a finite number of link reversals.

Then we prove the second part, that is, if a node starts link reversal due to adjacent link failure, denoted by a, and it belongs to a different component, a detects the disconnection when all its links are reversed again.

It is equivalent to prove a will have all its links reversed again if and only if it belongs to a different component.

First, if a belongs to a different component, all other nodes in the component will not stop link reversal process because none of them has valid paths to dst. So all outgoing links of a after its link reversal will be reversed again by the other end.

If a is in the same component as dst, there exists a path from a to dst, and every link along the path will be reversed at most once. So a has at least one link remains outgoing.

4.5 Maintain DAG

Notice that the Healing step in SRAD, which finds shortest paths, is not used for any of these properties. That is, finding shortest paths is not a necessary component of the RAD algorithm, but merely a useful but optional optimization. This reflects the fact that RAD's separation of concerns which is very different from traditional routing algorithms, for which finding shortest paths is the first, not the last, consideration.

Why is the Healing step included? After a series of node reversals, the DAG no longer contains all shortest paths. Every node that is topologically connected to the destination is also connected via the DAG, but some of the shortest paths may not be in the DAG. The Healing step helps restore these shortest paths to the DAG by iteratively bringing the routing metric into alignment with the true topological distances. One should think of Healing as being run in the background, moving the DAG back towards one which contains all shortest paths.

Notice that SRAD uses one algorithm to quickly establish connectivity (the node reversal step), and another to optimize the resulting paths (the healing step). The node reversal algorithm is guaranteed to avoid loops: more specifically, at no time during the running of SRAD are the forwarding tables such that there is a loop. This is because at all times

the direction of the links is determined by a global ordering of the nodes, and *any* global ordering will prevent cycles. The job of the node reversal algorithm is to find an ordering that provides connectivity. The job of healing is to adjust this ordering so that it includes all shortest paths (and it does so by driving the labels d_r to reflect the shortest path distances). Thus, the hierarchy of concerns in RAD is:

- Loop avoidance: built in to the definition of link directions, never violated.
- Connectivity: achieved by the node reversal algorithm, which is run whenever a node gets disconnected and terminates only when connectivity is achieved.
- Shortest paths: achieved by repeated applications of the Healing process.

In contrast, the classical distance vector routing model reverses this hierarchy. The first job of the routing algorithm is to find shortest paths, and only when these paths are shortest does the algorithm guarantee that connectivity is achieved and no loops are present. Thus, cycles may be created during the iteration of the algorithm before convergence has been reached. Much of the work that has gone into producing loop-free distance-vector algorithms implicitly involve a DAG, but in the generic distance-vector algorithms loops can be created. Moreover, the fact that connectivity is not guaranteed until these optimal paths have been computed (*i.e.,*the algorithm has fully converged) means that restoring connectivity may take a while.

In SRAD, the node reversal process can more quickly establish connectivity without having to worry about cycles, and the healing process can come along later to improve the DAG. We believe that this separation of concerns is an important one, and is similar in spirit to that advocated in consensus routing [37].

4.6 Distributing load

In terms of congestion, a node considers all traffic (not just traffic to a specific destination) and the actions below are not specific to individual destinations. As in SRAD, the basic load distribution scheme is twofold:

- When one outgoing link is congested, the node attempts to spread the traffic to uncongested output links (the congestion applied to all flows passing through that link, and the node can decide which traffic to move).
- When all outgoing links are congested, the node sends a congestion message to one or more of its neighbors. These neighbors treat this message as if that link were congested, and try to spread some of their load to other uncongested links.

In our implementation of RAD, these congestion signals are triggered when the utilization is above some threshold and contain an explicit request for the receiving router to reduce traffic by a certain percentage.

Note that this load distribution approach does not attempt to minimize maximum utilization (which is what we call *load balancing*), it merely attempts to avoid congestion (which is what we call *load distribution*). Each approach has its advantages; load balancing minimizes the maximal load on links, while load distribution allows the network to use the shortest paths until they become congested, and only takes nonshortest paths as needed.

When we discuss the performace of our load distribution scheme, we compare against an optimal load balancer which maximizes a particular utility function we use to measure the "goodness" of the traffic distribution. The RAD load distribution approach is not tuned to optimize this, or any other, utility function; it merely avoids overloaded links. Thus, one might not expect the RAD load distribution method to work particularly well at this optimization task. However, we find that RAD performs surprisingly well despite its ignorance.

4.6.1 Proof of Optimal Load Distribution

We consider a simplified network model described by the following definitions. The network is a DAG $G = (V, \overrightarrow{E})$ rooted in a destination d. There are n source nodes $s_1, \ldots, s_n$, and each has a *demand* $dem_i \in R_+$ (amount of traffic it wishes to send). Each edge $\overrightarrow{e} \in E$ has *capacity* $c_e \in R_+$.

Definition 1 (flow patterns) *A* flow pattern *is a function* $f : \overrightarrow{E} \times [n] \to R_+$*, such that:*

- *Edge capacities are not exceeded:*
  $\forall e \in \overrightarrow{E}$, $\Sigma_{i\in[n]} f(e,i) \leq c_e$.
- *Demands are not exceeded:*
  $\forall s_i$, $\Sigma_{e|e=(s_i,v)} f(e,i) \leq dem_i$.
- *Incoming traffic equals outgoing traffic:* $\forall v \neq d \in V$, $\forall s_i \neq v$, $\Sigma_{e|e=(u,v)} f(e,i) = \Sigma_{e|e=(v,u)} f(e,i)$.

Definition 2 (link utilization) *Let* f *be a flow pattern. For every edge* $e \in \overrightarrow{E}$*,* e*'s* utilization *in* f*,* $f(e)$*, is defined as* $f(e) = \Sigma_{i\in[n]} f(e,i)$.

Definition 3 (constrained edges, paths) *We say that an edge* $e \in \overrightarrow{E}$ *is* constrained *in a flow pattern* f *if* $f(e) = c_e$ *(i.e., the edge is fully utilized). We say that a path* P *in* G *is constrained in a flow pattern* f *if one of the edges on* P *is constrained in* f.

Definition 4 (throughputs, satisfaction) *We define* s_i*'s* throughput *in flow pattern* f*,* $\alpha_i(f)$*, to be*
$\Sigma_{e|e=(s_i,v)} f(e,i)$*. If* $\alpha_i(f) < dem_i$ *we say that* s_i *is not* satisfied *in* f.

We now define two natural local properties of flow patterns: *local optimality* and *congestion-avoidance*. Intuitively, these two properties correspond to natural requirements from any flow allocation (load balancing) algorithm: (1) that, in the outcome of the algorithm, no source node be able to get more throughput simply by transmitting at a higher rate (sending more flow), and (2) that if a source node encounters congestion along its path, and has an alternative, uncongested, path, it make use of it.

Definition 5 (locally-optimal flows) *Flow pattern f is* locally-optimal *if, for every source node s_i that is not satisfied in f, there is no path P from s_i to d that is not constrained in f.*

Definition 6 (congestion-avoiding flows) *Flow pattern f is* congestion-avoiding *if, for every dource node s_i, there are no two paths P and Q from s_i to d such that P is constrained in f, and Q is not.*

Informally, global optimality of a flow pattern means that the sum of sources' throughputs is maximized.

Definition 7 (globally optimal flow patterns) *Flow pattern f is* globally optimal *if there is no flow pattern f' such that $\Sigma_{i\in[n]}\alpha_i(f') > \Sigma_{i\in[n]}\alpha_i(f)$.*

We can now state and prove the result.

Theorem 4 *If a flow pattern is locally-optimal and congestion-avoiding then it is globally optimal.*

Proof 1 *Let f be a flow pattern that is locally-optimal and congestion-avoiding. By contradiction, we assume that f is* not *globally optimal. We now reduce our problem to a single-source-single-destination max-flow problem as follows:*

1. **Network.** *Construct a network $G' = (V', \overrightarrow{E}')$, where $V' = V \cup \{s\}$, and $\overrightarrow{E}' = V'^2$ (*i.e.*, G' is a complete graph).*
2. **Edge capacities.** *Each edge $e \in \overrightarrow{E}$ has a capacity of $c'_e = c_e$ in G'. For every $i \in [n]$, the edge $e = (s, s_i)$ has capacity $c'_e = dem_i$. All other edges have capacity $c'_e = 0$.*
3. **Flow.** *We define the flow $f' : E \to R_+$ from s to d as follows: For every $i \in [n]$, $f(s, s_i) = \alpha_i(f)$, and $f(s_i, s) = -\alpha_i(f)$. For every u such that $u \neq s_i$ for all i's, $f'(s, u) = f'(u, s) = 0$. For every other edge $e = (u, v) \in V^2$, $f'(u, v) = f(u, v) - f(v, u)$, where $f(e)$ is defined to be 0 if $e \notin \overrightarrow{E}$.*

Observation 4.6.1 *By our definition of flow pattern (Def. 1), and the construction of f', it holds that f' has the following required properties of flows:*

- **Capacity constraints.** $f'(e) \leq c'_e$ *for every* $e \in \overrightarrow{E}'$.
- **Skew symmetry.** $\forall (u,v) \in \overrightarrow{E}'$, $f'(u,v) = -f'(v,u)$.
- **Flow conservation.** $\forall u \neq s, d \in V$,
  $\Sigma_{v \neq u \in V'} f'(u,v) = 0$.

Observation 4.6.2 *f' is a maximum flow in G' iff f is globally optimal in G.*

Hence, our assumption that f is not globally optimal implies that f' is not a maximum flow in G'.

The residual network and max-flow min-cut. *We now consider the* residual network *for f', denoted by $R(f')$, obtained from G' by setting the capacity of each edge $e \in \overrightarrow{E}'$ to be its* residual capacity *$c'_e - f'(e)$. By the max-flow min-cut theorem, f' is a maximum flow in G' iff there is no* augmenting *path in $R(f')$, that is, a no path from s to d in $R(f')$ such that every edge on that path has positive capacity in $R(f')$. Therefore, the fact that f' is not a maximum flow implies the existence of an augmenting path P must exist in $R(f')$.*

Let P be the path $s = q_o, q_1, \ldots, q_k = d$.

Lemma 4.6.3 *For every $1 \leq j \leq k-1$, there is a path from q_j to d in G that is unconstrained by f.*

Proof 2 *Consider the edge $e = (q_{k-1}, q_k)$. Observe that because $q_k = d$, and the construction of f', it must be that e has positive capacity in $R(f')$ because $f(e) < c_e$. Hence, there is an unconstrained path in G leading from q_{k-1} to d (specifically, the direct path (q_{k-1}, q_k)).*

Now, consider the edge $e = (q_{k-2}, q_{k-1})$. By definition of P, this edge, too, has positive capacity in $R(f')$. Observe that, by the construction of f', this can only happen in one of two cases: (1) $f(e) < c(e)$, and (2) $f(q_{k-1}, q_{k-2}) > 0$. In case (1), because q_{k-2} has an unconstrained edge to q_{k-1} in G, and q_{k-1} has an unconstrained path to d in G, we have that q_{k-2} also has an unconstrained path to d in G. In case (2), since q_{k-1} has an unconstrained path to d in G, and is sending traffic through q_{k-1} in f, the fact that f is congestion-avoiding implies that q_{k-1} must also have an unconstrained path to d in G. The lemma now follows from applying the same argument to q_{k-3}, q_{k-4}, etc.

To conclude the proof, consider the edge (q_o, q_1). Because this edge in on P it must have positive capacity in $R(f')$. Observe that, because $q_0 = s$, and the construction of f', this can only happen if q_1 is some source node s_i that is not satisfied in f. However, by the above lemma, q_1 has an unconstrained path to d in G — a contradiction to the local optimality of f.

4.6.2 Stability

In the traffic engineering literature, stability is usually defined as the ability to reach a steady state, i.e. the utilization of every link does not change, under constant network traffic. Because of its practical impact on network operation, stability is a major concern of many load distributing algorithms. From the literature, one can easily discover that approaches that adopt global optimization seldom have stability issue, while adaptive schemes usually have more difficulties to avoid oscillation. However, the root cause is not whether a method is local or adaptive, but quite often, they try to do local optimization. When local optimizations generate conflicting results, the decisions affect each other asynchronously, hence the unstable state. As a comparison, ECMP always spreads load evenly across multiple choices, and does not suffer from the oscillation problem.

The load distributing method of RAD follows a similar principle: avoid local optimization. Instead of minimizing link utilization or reduce some sophisticated penalty function, RAD merely avoids congestion, i.e. offloads excessive traffic to another choice. The utilization of previously congested link also stays close to the threshold that triggers offloading. [3] Since the mechanism does not actively reduce load on links, the load distributing results are similar to ECMP with weights on links, i.e. unequal but deterministic load among links.

4.7 Evaluation

We now describe our experiment results, for which we used 6 ISP topologies and 3 datacenter topologies as shown in Table 4.1.

For datacenter topologies, we picked the classical 3-tier hierarchical topology recommended by Cisco Inc. ([11]) as well as two different topologies from recent proposals: FatTree ([6]) and VL2 ([28]). While all three datacenter topologies are highly symmetric, there is little similarity beyond that. Cisco topology has a small core of high-end routers. VL2 topology has a much larger and denser core. While FatTree topology separates aggregation switches into pods.

We analyze five aspects of RAD's performance: resilience, scope of link reversal, load-distribution, control traffic, and dynamic response.

4.7.1 Connectivity

Outdegree and local resilience The question we ask first is: how often can RAD recover locally from a single link failure by using another outgoing link?[4] We know that there must be some occasions when a single failure disconnects a node (and forces it to initiate a

[3] An analogy for the mechanism is filling multiple buckets with water, and the rule is not using a new bucket until all used buckets are full.

[4] In answering this question, we ignore the nodes that are physically connected by a single link, because there is no way routing can help them.

Topology	Nodes	Edges	Avg. degree
AS1221	83	131	3.16
AS1239	361	1479	8.19
AS1755	111	234	4.22
AS3257	151	288	3.81
AS3967	91	180	3.96
AS7018	382	1299	6.80
Cisco	76	160	4.21
FatTree	80	256	6.40
VL2	88	256	5.82

Table 4.1: Topologies Used for Simulations

Topology	SP FIB	DAG NR
AS1221	10.21	7.19
AS1239	41.43	2.09
AS1755	19.92	10.88
AS3257	21.18	9.64
AS3967	26.18	10.94
AS7018	32.10	2.92
Cisco	22.63	1.05
FatTree	33.12	8.12
VL2	10.80	0.57

Table 4.2: Average scope of Shortest Path FIB change and RAD Node Reversal when link fails

Topology	DV	RAD
AS1239	2.26	0.02
AS1755	0.88	0.11
Cisco	0.02	0.02
FatTree	0.32	0.13

Table 4.3: Average number of messages triggered by link failure, shown as ratio to link-state's performance.

node reversal) because, as mentioned earlier, for each DAG there is at least one node with DAG outdegree one. Figure 4.1 shows for each topology and averaged over all DAGs (*i.e.*,a

separate DAG per destination), the percentage of resilient nodes (outdegree bigger than one), and "redundant" links (whose failure does not trigger a node reversal). In the two larger ISP topologies (AS1239 and AS7018), over 90% of the nodes are resilient and over 95% of the links are redundant. In the other four ISP topologies, the resilient node percentage is between 60% and 65%, while the redundant link percentage is roughly 80%. The datacenter topologies Cisco and VL2 have over 97% resilient nodes and redundant links. In contrast, FatTree has only about 30% resilient nodes, mainly because in any particular FatTree pod all aggregation layer switches have outdegree one to each destination edge switch.

(Free) Failure Resilience Before showing how much failure resilience is free, we would like to note that "recovery resilience" is completely free. By "recovery resilience" we mean that link recovery should not disrupt communication. In RAD, this is obviously the case - when a link comes up, it is added to the DAG without affecting anything else. In traditional routing protocols, however, link recovery can cause path recomputation, temporary loops, and packet losses.

Turning to failure recovery, we ask how often can RAD recover from a link failure locally (because there is another outgoing link present) and how often does a node have to initiate node reversal? As mentioned earlier, any DAG has at least one node with outdegree one, but how many other nodes are vulnerable to a single link failure?[5]

Figure 4.1 shows the percentage of resilient nodes (those with outdegree bigger than one), and redundant links (those whose failure does not trigger node reversal). The data for each topology has been averaged over all DAGs - one DAG per destination.

Among ISP topologies, two large ones (AS1239 and AS7018) have over 90% resilient nodes and over 95% redundant links In the other four topologies, the node percentage varies between 60% and 65%, and link percentage stays around 80%. Among datacenter topologies, Cisco and VL2 both have node and link percentages greater than 97%. In contrast, FatTree has roughly 30% of nodes with outdegree one in each DAG because in any FatTree pod all aggregation layer switches have a single link to the destination's edge switch. Also note that these numbers were produced with RADs simple DAG construction. It is possible to construct DAGs with even higher failure resilience.

Scope of failure recovery We now investigate the impact of single failures in a different way; when a link fails, how many nodes need to respond? For RAD we measure how many nodes had to invoke a reversal. To benchmark these results, we compare against shortest-path routing, where we measure the number of nodes whose outgoing port changed (we denote this as a change in the FIB). In datacenter topologies, because they use ECMP, we include all cases where the ECMP group changes (the set of shortest path ports). This is not

[5]In answering this question, we ignore the nodes that are physically connected by a single link, because there is no way routing can help them.

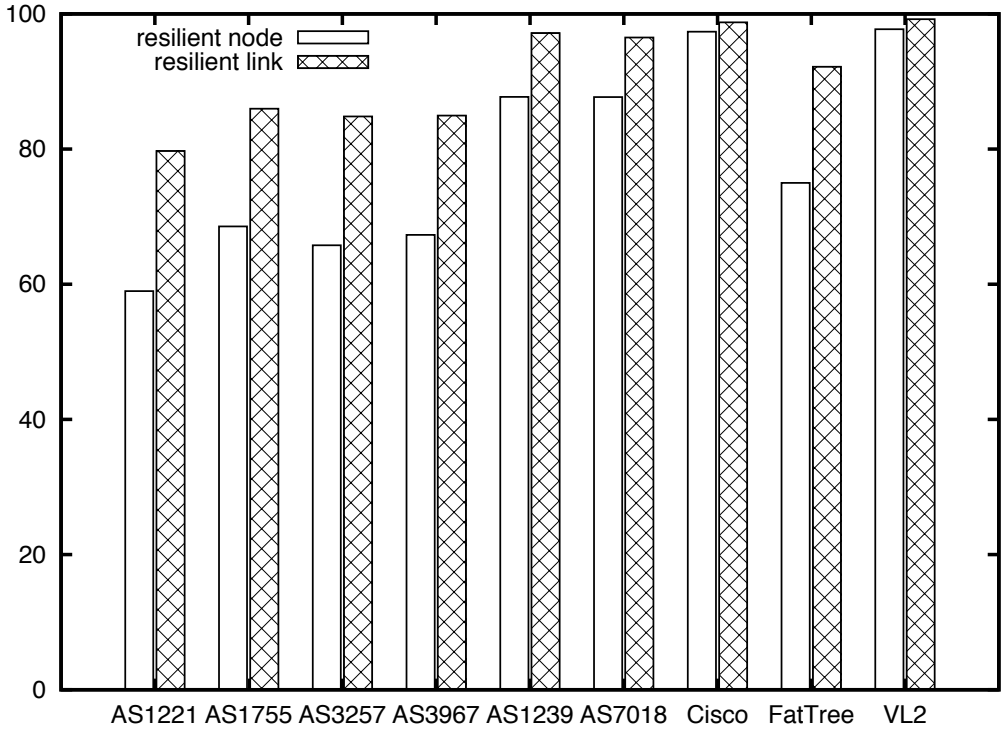

Figure 4.1: Percentages of resilient nodes and redundant links in DAGs over various topologies

an entirely fair comparison, because node reversals and FIB changes are somewhat different, but they both measure the scope of response in some way.

Note the scope only includes nodes that need to make changes to its RIB or FIB, which is independent of how messages are exchanged. So the scope analysis also applies to link state routing, e.g. although the message is broadcasted to every node, only a fraction of the nodes actually need to make changes to RIB or FIB.

Now we consider when link fails, how many nodes need to be aware of this failure and update their RIB/FIB accordingly. The scope of failure events is important to verify RAD's scalablility, and the scope of node reversal determines RAD's recovery time.

We compare to shortest path routing and compute all pairs shortest paths for every topology. After a link failure, if a node has a different distance to some destination, we add it to the set of nodes that have RIB change. If the node also needs to use a different link to reach the destination, we add it to the set of FIB change nodes. In datacenter topologies, because ECMP is used explicitly, if a node changes an ECMP group after a failure, we also consider it as a FIB change. Note the scope only includes nodes that need to make changes to its RIB or FIB, which is independent of how messages are exchanged. So the scope analysis

also applies to link state routing, e.g. although the message is broadcasted to every node, only a fraction of the nodes actually need to make changes to RIB or FIB.

In RAD, if a node has its available outgoing ports changed, we add it to RIB change set, and if the node needs to use a different outgoing port, it is added to FIB change set. Particularly if a node has no outgoing ports available, it will start node reversal. So the set of nodes that do reversal is a subset of nodes that have FIB changed, which is a subset of nodes that have RIB changed.

First, we show the percentage of nodes responding averaged over all single link failures in Table 4.2, where RAD's scope is always much less than shortest-path (although, again, these two quantities are not directly comparable). These average results give some idea of general trends, but they hide the distributions. An examination of the scope distributions in Figure 4.3-4.6, where the link events are ordered in terms of the percentage of responding nodes, shows a variety of behaviors. On the large ISP 1239, most events created very few node reversals, but the tail of the distribution had over 30% of the nodes responding. For the smaller ISP 1755, the distribution of node reversals is much smoother. On Cisco, RAD caused very few node reversals, whereas for FatTree there were more (due to its poorer connectivity).

To summarize, the average scope of RIB/FIB changes varies with topology, but RAD always has many failures that require little recovery. For failures that disconnect a large fraction of nodes, both RAD and shortest path routing need to involve more nodes for recovery. That is, RAD finds the cases where failures can be recovered locally, but both RAD and shortest path have cases where failures are nonlocal.

Scope of Link Reversal: While each individual link-reversal is a local decision, the process can cascade to other routers (but does so at hardware speeds, so packets are not dropped while this process is ongoing). In fact, the process continues until connectivity is established or a disconnection is confirmed. A natural question is how far does this cascading process go before connectivity is re-established. We investigated this question on our four topologies under multiple failure conditions (either one random failure, or two random failures, or five random failures) by measuring the number of nodes whose $L_r(v)$ was changed for any v.

The various graphs in Figure 4.2 have significantly different behavior. At one extreme, the link reversals on AS1239 and the Cisco topology are of limited scope under all conditions. At the other extreme, the link reversals on the Fat-Tree and AS1755 spread significantly more widely.

Messages We have not precisely defined the message exchanges in RAD, so we can't make any precise statements about its control overhead. However, if we assume a node reversal is a single message, we can give a rough estimate of how it compares to standard distance-vector and link-state routing algorithms in terms of the messages sent when responding to a failure. In link-state, if we assume that link updates are sent with maximum efficiency,

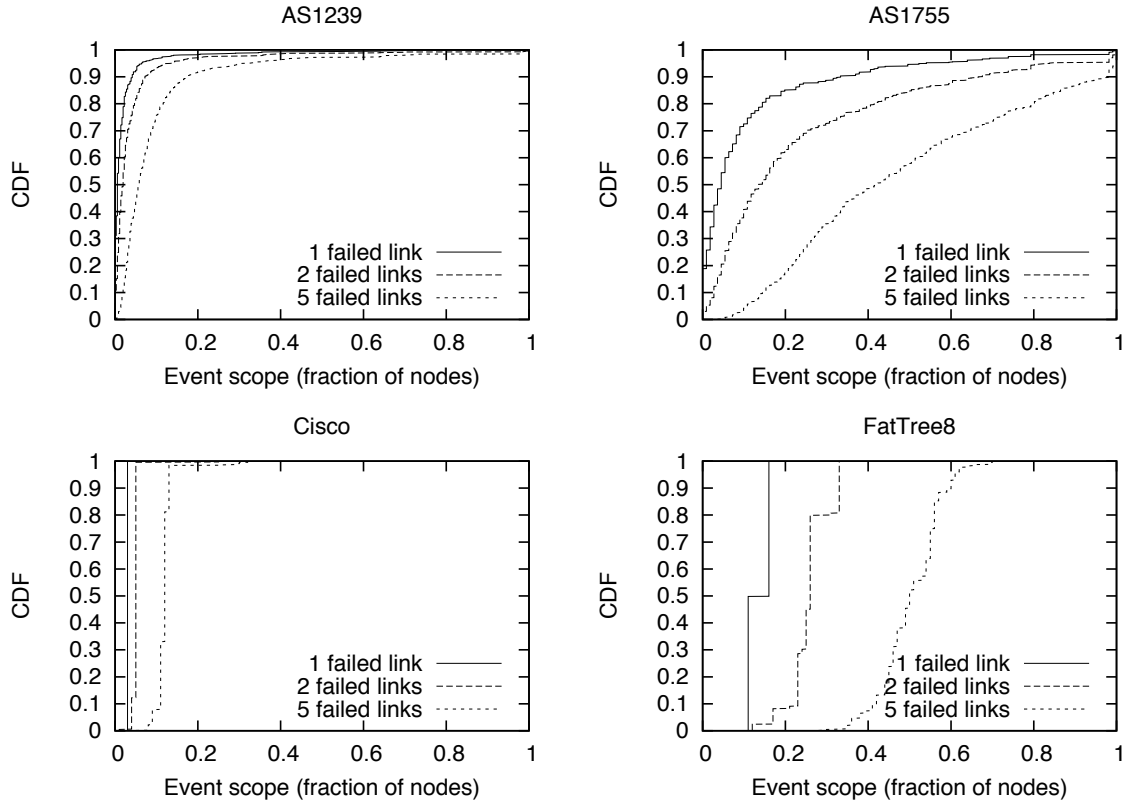

Figure 4.2: CDF of link-reversal scope under multiple concurrent failures. The x-axis is the fraction of nodes that experience changes in their $L_r(v)$ as a result of these failures.

it takes $N-1$ messages to reach all N nodes in the network. We consider four sample networks and randomly fail links, one at a time. In Table 4.3 we show the ratio of messages in distance-vector to that in link-state, and similarly the ratio of control messages in RAD to that in link-state, over all four topologies.

On the Cisco network, both distance-vector and RAD require very few messages, because there are so many alternate paths available, and on the FatTree RAD shows an improvement ratio of only 2.5. However, on the more irregular AS networks, RAD outperforms distance-vector significantly, with 10% messages on AS1755 and only 1% messages on AS1239.

4.7.2 Load Distribution

We now investigate how RAD can distribute load by first comparing it to an optimal load balancing algorithm and then looking at RAD's performance in the face of network load and topology changes.

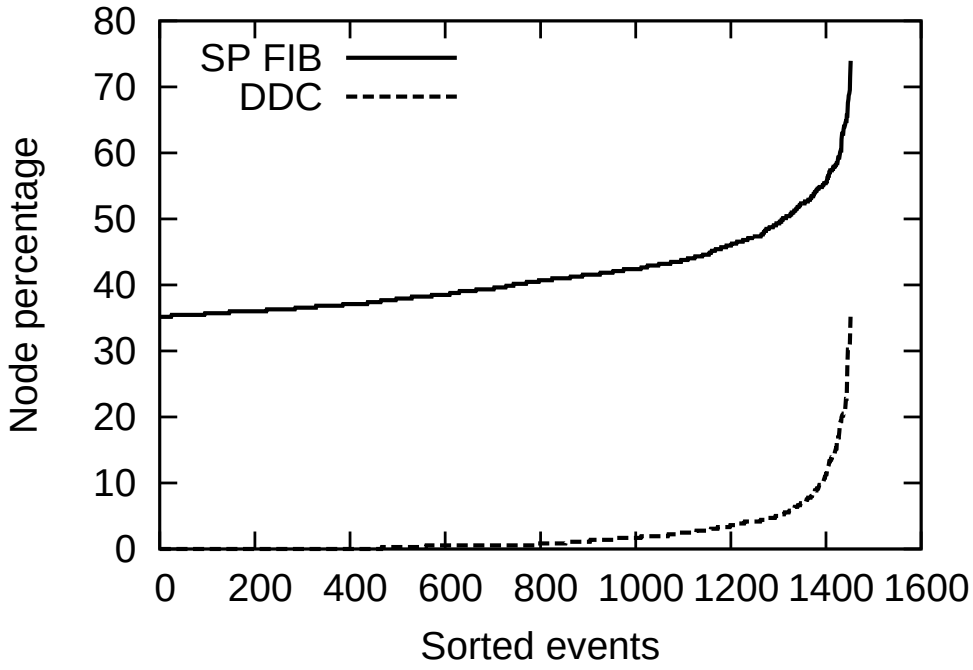

Figure 4.3: Scope of NR/FIB changes on AS1239

RAD vs. Optimal We evaluate RAD's ability to distribute load by comparing it to an "optimal" load balancing algorithm that uses a linear programming model from the traffic engineering literature (e.g. [19]). This model assigns a penalty for various usage levels on each link, with the penalties rising steeply as the link becomes overloaded; the load balancing mechanism arranges the traffic so as to minimize the total penalty. In order to make the optimization problem tractable, links are allowed to carry more than 100% of their capacity (but at a heavy penalty). Note that RAD was designed merely to distribute load when links get overloaded; for instance, it does not try to balance the load on two outgoing links as long as neither is overloaded. Moreover, RAD is not trying to minimize some given utility function, whereas the optimal algorithm is. Nonetheless, we ran several experiments to measure how close to optimal RAD's load distributions are.

We use the same 9 topologies for our experiments and set each link's capacity to 10k units. For the load model, we had each source sending to a random destination at a constant rate chosen uniformly randomly between 1 and 1000. To investigate load balancing at different load levels, we used a traffic matrix λT where T is the original traffic matrix and $1 \leq \lambda \leq 4$. To get a sense of how much the network is actually loaded with a given traffic load, we measured the fraction of links that had over 90% utilization in the optimal solution.

Figure 4.7 shows the results on four topologies, two ISPs and two datacenters. On the

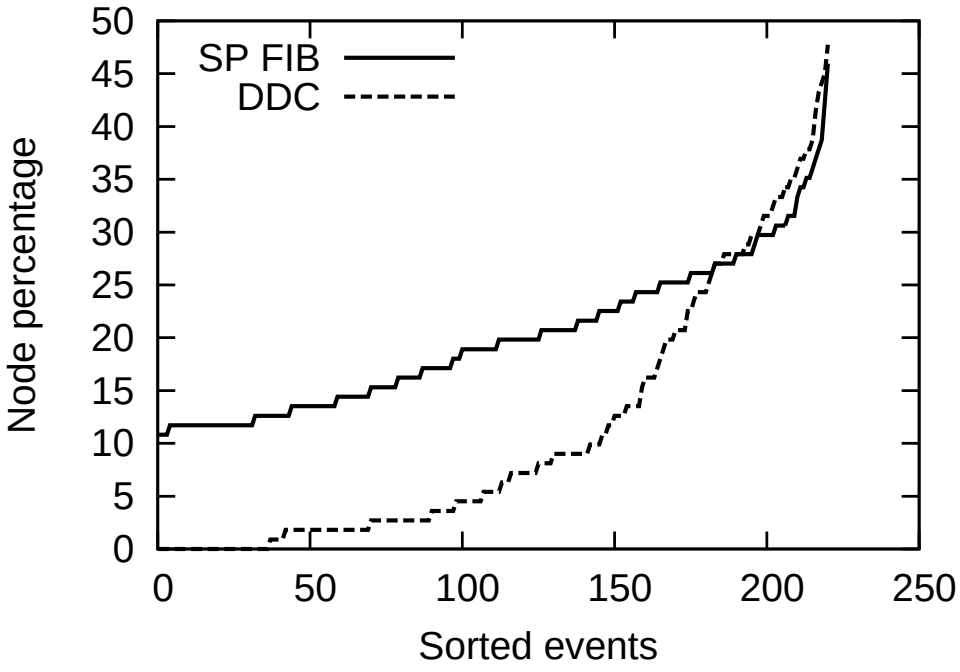

Figure 4.4: Scope of NR/FIB changes on AS1755

AS1239 topology, RAD tracks the optimal solution quite well. It similarly does well on the AS1755 topology, although under high loads (when over 8% of the links are over 90% utilized) the gap becomes significant. On the Cisco topology RAD tracks the optimal even under high loads (as many as 20% of the links highly loaded) presumably because there isn't much path diversity available so both the optimal solution and RAD are highly constrained. It is only on the FatTree topology where RAD significantly departs from the optimal solution. This gap occurs because the FatTree topology provides a wealth of alternate paths with lengths much longer than shortest paths; RAD simply does not explore these paths, while the optimal solution manages to locate the underloaded edge links and utilize them. Note that such paths are typically disallowed to avoid loops, as seen in [6] and [53]. Also, the fraction of overloaded links is extremely small, so that while RAD is suboptimal, it is succeeding in its goal of avoiding overloaded links.

Figures 4.8 and 4.9 show the distribution of link utilizations produced by RAD and the optimal load distribution for $\lambda = 3$ and $\lambda = 4$ on the AS1755 topology (similar patterns are observed in other ISP topologies). RAD's curve follows the optimal one very closely and, more importantly, RAD's curve almost overlaps the optimal curve for highly loaded links. This suggests that RAD does not produce more highly loaded links than the optimal solution, it merely doesn't spread the load among the more lightly loaded links in the same

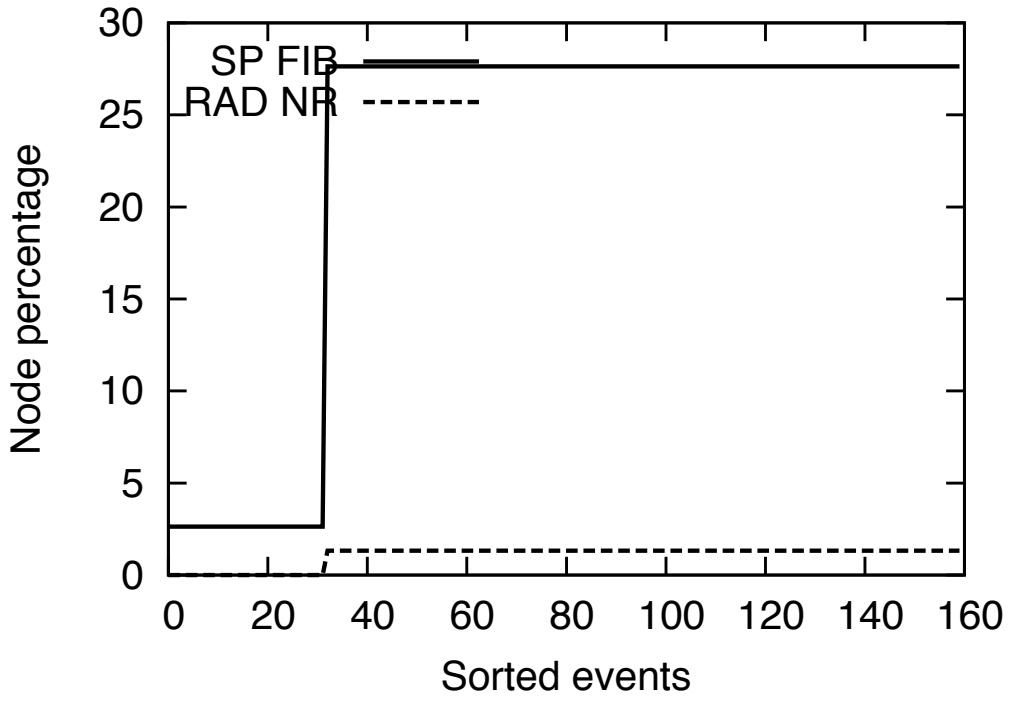

Figure 4.5: Scope of NR/FIB changes on Cisco

way that the optimal solution does. Also visible in the curve is the fact that the optimal solution is carefully tuned to the particular utility function; the optimal solution has many links at load levels where the utility function changes slope (the horizontal plateaus in the optimal curve correspond to inflection points in the penalty function), while RAD produces a smooth distribution.

Traffic classes In this set of experiments we separated traffic into three classes of

First we change the optimization model to assign class dependent penalty functions: On link l, **may need a table for all the notations**

- Latency sensitive: $p_l = C * \phi(t_l) + a * \phi(t)$
- Bandwidth sensitive: $p = \phi(t)$
- Best effort: $p = b * \phi(t)$

We also change the traffic to include 10% latency sensitive traffic, 40% bandwidth sensitive and half best effort. Then the traffic is scaled up to four times original, when the optimal solution overloads a link. Figure 4.10 shows RAD with its local load distribution

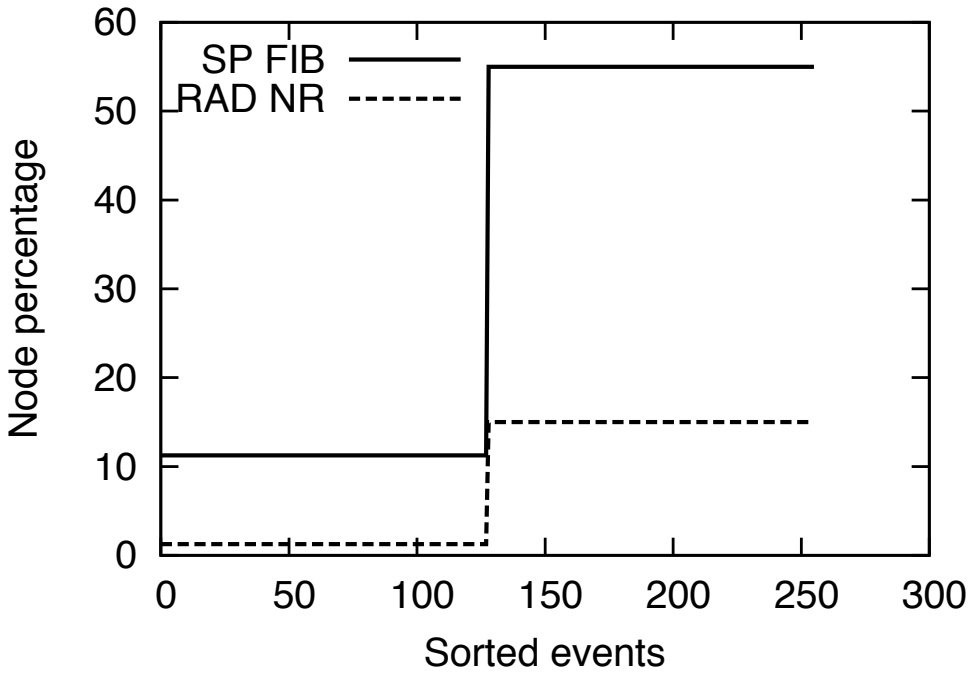

Figure 4.6: Scope of NR/FIB changes on FatTree

mechanism provides close to optimal results, and the difference is within 10% (the same is true for AS1239 topology) , even when penalty explodes due to links are overloaded.

Network Dynamics If the traffic matrix and network topology are static, optimal solutions can distribute load very well, but once the traffic matrix changes or links fail, the precomputed distribution can cause significant congestion. The usual practice is to recompute the optimal solution after the change[6]. However, such offline calculations take time while, in contrast, RAD's local mechanisms can react to the change instantaneously.

We evaluate two types of network dynamics: link failures (Figure 4.11) and bursts in demand (Figure 4.12). Both experiments were performed on the AS1755 topology under the traffic matrix described above with $\lambda = 3$. This load generates the link utilization shown in Figure 4.8. For link failure experiments, we randomly picked a link that carries some traffic and removed it from the graph. We then recomputed the penalty using the optimal algorithm and RAD. For demand burst experiments, we randomly picked a flow and increased its demand by a number of units picked uniformly from [1500, 2500], thus

[6]There is abundant literature on precomputing routing that behaves reasonably well in dynamic networks (e.g. [72], [10]), however their static nature still results in significant performance degradation (30% - 90%) even when only one or two links fail.

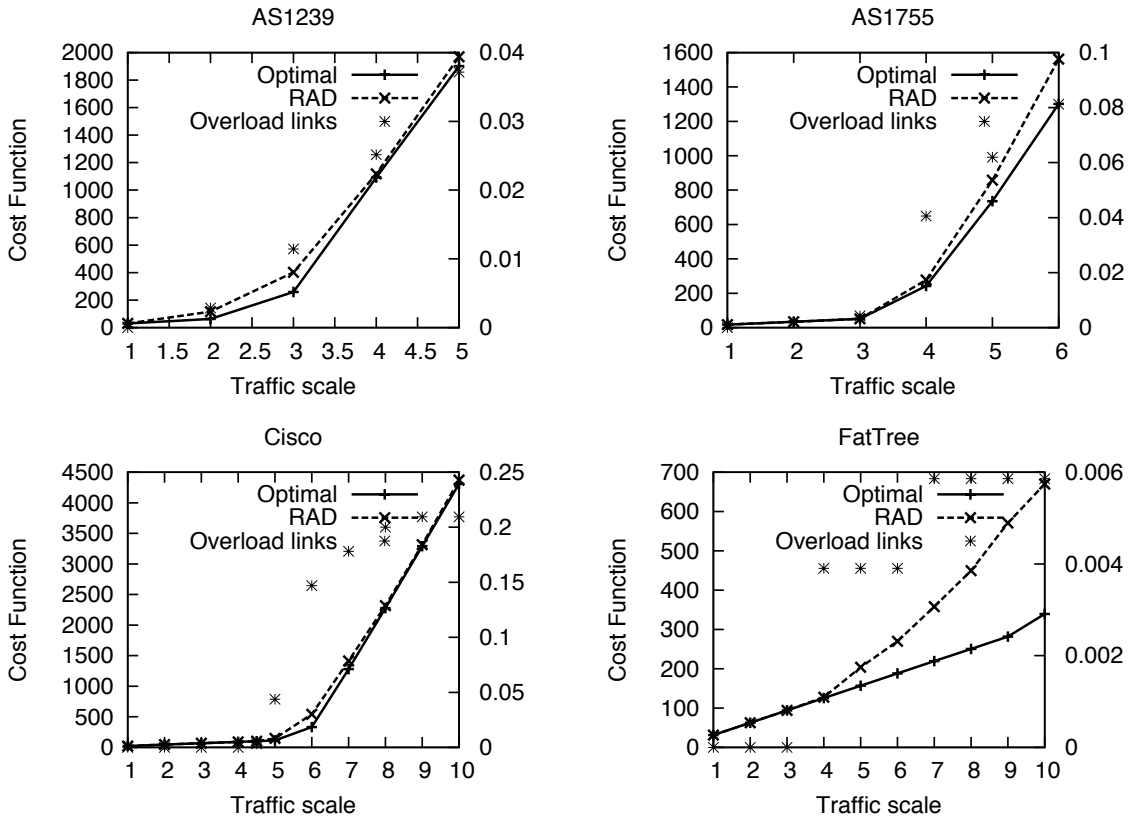

Figure 4.7: Comparison of RAD to the optimal linear programming solution. The dots (using the right hand scale) indicate the fraction of links with over 90% utilization in the optimal solution (which gives an indication of how heavily loaded the network is).

increasing the demand on average by a factor of 2.33.

Figures 4.11 and 4.12 show a representative sample of events comparing RAD's performance versus the reoptimized optimal solution (*i.e.,*the optimal solution that is computed after the change event). In all cases, RAD is very close to the optimal with the average difference of 7.8% for link failure experiments and 7.1% for demand burst experiments. Note that RAD is able to achieve these results with a purely local algorithm that distributes load but does not balance it, knows nothing about the penalty function, and does not require centralized recomputation.

While performing these experiments, we measured how many times each node adjusts the load distribution among its outgoing links. In no experiment did any node adjust its distribution more than 10 times and the average number of adjustments was less than half that. This suggests that RAD's load distribution does not suffer from slow convergence or

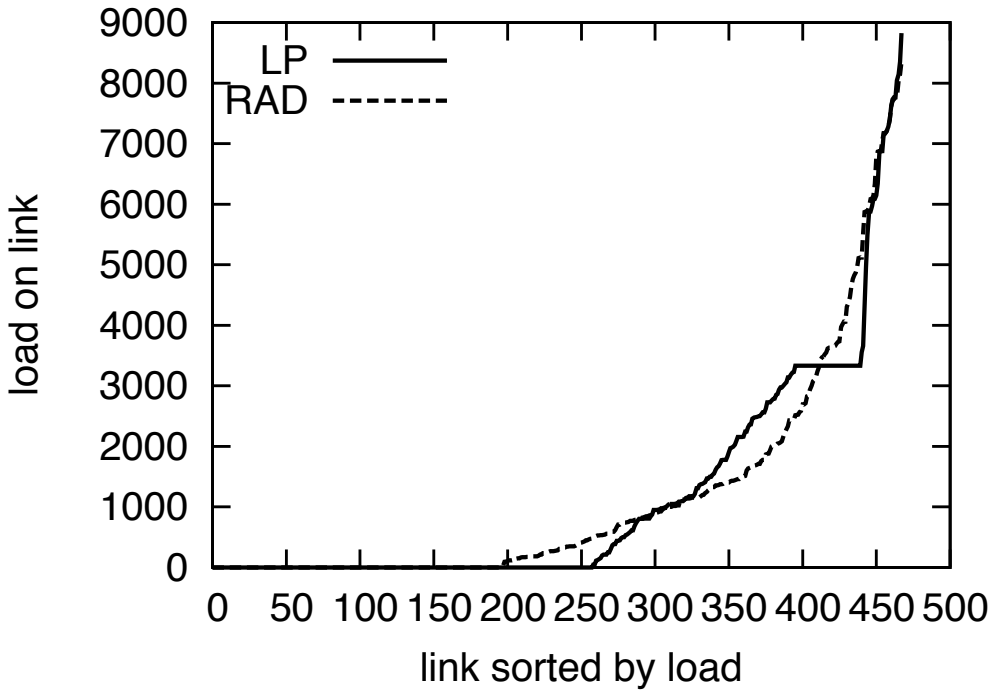

Figure 4.8: RAD and optimal link loads, with links ordered by utilization and $\lambda = 3$

oscillations.

Finally, we compare RAD's characteristics with that of traditional routing approaches following a link recovery event. While handling a link recovery event, traditional routing approaches can create transient loops that disrupts communication ([20], [70]). RAD, on the other hand, simply adds a recovered link to the DAG without causing any disruptions. This is because of the hierarchy of concerns; loop-freeness is always preserved, even under change events.

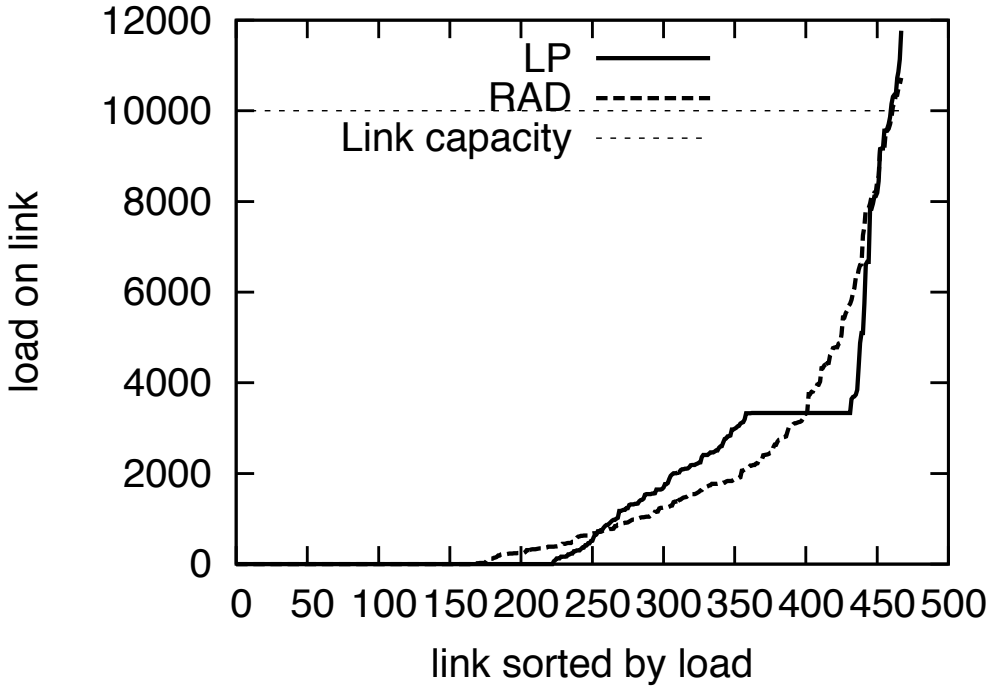

Figure 4.9: RAD and optimal link loads, with links ordered by utilization and $\lambda = 4$

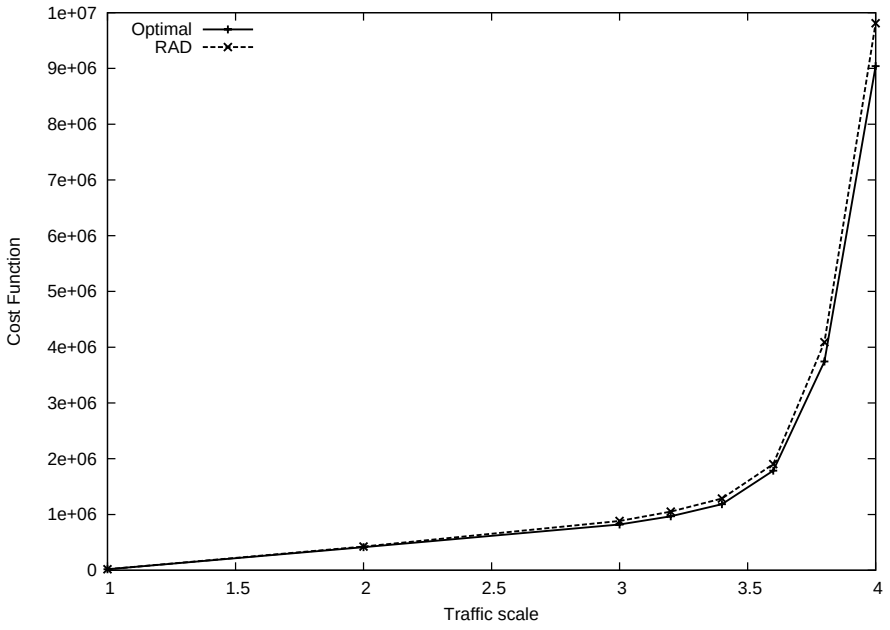

Figure 4.10: 3 classes of traffic, RAD vs optimal, on AS1755

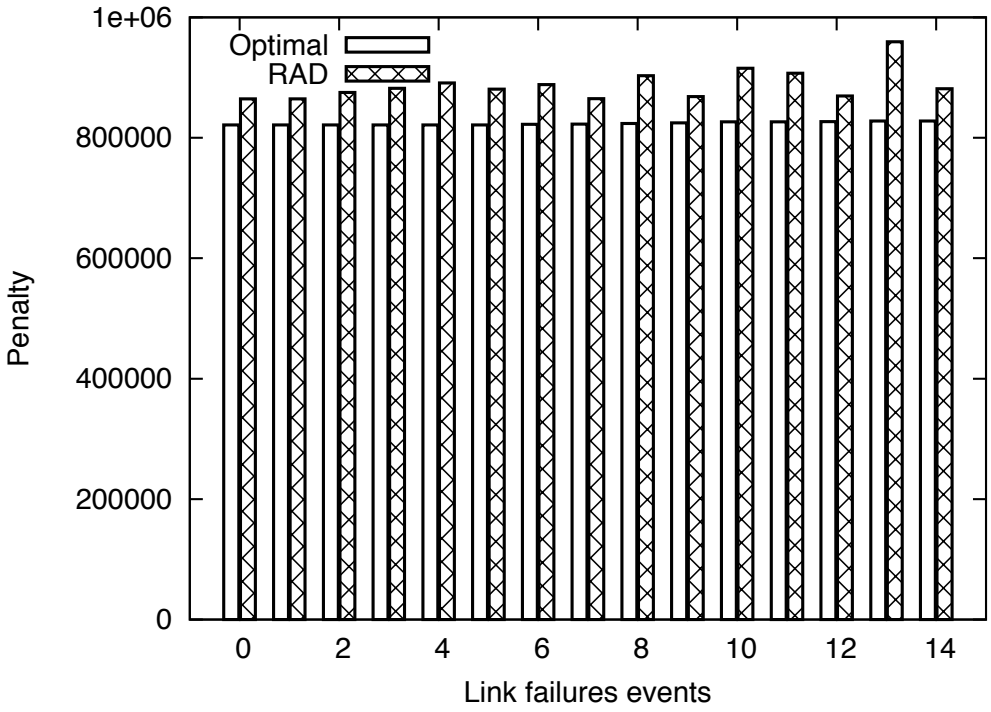

Figure 4.11: Penalty change when link fails: RAD vs optimal sorted by new optimal penalty value.

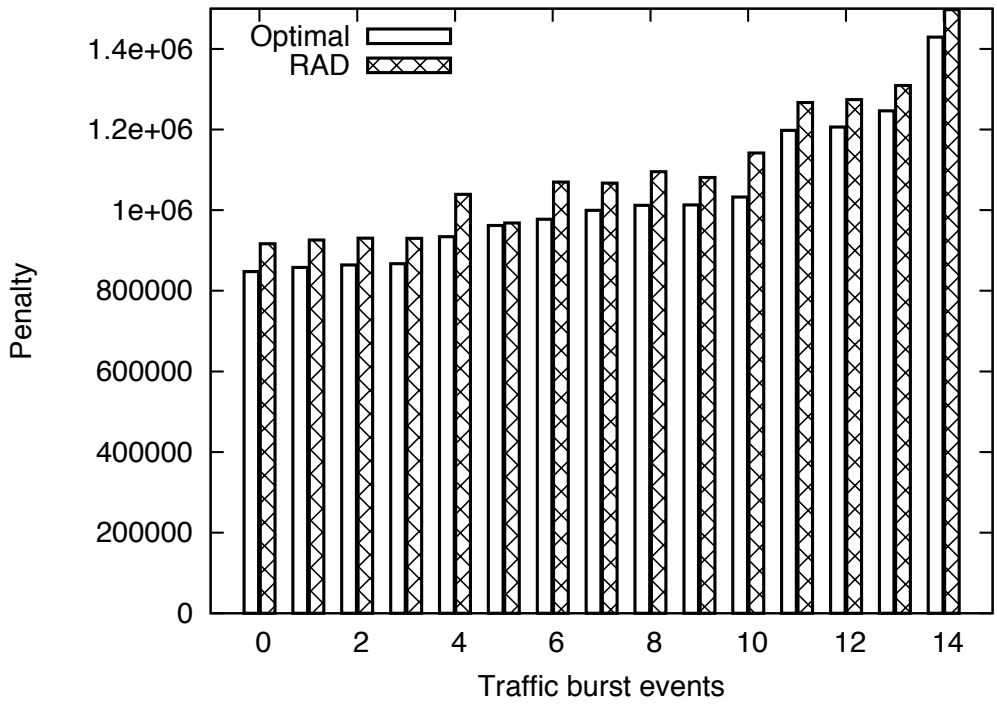

Figure 4.12: Penalty change when traffic bursts: RAD vs optimal sorted by new optimal penalty value.

Chapter 5

Improve RAD

In previous chapter, we have shown that RAD works well on various topologies. It is clear that compared with conventional approaches, RAD has smaller event scope and faster recovery. But can it perform even better?

In this chapter, we focus on how to improve RAD and achieve even better performance and more use cases. First, we analyze the time needed for recovery, and realize the control message exchanges, even only between neighboring routers, may slow down the process. Following the same local reaction principles, we devise a data packet driven recovery mechanism which provides same connectivity guarantee without explicit control message exchanges. It leverage the data packet as implicit signal and therefore can run on the speed of data forwarding.

Another improvement effort to reduce recovery time is to reduce event scope and reactions even further. So the perspective is not performing the actions faster, but invoking fewer actions. We found that although the DAG structure can provide multiple forwarding choices for many routers, due to the loop-free nature however, there is at least one router has only one single outgoing link. This means that if that particular link fails, the router has to go through the reversal process. The solution is to pre-compute backup choices, and the challenge is to keep the forwarding loop-free. By leveraging another locally available information, the ingress port of the data packet, we have managed to keep a loop-free forwarding with more choices.

We then seek more use cases and scenarios that will benefit from RAD. First, the forwarding pattern is extended from unicast only to include anycast, multicast and broadcast. Then we consider feedbacks from network operators about vanilla RAD and better understand their requirements and preferences. More specifically, they want more control over the forwarding in the networks, and also need to provide tiered services and set priorities for different traffic. We extend RAD to support all these new challenges with little to none modifications.

5.1 Faster Reversal

Following our timescale discussion in Chapter 2, even for the same local reaction, if control CPU and software are required to be involved, the process may be much slower than local hardware reaction. Observing that the local repair mechanism of RAD only require local states and information exchanges between neighboring routers, we believe it has the potential to run even faster, with local hardware reactions.

5.1.1 Two Planes

But first, we need to build some background. Networking researchers often refer to two "planes" in networks. The *data plane* forwards packets based on the packet header and local forwarding state in the router (such as a FIB). The *control plane* is responsible for providing that forwarding state. By setting the forwarding state appropriately, the control plane enables networks to achieve connectivity (forwarding tables provide end-to-end paths), route optimization (choosing shortest or otherwise desirable paths), load distribution (links are not overloaded), and other network control goals.

The data plane is typically implemented in hardware and the control plane is typically implemented in software. However, the fundamental distinction between the two planes is not how they are implemented but instead lies in what state they use and what state they change. The data plane only uses forwarding state local to the router in making its decisions, and does not change this state.[1] The control plane typically uses external state — obtained from a distributed algorithm such as a routing protocol — to set the forwarding state. As a result, the two planes operate at very different speeds: for instance, on a 10gbps link, a 1500byte packet is sent in 1.2μsec while exchanging control plane messages takes on the order of 50msec or more (according to vendors we've spoken with) and global convergence of the dataplane can take many seconds.

In the naive version of this two-plane approach, the network can recover from failure (*i.e.,*restore connectivity) only after the control plane has computed a new set of paths and installed the associated state in all routers. The disparity in timescales between packet forwarding and control plane convergence means that failures often lead to unacceptably long outages. To alleviate this, the control plane is now often assigned the task of precomputing failover paths; when a failure occurs, the dataplane utilizes this additional state to guide the forwarding of the packet. This approach works for a given failure scenario as long as an appropriate backup path has been established, but the degree of resilience achievable with a reasonable number of precomputed backup paths is quite limited (though perhaps enough for most network requirements). More recently, researchers have been developing methods for computing multiple paths between each source and destination; the end host chooses an

[1] We consider state that tells the router which of its connected links are up to be local configuration state; it is not derived from either the control plane or data plane, but can be changed by local detection of the link.

alternate path when the primary goes down. This approach suffers from the same limited (but perhaps sufficient) degree of resilience, and also requires a downtime of roughly the round-trip-time in the network, which (if the path has any significant speed-of-light latency) is quite long compared to data plane timescales.

This raises the question of whether there is any way to extend the data plane so that one can achieve ideal connectivity, where by ideal connectivity we mean that packets are delivered as long as the network remains connected. Through some simple counterexamples, we have proven that the answer is no: if the forwarding state remains constant (*i.e.,*the control plane has not yet had a chance to recompute the forwarding state) one cannot always achieve ideal connectivity. We omit the formal statement and proof here, but the intuition is obvious: if you allow no state changes in the router (other than knowing which local links are up), and no rewriting of packet headers (as in [48, 47]), then there is no way that local state could compensate for any arbitrary set of connectivity-preserving failures. Given that this impossibility result precludes achieving ideal connectivity solely using the data plane, we must then ask: can we find a way to achieve ideal connectivity on a timescale much less than that of the control plane?

5.1.2 Data-Driven Connectivity

The timescale of the control plane is so large because it must deal with situations where changes far from a router will have an impact on its state (*e.g.,*it must check remote state to see if the shortest path has changed). However, one can ask if there is a much smaller class of forwarding state changes, ones that depend only on nearby information, that could support ideal connectivity. To this end, we propose the idea of *data-driven connectivity* (DDC), which maintains connectivity by allowing simple changes in forwarding state predicated only on the destination address and incoming port of an incoming packet. The DDC state changes we allow are those that are simple enough to be easily done at packet rates with revised hardware (and, in current routers, can be done quickly in software).

The advantage of the DDC paradigm is that it leaves the general control requirements which require globally distributed algorithms (such as optimizing routes, detecting disconnections, and distributing load) to be handled by the control plane, and moves connectivity maintenance, which has simple semantics, to DDC. DDC has, at worst, a much faster time scale than the control plane, and with new hardware can keep up with the data plane.

In this section we apply this approach to the general problem of intradomain routing (whether it be datacenter, WAN, or enterprise) at either layer 2 or layer 3. The only assumption we make is that all forwarding decisions are made on exact-match lookups (whether it be over MAC addresses or IP addresses/prefixes).[2] In what follows, we will refer to addresses as the unit of exact match and nodes as the forwarding element (whether it be a

[2]This assumption is not essential to the correctness of our algorithm, but it removes the possibility that a change in one address' route causes a much larger rewriting of the forwarding state (as can happen in LPM), which is a slow process.

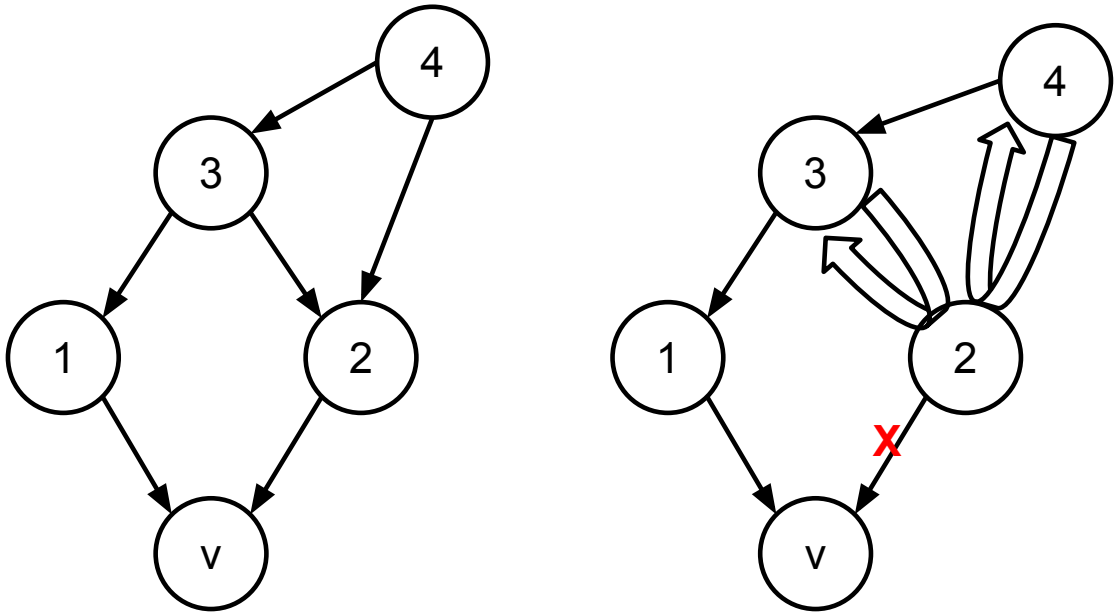

Figure 5.1: Illustration of DDC. (a) normal forwarding. (b) DDC bounce back when failure happens.

switch or a router). Because of the exact match requirement, one can consider the forwarding state for each address independently, so we typically consider only how DDC updates the forwarding state for a single address (because the state change is driven by the arrival of a single packet, and it is that packet's destination address that determines which address' forwarding state will be updated).

DDC maintains connectivity via simple changes to local forwarding state, predicated only on the destination address and incoming port of an incoming packet. We first give a brief overview of how DDC works, followed by a more detailed description and a brief sketch of its correctness.

5.1.3 DDC Example

Consider a network modeled as an undirected graph $G = (N, E)$, where N is the set of nodes, and E is the set of links. In what follows, we only consider the forwarding state associated with a unique destination node v, and packets addressed to that destination. The forwarding state at each node other than v (which is initialized by the control plane) specifies which links it should use to reach v. Directed edges in Figure 5.1(a) illustrate the forwarding state in a simple example, *e.g.,*node 3 forwards packets (destined to node v) to either node 1 or node 2. If node i forwards packets through next-hop node j, we call link (i, j) an *outgoing*

link for i, and an *incoming link* for j.[3] Let I_n and O_n be the sets of incoming and outgoing links (respectively) at node n. When node n's outgoing link fails, that link is immediately removed from n's set of outgoing links O_n. In Figure 5.1(a), $I_2 = \{3, 4\}$ and $O_2 = \{v\}$.

Intuitively, in DDC, a router should send out a received packet along an outgoing link as long as such a link exists, and "bounce back" the packet to the sending neighbor otherwise. To wit, when a packet destined to v arrives at node n, the following two steps are executed:

Update: If the packet arrived on an incoming link, no updates are needed. If it arrived on an outgoing link, remove that link from O_n and place it in I_n.
Forward: If O_n is not empty, forward the packet along one of the available outgoing links. If O_n is empty, forward packet back through the incoming link from which it arrived.

Consider Figure 5.1(b). When node 2 loses its outgoing link $(2, v)$ due to link failure it immediately removes the link from O_2, which then becomes empty. When a packet destined to v arrives at 2 from nodes 3 or 4, it sends it back through the same incoming link ("bouncing it back"). Once node 3 receives a packet from 2, it removes the outgoing link $(3, 2)$ from O_3 and send the packet through its other outgoing link $(3, 1)$. Similarly, once node 4 receives a packet from 2 it removes its outgoing link to 2 from O_4 and send that packet through its other outgoing link.

Note how node 2's "bounce back mechanism" prevents on-the-fly packets from being dropped, and that after 3 and 4 remove $(3, 2)$ and $(4, 2)$ from their outgoing link sets this bounce back is no longer needed. It is also obvious DDC works without global communication. Instead, nodes' actions under DDC are fast and simple, and are based on local information only. However, what if multiple nodes need to bounce back? In order to ensure connectivity, we need to augment this basic algorithm to prevent O_n from remaining empty. We next extend the basic idea to a complete design. This will require a more detailed description; in what follows, we assume all packets are destined for v. [4]

5.1.4 Design Details

Because forwarding state is local, we choose to refer to ports (which are local) rather than links (which are shared between the two endpoints). We define four types of ports: incoming ports (I-ports), outgoing ports (O-ports), reversed incoming ports (RI-ports), and reversed outgoing ports (RO-ports).

[3] For convenience we assume that all links are either incoming or outgoing (as opposed to unused), as it makes our later state transition diagrams easier.

[4] Destinations could be an individual host (as in L2) or a prefix; in ISP networks where IP prefixes are disseminated using a different protocol — such as iBGP — and the IGP is only used to compute routes between the various routers, the destination can be a destination IP prefix or a set of prefixes reachable from one edge router. One could also use an approach similar to TRILL [4] and LISP [18] in which the core routing design provides switch-to-switch delivery, and an auxiliary mapping function determines which switch a host is attached to.

Port State	Possible Packet Actions
Incoming	Receive
Reversed Incoming	Bounce back and send out
Outgoing	Send out and receive
Reversed Outgoing	Receive

Table 5.1: Port state and packet actions on port

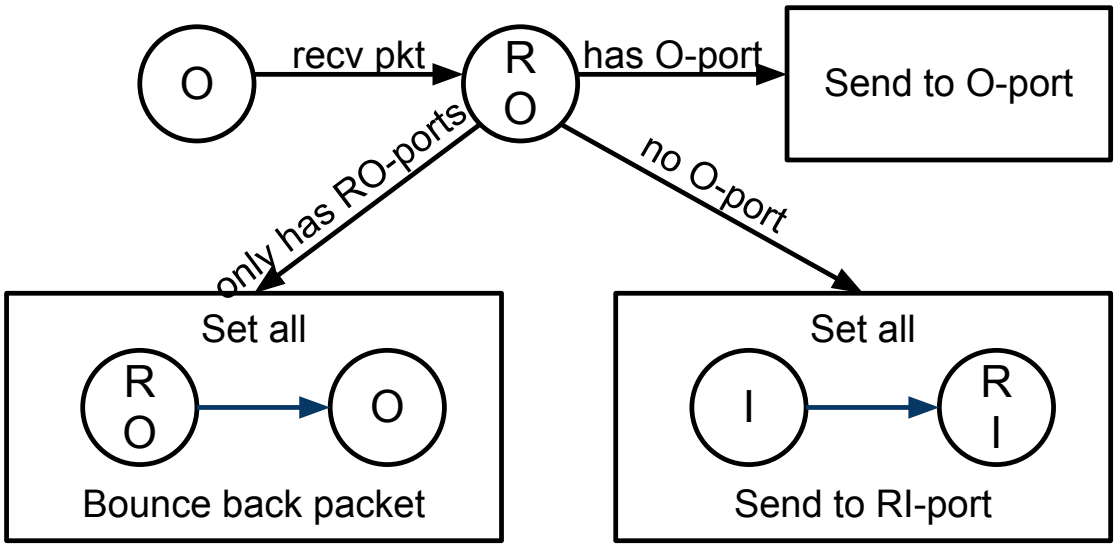

Figure 5.2: State transition when an O-port receives packet.

Initially, all ports are either incoming ports (I-ports), through which packets are received, or outgoing ports (O-ports) through which packets are sent out. In the example described in Figure 5.1, the port connecting node 3 to node 4 is an I-port whereas the port connecting node 3 to node 1 is an O-port. Thus, I-ports and O-ports capture "normal" behavior, and the network is initialized in a state where all ports are in one of these categories.

In the presence of failures, packets in DDC must occasionally be "bounced back"; to do so, the corresponding ports must be "reversed" so that packets can travel in the opposite direction than dictated by their original I-port or O-port designation. To handle such "reversal" situations, we introduce the categories of reversed incoming ports (RI-ports), and reversed outgoing ports (RO-ports). At the DDC level, the following transitions are allowed: I to RI, RI to I, O to RO, and RO to O. The only way an I-port can become an O-port, or vice-versa, is through actions of the control plane (which we discuss in the next section). However, the DDC transitions are sufficient to guarantee connectivity. We now describe

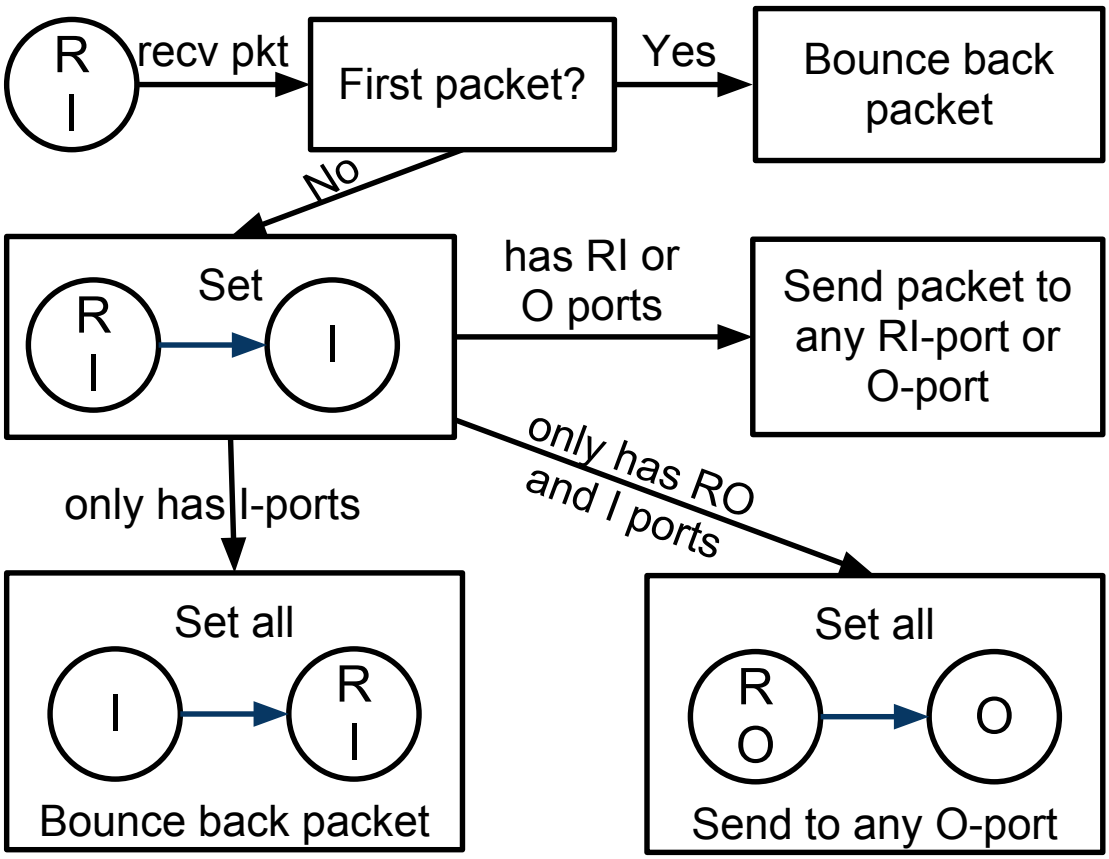

Figure 5.3: State transition when an RI-port receives packet.

these transitions in greater depth.

We define four types of ports: the first two capture "normal" behavior, incoming ports (I-ports), outgoing ports (O-ports); the latter two capture "reversal" situations, reversed incoming ports (RI-ports), and reversed outgoing ports (RO-ports). At the DDC level, the following transitions are allowed: I to RI, RI to I, O to RO, and RO to O. The only way an I-port can become an O-port, or vice-versa, is through actions of the control plane (which we discuss in the next section). However, as we show below, the DDC transitions are sufficient to guarantee connectivity.

The O-RO and I-RI transitions are simple: RI-ports are (originally) incoming ports along which the node "bounces back" an incoming packet, and RO-ports are (originally) outgoing ports along which the node receives an incoming packet. The transitions from RO to O, and RI to I, are more complicated and the associated state-diagrams are shown in Figures 5.2 and 5.3.

To illustrate these scenarios, consider a node i that has a single port which is an RO-port. When a packet arrives along this RO-port, i has no choice but to bounce it back through the same port, with the hope that the receiving neighbor j is able to forward the packet towards the destination. According to our rules (see Figure 5.2), i must make its single port an O-port again, hence the term "re-reverse". Observe that as i received the packet from j through an RO-port it must be that i's packet will reach j through j's RI-port. At this point j should, if possible, send to another RI-port or O-port. The transition is illustrated in Figure 5.3. Now, consider the case that j only has RI ports. Intuitively, we wish to avoid the problematic scenario that j bounce the packet back to i, making i's port an RO-port again, driving i to reverse its port and make it an O-port again (see Figure 5.2), and so on.

To achieve this, a timeout described in Figure 5.3 is introduced to detect re-reverse at node j. When node j receives a packet from i through an RI-port, it will set an internal timeout, *e.g.*, 0.1 seconds, and bounce the packet back to i. When the same packet reaches j again from i after the timeout has expired, j will turn that RI-port into I-port and will forward the packet through a different RI-port (see Figure 5.3). To avoid false positives in the detection of re-reverse, the timeout should be at least $2\times \mathrm{D}_{link}+\mathrm{D}_{DDC}$, where D_{link} is the link delay and D_{DDC} is the time it takes a node to implement the DDC rules upon the receipt of a packet. The timeout should also not be much bidder than this value so as to avoid unnecessary bounce backs when re-reverse happens.

Because at this time, the node on the other end of the link, call it m, is not aware of any difference and may still keep sending. m will not change any state until it receives packet from its O-port. Therefore all on-the-fly packets will be bounced back and sent to another O-port of m if possible, and after that n will not receive packet directly from m. But if m has to re-reverse, n will receive packet after the timeout, and realize m has no path to deliver the packet. So n will accept the re-reverse by setting port state from RI to I, and then try to send to other ports that can be used to send out packets.

Every singly connected node will go through the process should it receive packet on its only O-port. In practice, it is easy to avoid such inefficiency in the first place by suppressing reversal if the other end of the link has degree 1. However, it is also possible that a richly connected node has no I-port, then the re-reverse is necessary to restore its connectivity if all its O-ports become RO-ports.

5.1.5 Ideal Connectivity

DDC achieves ideal connectivity, in the sense that so long as a set of network failures does not disconnect a node n from the destination v, packets from n are delivered to v. More specifically, we can prove the following:

Theorem 5 *Let $\bar{G}$ be a network graph obtained from G via the removal of a subset of the links in E. For every node n that has some route to the destination v in $\bar{G}$ it holds that under DDC every packet sent from i to v is guaranteed to reach v.*

Observe that the fact that packets are never dropped follows immediately from our DDC forwarding rules (a packet is always either bounced back or forwarded through another port). To establish Theorem 5, we are left with proving that packets never enter endless loops, and thus always reach v eventually. Our proof of this claim is similar in spirit to the correctness proof for the algorithm in [21], and the key idea is showing inductively that the claim holds for a gradually expanding set of nodes in v's connectivity component in $\bar{G}$.

DDC may seem too simple to be able to provide ideal connectivity and going through all corner cases to show it works is apparently not feasible. Here we present an induction inspired proof for DDC and ideal connectivity.

Proof:

Because every link in G is used in one direction to reach d, those links with direction form a Directed Acyclic Graph (DAG). Let G be the network graph and let D be the DAG prior to failures. Now, consider the graph G' obtained from G after removing failed links. Our aim is to prove that every node that has some route to the unique destination d in G' will be connected to d after the DDC process. The proof iteratively constructs a set of "good nodes"; a good node is a node that, from some point in time, has a forwarding path to the destination d. We show that the set of good nodes eventually contains all nodes that have some route to d in G', thus establishing our claim.

Initialize the set of good nodes to contain d, and all nodes that are directly connected to d in D. Observe that every node that is 1-hop away from d in G' was also directly connected to d in D and so that node will never lose connectivity to d. Now, consider a node i that is connected to a good node. We handle 3 cases:

1) Node i is connected to a good node in D. Observe that in this case i will never disconnect from d (as the good node that it is connected to will never disconnect from d). Add i to the set of good nodes.

2) Node i is not connected to a good node in D and reverses all of its incoming links at some point in the process. As D utlizes every edge in G (in one direction), if i is not connected to a neighboring good node that good node is connected to i. So, when i reverses all incoming links it will connect to the neighboring good node and remain connected thereafter. Add i to the set of good nodes.

3) Node i is not connected to a good node in D and, at no point in time reverses all of its incoming links. Observe that in this case, i must be connected to a node that also never reverses its incoming links. That node, in turn, is also connected to such a node, etc. Following this sequence of nodes we either reach d (in which case we're done), or end up with a loop—a contradiction to the fact that D is acyclic! Add i and all other nodes in this sequence of nodes to the set of good nodes.

Via this process we are gradually placing all nodes in d's connectivity component in G' in the set of good nodes. The theorem holds.

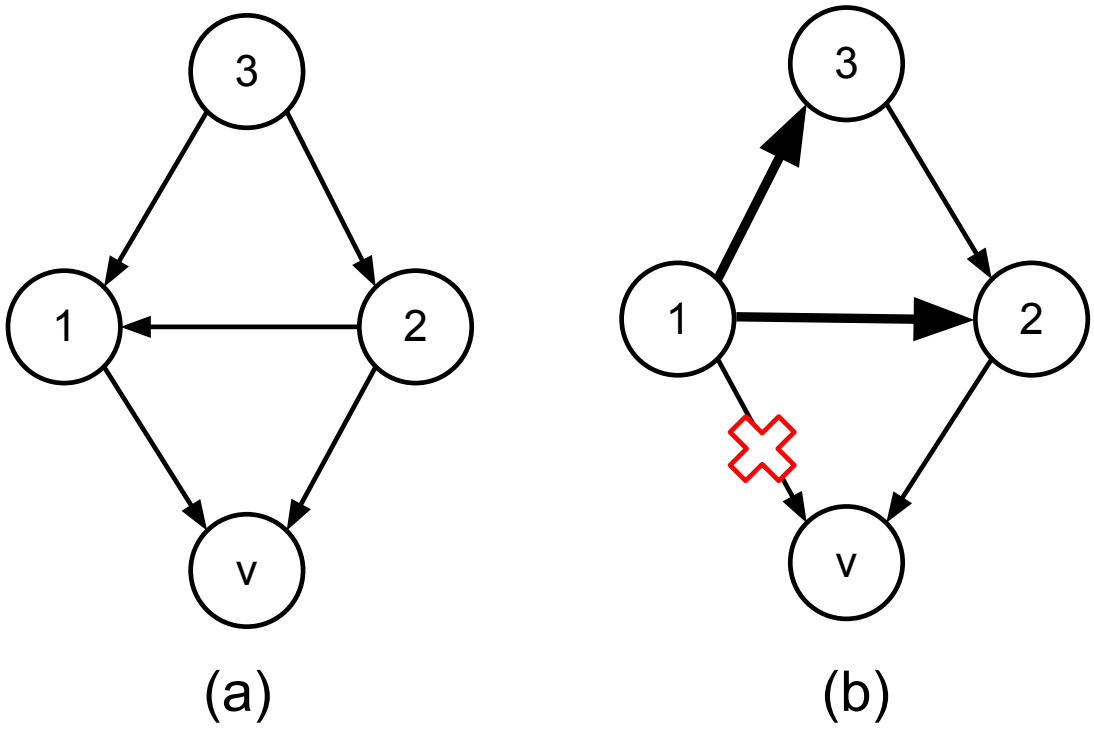

Figure 5.4: Node 1 reverses incoming links when link $1 \rightarrow v$ fails.

5.2 Fewer Reversal

The local repair process will automatically stop as soon as every router has forwarding choices. But it is still desirable to further reduce the number of reversals, or avoid reversal completely. In this section, we discuss possibilities of how connectivity can be provided with fewer reversals.

We first review the single link vulnerability of DAG, which is unfortunately an inherent property and can not be removed. Then we illustrate how adding extra forwarding choices can avoid link reversal through an example. We generalize the approach and propose a new network model to describe the new kind of graphs. We also provide one algorithm to compute the needed choices.

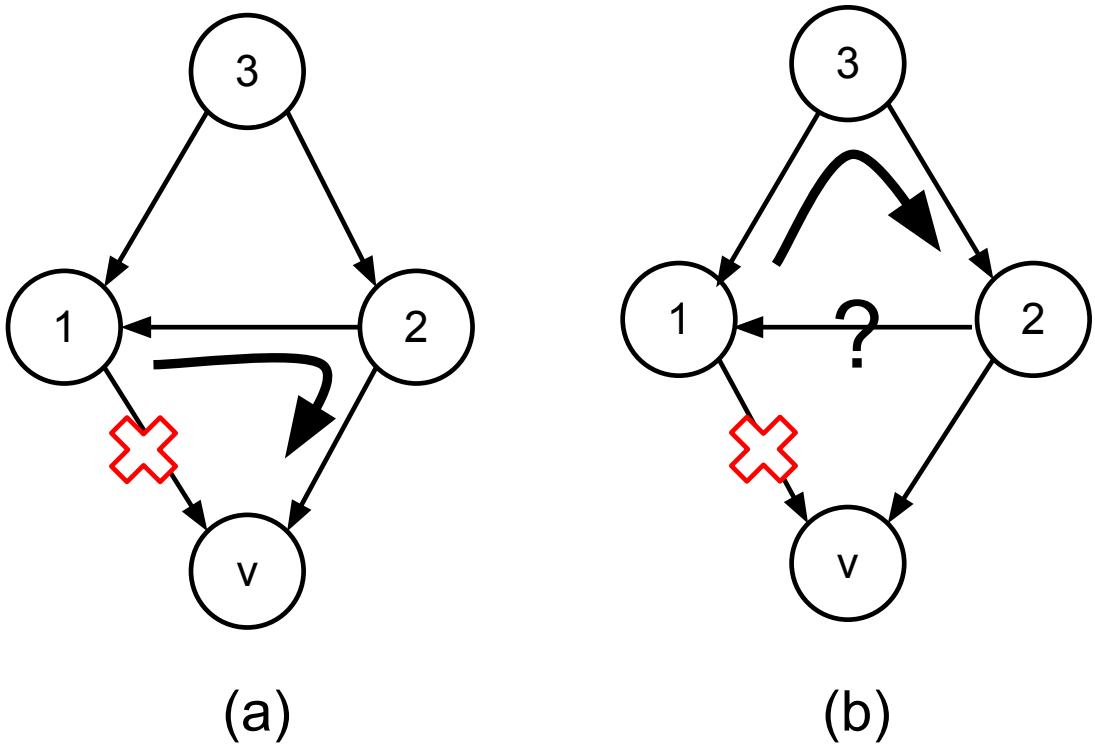

Figure 5.5: With added choice, no need to reverse link 3 → 1. But the choice should be carefully computed to avoid forwarding loops.

5.2.1 Single Outgoing Node

Although many nodes in the DAG have multiple outgoing links, at least one node has only a single choice. Such a node is vulnerable to a single link failure (as noted in [45]).[5]

Recall that the destination node in the DAG is the only node that has no outgoing links. If we remove the destination node and all its incoming links from the DAG graph G, the resulting graph D is a subgraph of the original graph, so D is also a DAG. For any DAG, it is well known that there must be a node that has no outgoing links. Denote it o in DAG D. Since o is not the destination node, it must have at least one outgoing link in original graph G. Because after removing destination node and adjacent links o has no outgoing links, we know that o is a neighbor node of the destination, and o only has one outgoing link in G.

The same conclusion can also be drawn from the perspective of node ordering. Because a DAG corresponds to an ordering among nodes, and the link direction follows the ordering,

[5]It is obvious that if the node is also singly connected in network topology, i.e. only has one neighbor, there is little to improve.

the node that is closest to destination will only have one outgoing link towards destination, and its other links are all incoming links.

5.2.2 DAG with Backups

One can compute backup paths (using an additional routing table that determines where to send packets that come in along the reverse direction of a link, somewhat as in [42]) such that every node has at least two outgoing links, and there are no cycles (as long as these backup paths are not used when the normal links are available).

By adding backup paths to DAG, we can avoid certain reversals and make failure reactions even faster. This is different from previously discussed DDC, in that it just enhances existing DAG structure, and still requires RAD reversals for the connectivity guarantee.

Figure 5.4 and 5.5 illustrate how DAG with backups can avoid reversals. With a DAG computed as Figure 5.4(a), when link $1 \rightarrow v$ fails, node 1 will reverse its two incoming links: $3 \rightarrow 1$ and $2 \rightarrow 1$. Then node 2 and 3 will react accordingly.

But if we add one backup choice for node 1, indicating that if link $1 \rightarrow v$ is not available, send packets to node 2, there is no need for reversal. Forwarding rules on node 2 is as follows:

- If packet comes from node 1, send to v directly
- Send packet to v or node 1

Note it is possible that node 2 sends packet to 1 and then receives it back, as discussed in chapter 2, we do not consider it a forwarding loop, because after node 2 receives the packet, it will send to v directly.

However, if the backup choice is link $1 \rightarrow 3$, there will be a problem. Because after the packet is received by node 2, it has no idea node 1 can not reach v directly anymore, and may forward packet to 1. Then a loop is formed by traversing nodes 1-3-2-1.

5.2.3 Compute Backups

Below we describe a general algorithm to compute backups for singly outgoing nodes in the DAG.

Let $G = (V, E)$ be an undirected graph without self loops in which the set of vertices $V(G)$ consists of n *source nodes* $\{1, 2, \ldots, n\}$ and a *destination node* $d \notin \{1, 2, \ldots, n\}$. Each source node i has a *forwarding function* f_i^d. Let $N(i) = \{i\} \cup \{j \text{ such that } (i, j) \in E\}$. That is, $N(i)$ contains all neighbors of i in G and i itself. The domain of the forwarding function f_i^d is $N(i)$, and the range of f_i^d is $2^{N(i)-\{i\}}$. The value $f_i^d(j)$ is the set of neighbors of i to which i can forward a packet destined to d if i receives that packet from its neighbor j (the case in which $j \neq i$) or if the packet originates at i (the case in which $j = i$). We call an n-tuple $f^d = (f_1^d, f_2^d, \ldots, f_n^d)$ of forwarding functions a *forwarding pattern*.

We want the packet-forwarding scheme that source nodes use to route traffic to d to be *loop-free*, that is, we want to avoid forwarding loops in which packets traverse the same links over and over again. We also want the packet-forwarding scheme to be such that all packets eventually reach the destination node d. Consider a forwarding pattern f^d.

1. **Initialization.** $\forall (i,j) \in E$, $f_i^d(j) := \emptyset$.
2. **DAG construction.** Construct a DAG D (*e.g.*, using BFS/DFS) that is rooted in d and such that $\forall (u,v) \in E(G)$, $(u,v) \in E(D)$ or $(v,u) \in E(D)$. Observe that 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. **DAG-based forwarding rules.** $\forall i \in V$, $F^D(i) := \{i\} \cup \{j \text{ such that } (i,j) \in D\}$ (that is, F_i^D is the set of all nodes to which i has a directed edge in D, and i itself). $\forall k \in N_i$, $f_i^d(k) := F_i^D$.
4. **Installing additional forwarding rules.** While there exists a node that is 2-connected to d in G, but is not 2-connected to d in $f^d = (f_1^d, \ldots, f_n^d)$, do:
   - Choose i to be a minimal node (under $<_D$) that is 2-connected to d in G, but is not 2-connected to d in $f^d = (f_1^d, \ldots, f_n^d)$.
   - Choose j to be a minimal node (under $<_D$) such that (1) $i <_D j$ and (2) $\exists s \in V$ such that $(j,s) \in D$ and $i_D s$.
   - Choose a simple route from j to i in D, $R = (j = a_1, a_2 \ldots, a_k = i)$.
   - $r := k - 1$.
   - While $(r > 1)$ and $(f_{i_r}^d(i_{r+1}) = \emptyset)$ do:
     - $f_{i_r}^d(i_{r+1}) := \{i_{r-1}\}$
     - $r := r - 1$
   - If $r = 1$, then $f_j^d(a_2) := \{s\}$.

5.3 Different Communication Patterns

The current set of RAD proposals only consider unicast delivery. RAD can be simply extended to provide highly resilient anycast (the same way unicast routing automatically supports anycast). Broadcast can be achieved by following the reversed direction of every link in the DAG. The broadcast packet is guaranteed to reach every node as long as the network is connected. Comparing with a tree-based broadcast, there are $E - N + 1$ more broadcast packets sent between nodes, where E, N are the number of links and nodes in the broadcast domain respectively. Multicast can be supported with pruning non-member

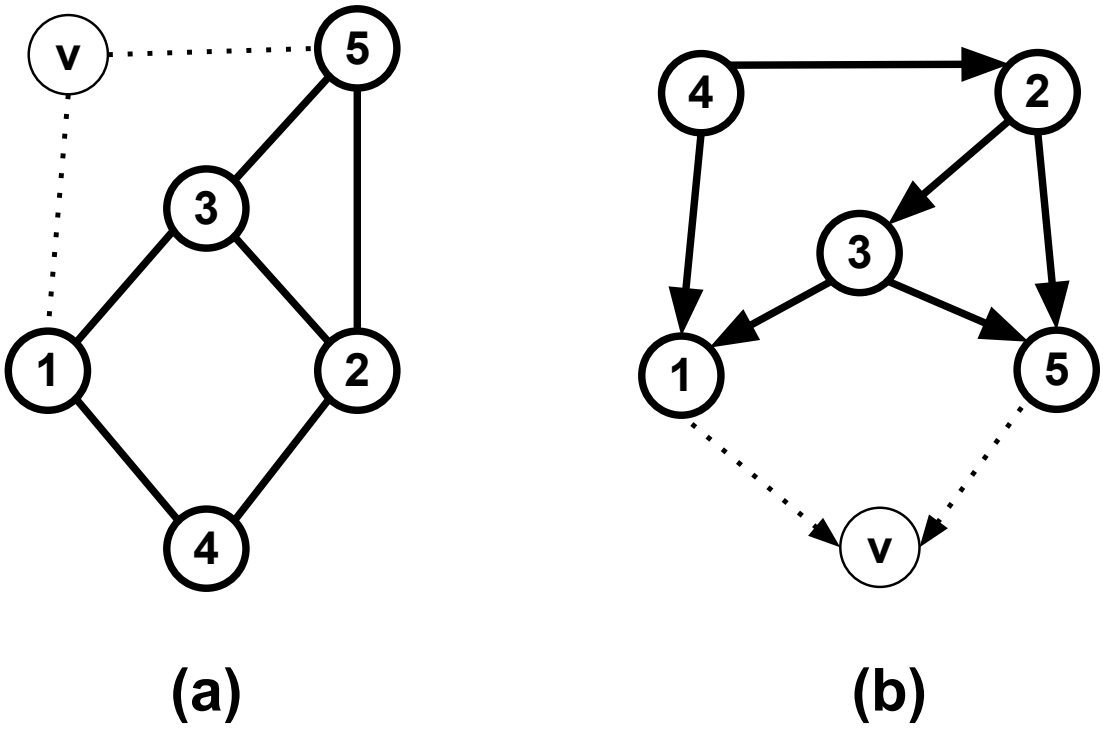

Figure 5.6: Illustration of RAD with anycast. Node 1 and 5 are receivers. (a) Add virtual node v to the graph. (b) Build DAG rooted at v.

routers. But if one builds multicast delivery over unicast paths (as is typically done) then multicast will benefit from the increased reliability of RAD's unicast paths, and there may be no need for RAD to support multicast directly.

5.3.1 Anycast

Anycast defines a group of potential receivers, and considers the packet is successfully delivered when it is received by any one of them. For IP anycast, the destination IP address is usually the same for all receivers. RAD can support anycast without modification. The only difference is how the initial DAG is built. We adopt a common technique in graph research, which is adding virtual node to existing graph and connecting it to several nodes in the graph.

Figure 5.6 shows an example. In the original graph, node 1 and 5 are the receivers of the anycast group. Add virtual node v to the graph, and connects it to node 1 and 5, shown as dashed lines in Figure 5.6(a). Then we build a DAG with node v as destination. The result is shown in Figure 5.6(b). Anycast packets will be forwarded along the directed links,

therefore they will be delivered to either node 1 or node 5.

In general, we will add a virtual node connecting to all receivers in the anycast group, and build a DAG with the virtual node as destination. This way anycast can benefit from many nice features of RAD, and have resilient forwarding.

5.3.2 Broadcast

First of all, broadcast is never a scalable communication pattern. Flooding message or packet to every router in the network is both expensive and inefficient. But we do not want to mandate how people are using the network, because in certain circumstances, broadcast may be an acceptable solution. Just like currently broadcast packets are sent in the reversed direction of the forwarding tree, RAD can support broadcast packets by sending them in the reversed direction of DAG links, i.e. receive from an outgoing link and send to incoming links. If resiliency is preferred, broadcast packets can be sent to all incoming links, at the cost of many duplicates. Or packets can also be only sent along shortest paths in the DAG, so it will generate fewer duplicates but may have packet loss if link fails during forwarding.

5.3.3 Multicast

The support for multicast is similar to broadcast, with pruning non-member routers to improve efficiency. We can leverage existing group management protocols like Protocol-Independent Multicast Dense Mode (PIM-DM) and Sparse Mode (PIM-SM) and remove non-member routers from the DAG. So it will provide a "slim" DAG which has better resiliency than single multicast tree.

5.4 Thin DAGs

RAD is designed to deliver the packet whenever it is possible. In practice, however, network operators may prefer more control over packet forwarding and better visibility of router updates. But RAD by itself does not support the trade-off between connectivity and controllability.

Another challenge is source control. A recent trend in routing research is to give sources (or edge routers) control over routing. RAD handles link failure and congestion inside the network, so it flies in the face of this recent trend towards source control.

The extension to above questions is what we call "thin" DAGs, DAGs that are defined around a given default path that protect against any individual failed link. Instead of building a DAG that uses all available links, we can build multiple thinner DAGs and switch between them. This is essentially the idea of multi-path routing, with path replaced by DAG.

For example, MPLS currently has Label Switching Path (LSP) which dictates an end-to-end path. With RAD and thin-DAG, the LSP can become Label Switching DAG (LSD),

which handles link failure and congestion to a certain degree, then network operators can switch to a different LSD for better resiliency or load distribution.

Source control can also be supported by adding the chosen DAG identifier into data packets. So the source can decide which DAG to use, and change it when necessary. Our preliminary efforts are presented in paper Slick Packets [54].

Chapter 6

Conclusion

In this thesis, we presented a new routing framework: Routing Along DAGs. We start from the challenges of today's networks and first understand how they impact routing, the fundamental function of any network. High availability and scalability are identified as the two key requirements for routing. Although they are not completely new, we realize conventional approaches have difficulties to meet all the stringent requirements today and tomorrow. Instead of tweaking existing methods or making incremental improvements, we analyze conventional routing mechanisms and argue that they have inherent problems that limit their performance in large and complicated networks. More importantly, the assumptions about network, and trade-offs of their designs, all require a review and should be changed appropriately to reflect what we have today and tomorrow.

The revised assumptions and design trade-offs lead to RAD, a complete and coherent routing framework, which can meet even more strict scalability and availability requirements. The most notable design philosophies of RAD include: separate optimization and connectivity, build explicit loop-free topology and prefer fast and local recovery. We test RAD on various ISP and data center topologies, and show that its performance is consistently better than traditional methods.

Then we recognize several aspects of RAD that have more potential and continue improving its performance, functionality and applicable scenarios. All these improvements follow the same design principles and extend the RAD framework. Based on feedbacks from network operators, we also acknowledge that compared to guaranteed connectivity, which is more appealing in theory, more controllable and visible recovery choices are preferred in many practical cases. RAD is flexible to adapt to these new challenges and provide good performance with little modification.

Overall, we also showed the evolution of the design of RAD:

- Start from reviewing shortcomings of conventional mechanisms and current improvements
- Analyze bottlenecks, discover and understand the root causes

- Propose new designs based on revised assumptions and trade-offs
- Review the interoperability with existing mechanisms
- Iterate the design with improvements and extensions, all based on feedbacks in practice

6.1 Future Directions

We believe RAD is just a first step into the new routing paradigm, and what we have done and presented is only a small fraction of the design space. There are more to explore, and many interesting questions to answer. We roughly categorize future directions into two groups: theory and practice. Questions and directions in the theory group focus on the abstract network model, and are related to graph properties, like connectivity and resilience. The practice group includes more practical concerns, which consider actual hardware platform, deployment and operation, and are more related to implementation.

6.1.1 Theory Questions

We have shown in the thesis that precomputed DAG combined with local repair mechanism can achieve ideal connectivity, i.e. routing and forwarding topology are connected as long as underlying network topology is connected. Then seeing the limitation of DAG, we introduce a method to leverage incoming port of the packet, which is also a local information available to the router, to improve failure recovery performance. These experiences lead us to believe that local reaction has more possibilities to be studied.

- Ingress port in forwarding model. The ingress port of the received packet has been readily available since the beginning of packet switching, but is unfortunately neglected in almost all forwarding model abstractions. Common enhancements to vanilla destination based forwarding usually employ proprietary packet labeling schemes, which will introduce overhead in forwarding hardware, complicate packet header and processing, and generally imply higher barrier and cost for adoption. The ingress port, on the other hand, is a local information available to the router, agnostic to layers and packet formats, and easy to implement. Then the question is how much extra hint the ingress port can carry. It will be interesting to see new designs that consider network topology characteristics and packet forwarding together, and leverage the ingress port information to learn what has changed in the network.
- A list or set of egress ports. Now we consider the choices for forwarding. One abstraction can be a list of candidate egress ports, and the decision is simply to pick the first available one. However, more sophisticated schemes can be developed following the concepts of choices and decision, i.e. routing topology and forwarding topology. The

strategy of picking which egress port to use may also incorporate more intelligence. The egress ports can also be optimized for different criteria, such as latency, throughput etc. How to maintain the set of egress ports to keep loop-free forwarding and improve failure resilience and network capacity, is another topic worth discovering.

Another appealing feature of RAD is local and dynamic updates driven by control messages or data packets, which can be categorized as a local repairing mechanism without global synchronization and computation. Local and dynamic routing changes definitely has more potential in improving failure resilience and routing performance. We list several interesting questions below:

- Static routing. Instead of dynamic route updates, a completely static set of routing choices is appealing to network operators in many ways, including forwarding control, debugging, visibility etc. Unfortunately we have demoed that static routing without packet labeling can not guarantee loop-free and connectivity even on some simple topologies. But the resilience performance may be really close to the optimal, so in practice, operators may be willing to accept the slightly less than ideal connectivity to get more confidence in the control of packet forwarding. There may also be a class of topologies which supports static routing very well. It would be possible to derive a new metric or property for network topologies to describe how well static routing can perform. The practical impact would be an additional guidance to network design in scenarios like data center networks, where the topologies are carefully designed and controlled by operators.

- K-failure resilience in N-connected graphs. It is not fully exploited how properties of network topology, i.e. the undirected graph, affects failure resilience performance. Challenging questions include: given a graph that is N-connected, how a routing scheme provides K-failure (concurrent removal of K links) resilience, what would be the relationship between K and N, and what extra information or mechanisms would be needed to improve K. Although we did some initial study, the full extent of these questions require more theoretical thoughts and would attract interests from researchers in other fields.

6.1.2 Practical Questions

Throughout the discussion of RAD and its extensions, we consider the router will behave as one consistent unit, e.g. all the actions will be performed as atomic. This may pose a challenge for modern distributed router architecture.

- Accurate hardware model. A typical component layout of high end routers is shown in Figure 6.1. The router usually consists of multiple line cards and a backplane, through which line cards communicate with each other. The line cards may be configured

differently. A data forwarding card will contain switching fabric chip and companion processor chips for state updates, while a control card may only have powerful processors to compute routes etc. When a control card needs to update forwarding states on other line cards, it will send commands through the backplane. Therefore the router itself is a distributed communication system.

- Distributed system and synchronization. Based on our discussions with hardware vendors, the signal latency on backplane is very small, so most of the time line cards are considered to be in consistent states. But because the backplane is designed to be high throughput and low latency, the mechanism is relatively simple, and any change to it may incur formidable costs. On the other hand, since each forwarding line card keeps its own copy of the forwarding table, it is possible to leverage the distributed nature to solve other challenges like scalability.

- Minimal requirement. Another unanswered question in this thesis is how RAD will perform if the consistency on one router is not strict. In other words, if the atomic requirement is loosened, will RAD still keep the loop-free and many other nice features? If not, to what extent or in which particular cases that RAD and its various extensions will have problems? We believe it will encourage research efforts that better track today's hardware progress, and may potentially change the graph model for network research completely.

- More software, more control. Even the switching fabric, which is definitely considered hardware in the network community, has code running on top of it to control different behaviors and state transitions. Given that most of RAD operations only need locally available information or states, the implementation of RAD may benefit from the advances in IC technology and does not require ASIC changes, well-known to be both time consuming and capital expensive. A semi-software implementation also enables faster iteration of improvements and better interacts with other components.

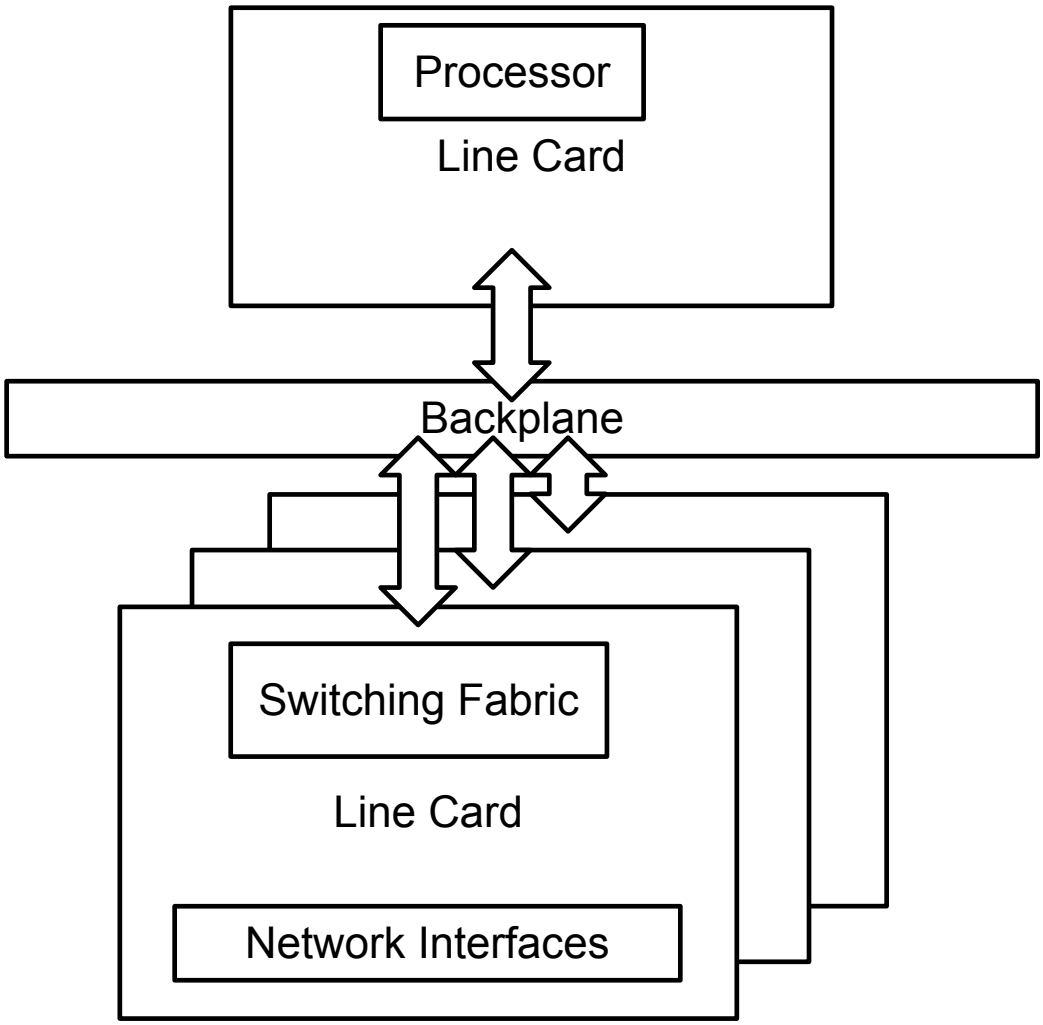

Figure 6.1: Architecture of distributed router architecture.

Bibliography

[1] Extending Networking into the Virtualization Layer. http://openvswitch.org/.

[2] IEEE 802.1W Rapid Reconfiguration of Spanning Tree. http://www.ieee802.org/1/pages/802.1w.html.

[3] Open Cirrus. http://opencirrus.org/.

[4] Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement (RFC 5556). http://www.ietf.org/rfc/rfc5556.txt.

[5] *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, 19-25 April 2009, Rio de Janeiro, Brazil*. IEEE, 2009.

[6] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[7] R. Albrightson, JJ Garcia-Luna-Aceves, and J. Boyle. EIGRP-a fast routing protocol based on distance vectors. In *Proc. Networld/Interop*, volume 94, 1994.

[8] Ashok Anand, Archit Gupta, Aditya Akella, Srinivasan Seshan, and Scott Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *SIGCOMM*, 2008.

[9] David G. Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable internet protocol (AIP). In *SIGCOMM*, 2008.

[10] David Applegate and Edith Cohen. Making intra-domain routing robust to changing and uncertain traffic demands: understanding fundamental tradeoffs. In *SIGCOMM*, 2003.

[11] M. Arregoces and M. Portolani. *Data center fundamentals*. Cisco Press, 2003.

[12] C. Busch, S. Surapaneni, and S. Tirthapura. Analysis of link reversal routing algorithms for mobile ad hoc networks. In *SPAA*, 2003.

[13] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van der Merwe. Design and implementation of a routing control platform. In *NSDI '05*, 2005.

[14] T. Cicic, AF Hansen, A. Kvalbein, M. Hartmann, R. Martin, M. Menth, S. Gjessing, and O. Lysne. Relaxed multiple routing configurations: IP fast reroute for single and correlated failures. *Network and Service Management, IEEE Transactions on*, 6(1):1–14, 2009.

[15] M. Scott Corson and Anthony Ephremides. A distributed routing algorithm for mobile wireless networks. *Wireless Networks*, 1(1):61–81, 1995.

[16] Jonathan D. Ellithorpe, Zhangxi Tan, and Randy H. Katz. Internet-in-a-box: emulating datacenter network architectures using fpgas. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, 2009.

[17] Andrey Ermolinskiy and Scott Shenker. Reducing Transient Disconnectivity using Anomaly-Cognizant Forwarding. In *HotNets*, 2008.

[18] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis. Locator/ID Separation Protocol (LISP). *IETF Draft*, 2009.

[19] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29(1):13–48, 2004.

[20] Pierre François and Olivier Bonaventure. Avoiding transient loops during the convergence of link-state routing protocols. *IEEE/ACM Trans. Netw.*, 15(6):1280–1292, 2007.

[21] Eli M. Gafni, Dimitri, and P. Bertsekas. Distributed algorithms for generating loop-free routes in networks with frequently changing topology. *IEEE Transactions on Communications*, 1981.

[22] Igor Ganichev, Dai BIn, Philip Brighten Godfrey, and Scott Shenker. YAMR: Yet Another Multipath Routing Protocol. http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-150.pdf.

[23] J.J. Garcia-Luna-Aceves and W.T. Zaumen. Area-based, loop-free internet routing. In *INFOCOM'94. Networking for Global Communications., 13th Proceedings IEEE*, pages 1000–1008. IEEE, 2002.

[24] JJ Garcia-Lunes-Aceves. Loop-free routing using diffusing computations. *IEEE/ACM Transactions on Networking (TON)*, 1(1):130–141, 1993.

[25] Brighten Godfrey, Igor Ganichev, Scott Shenker, and Ion Stoica. Pathlet routing. In *SIGCOMM*, 2009.

[26] P. B. Godfrey, S. Shenker, and I. Stoica. Pathlet Routing. In *HotNets*, 2008.

[27] P. Brighten Godfrey, Matthew Caesar, Ian Haken, Scott Shenker, and Ion Stoica. Stable Internet Route Selection. http://www.cs.uiuc.edu/homes/pbg/papers/srs-nanog40.pdf.

[28] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, 2009.

[29] A. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, 2009.

[30] A. Greenberg, P. Lahiri, D.A. Maltz, P. Patel, and S. Sengupta. Towards a next generation data center architecture: Scalability and commoditization. In *PRESTO*, 2008.

[31] Albert Greenberg, Gisli Hjalmtysson, David A. Maltz, Andy Myers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. A Clean Slate 4D Approach to Network Control and Management. In *ACM SIGCOMM Computer Communication Review*, 2005.

[32] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: towards an operating system for networks. *CCR*, 38(3), 2008.

[33] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. In *SIGCOMM*, 2009.

[34] Jiayue He, Martin Suchara, Ma'ayan Bresler, Jennifer Rexford, and Mung Chiang. Rethinking Internet traffic management: From multiple decompositions to a practical protocol. In *Proc. ACM SIGCOMM CoNext Conference*, December 2007.

[35] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. *RFC 2992*.

[36] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang. Cooperative content distribution and traffic engineering in an ISP network. In *SIGMETRICS*, 2009.

[37] John P. John, Ethan Katz-Bassett, Arvind Krishnamurthy, Thomas Anderson, and Arun Venkataramani. Consensus routing: the internet as a distributed system. In *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, NSDI'08, pages 351–364, Berkeley, CA, USA, 2008. USENIX Association.

[38] S. Kandula, D. Katabi, B. Davie, and A. Charney. TeXCP: Responsive yet stable traffic engineering. In *SIGCOMM*, 2005.

[39] Atul Khanna and John Zinky. The Revised ARPANET Routing Metric. In *SIGCOMM*, pages 45–56, 1989.

[40] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *SIGCOMM*, New York, NY, USA, 2008.

[41] S. Kini, S. Ramasubramanian, A. Kvalbein, and A.F. Hansen. Fast recovery from dual link failures in IP networks. In *INFOCOM 2009, IEEE*, pages 1368–1376. IEEE, 2009.

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

[43] A. Kvalbein, C. Dovrolis, and C. Muthu. Multipath load-adaptive routing: putting the emphasis on robustness and simplicity. In *Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on*, pages 203–212. IEEE, 2009.

[44] A. Kvalbein, AF Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP network recovery using multiple routing configurations. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–11. IEEE, 2007.

[45] K.W. Kwong, L. Gao, R. Guérin, and Z.L. Zhang. On the feasibility and efficacy of protection routing in IP networks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

[46] K.W. Kwong, R. Guérin, A. Shaikh, and S. Tao. Balancing performance, robustness and flexibility in routing systems. *Network and Service Management, IEEE Transactions on*, 7(3):186–199, 2010.

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

[48] S.S. Lor, R. Landa, and M. Rio. Packet re-cycling: eliminating packet losses due to network failures. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, page 2. ACM, 2010.

[49] M. Menth and R. Martin. Network resilience through multi-topology routing. In *The 5th International Workshop on Design of Reliable Communication Networks*, pages 271–277.

[50] Murtaza Motiwala, Megan Elmore, Nick Feamster, and Santosh Vempala. Path splicing. In *SIGCOMM*, 2008.

[51] Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, and Jeffrey C. Mogul. SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies. In *Proc. Networked Systems Design and Implementation*, April 2010.

[52] A. Myers, E. Ng, and H. Zhang. Rethinking the service model: Scaling Ethernet to a million nodes. In *HotNets*, 2004.

[53] R.N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *SIGCOMM*, 2009.

[54] G.T.K. Nguyen, R. Agarwal, J. Liu, M. Caesar, P.B. Godfrey, and S. Shenker. Slick packets. In *Proc. SIGMETRICS*, 2011.

[55] Yasuhiro Ohara, Shinji Imahori, and Rodney Van Meter. Mara: Maximum alternative routing algorithm. In *INFOCOM* [5], pages 298–306.

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

[57] V.D. Park and M.S. Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *INFOCOM*, 1997.

[58] C.E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *CCR*, 24(4), 1994.

[59] P. Psenak, S. Mirtorabi, A. Roy, and L. Nguyen. P. Pillay-Esnault," Multi-Topology (MT) Routing in OSPF (RFC 4915). http://www.ietf.org/rfc/rfc4915.txt.

[60] S. Ray, R. Guérin, K.W. Kwong, and R. Sofia. Always acyclic distributed path computation. *IEEE/ACM Transactions on Networking (ToN)*, 18(1):307–319, 2010.

[61] C. Reichert, Y. Glickmann, and T. Magedanz. Two routing algorithms for failure protection in IP networks. In *Computers and Communications, 2005. ISCC 2005. Proceedings. 10th IEEE Symposium on*, pages 97–102. IEEE, 2005.

[62] C. Reichert and T. Magedanz. Topology requirements for resilient IP networks. In *MMB & PGTS 2004: 12th GI/ITG Conference on Measuring, Modelling, and Evaluation of Computer and Communication Systems (MMB) together with 3rd Polish-German Teletraffic Symposium (PGTS), September 12-15, 2004, Dresden, Germany*, page 379. Margret Schneider, 2004.

[63] M. Shand and S. Bryant. IP Fast Reroute Framework. *IETF Draft*, 2007.

[64] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh. Viking: a multi-spanning-tree Ethernet architecture for metropolitan area and cluster networks. In *INFOCOM*, 2004.

[65] Martin Suchara, Dahai Xu, Robert Doverspike, David Johnson, and Jennifer Rexford. Simple failure resilient load balancing. preprint, 2009.

[66] Martin Suchara, Dahai Xu, Robert Doverspike, David Johnson, and Jennifer Rexford. Network architecture for joint failure recovery and traffic engineering. In *Proc. ACM SIGMETRICS*, June 2011.

[67] Miia Vainio. Link reversal routing. preprint.

[68] S. Vutukury and J.J. Garcia-Luna-Aceves. MDVA: A distance-vector multipath routing protocol. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 1, pages 557–564. IEEE, 2002.

[69] F. Wang and L. Gao. On inferring and characterizing internet routing policies. In *IMC*, 2003.

[70] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end internet path performance. In *SIGCOMM*, 2006.

[71] Feng Wang, Zhuoqing Morley Mao, Jia Wang, Lixin Gao, and Randy Bush. A measurement study on the impact of routing events on end-to-end internet path performance. In *SIGCOMM*, 2006.

[72] Hao Wang, Haiyong Xie 0002, Lili Qiu, Yang Richard Yang, Yin Zhang, and Albert G. Greenberg. Cope: traffic engineering in dynamic networks. In *SIGCOMM*, 2006.

[73] Wen Xu and Jennifer Rexford. MIRO: Multi-path Interdomain ROuting. In *SIGCOMM*, 2006.

[74] Xiaowei Yang and David Wetherall. Source selectable path diversity via routing deflections. In *SIGCOMM*, 2006.