Network Architecture for Joint Failure Recovery and Traffic Engineering

Martin Suchara
Dept. of Computer Science
Princeton University, NJ 08544
msuchara@princeton.edu

Dahai Xu
AT&T Labs Research
Florham Park, NJ 07932
dahaixu@research.att.com

Robert Doverspike
AT&T Labs Research
Florham Park, NJ 07932
rdd@research.att.com

David Johnson
AT&T Labs Research
Florham Park, NJ 07932
dsj@research.att.com

Jennifer Rexford
Dept. of Computer Science
Princeton University, NJ 08544
jrex@princeton.edu

ABSTRACT

Today's networks typically handle *traffic engineering* (e.g., tuning the routing-protocol parameters to optimize the flow of traffic) and *failure recovery* (e.g., pre-installed backup paths) independently. In this paper, we propose a unified way to balance load efficiently under a wide range of failure scenarios. Our architecture supports flexible splitting of traffic over multiple precomputed paths, with efficient path-level failure detection and automatic load balancing over the remaining paths. We propose two candidate solutions that differ in how the routers rebalance the load after a failure, leading to a trade-off between router complexity and load-balancing performance. We present and solve the optimization problems that compute the configuration state for each router. Our experiments with traffic measurements and topology data (including shared risks in the underlying transport network) from a large ISP identify a "sweet spot" that achieves near-optimal load balancing under a variety of failure scenarios, with a relatively small amount of state in the routers. We believe that our solution for joint traffic engineering and failure recovery will appeal to Internet Service Providers as well as the operators of data-center networks.

Categories and Subject Descriptors

C.2.3 [**Computer-communication Networks**]: Network Operations—*Network Management*

General Terms

Reliability, Algorithms, Experimentation

Keywords

network architecture, failure recovery, optimization

SIGMETRICS'11, June 7–11, 2011, San Jose, California, USA.

1. INTRODUCTION

To ensure uninterrupted data delivery, communication networks must distribute traffic efficiently even as links and routers fail and recover. By tuning routing to the offered traffic, *traffic engineering* [27] improves performance and allows network operators to defer expensive outlays of new capacity. Effective *failure recovery* [28,34]—adapting to failures by directing traffic over good alternate paths—is also important to avoid performance disruptions. However, today's networks typically handle failure recovery and traffic engineering independently, leading to more complex routers and less efficient paths after failures. In this paper, we propose an integrated solution with much simpler routers that balances load effectively under a range of failure scenarios.

We argue that traffic engineering and failure recovery can be achieved by the same underlying approach—dynamically rebalancing traffic across diverse end-to-end paths in response to individual failure events. This reduces the complexity of the routers by moving most functionality to the management system—an algorithm run by the network operator. Our network architecture has three key features:

Precomputed multipath routing: Traffic between each pair of edge routers is split over multiple paths that are configured in advance. The routers do *not* compute (or recompute) paths, reducing router overhead and improving path stability. Instead, the management system computes paths that offer sufficient diversity across a range of failure scenarios, including correlated failures of multiple links.

Path-level failure detection: The ingress routers perform failure recovery based only on which *paths* have failed. A minimalist control plane performs path-level failure detection and notification, in contrast to the link-level probing and network-wide flooding common in today's intradomain routing protocols. This leads to simpler, cheaper routers.

Local adaptation to path failures: Upon detecting path failures, the ingress router rebalances the traffic on the remaining paths, based only on which path(s) failed—*not* on load information. This avoids having the routers distribute real-time updates about link load and prevents instability. Instead, the management system *precomputes* the reactions to path failures and configures the routers accordingly.

The first two features—multiple precomputed paths and path-level monitoring—are ideas that have been surfacing

(sometimes implicitly) in the networking literature over the past few years (e.g., [4, 14, 24, 39], and many others). Our architecture combines these two ideas in a new way, through (i) a specific proposal for the "division of labor" between the routers and the management system and (ii) an integrated view of traffic engineering and failure recovery within a single administrative domain. To support the simple network elements, the management system makes *network-wide* decisions based on the expected traffic, the network topology, and the groups of links that can fail together. The management system does *not* need to make these decisions in real time—quite the contrary, offline algorithms can compute the paths and the adaptations to path failures.

Our architecture raises important questions about (i) what configuration state the routers should have to drive their local reactions to path failures and (ii) how the management system should compute this state, and the underlying paths, for good traffic engineering and failure recovery. In addressing these questions, we make four main contributions:

Simple architecture for joint TE and failure recovery (Section 2): We propose a joint solution for traffic engineering and failure recovery, in contrast to today's networks that handle these problems separately. Our minimalist control plane has routers balance load based only on path-failure information, in contrast to recent designs that require routers to disseminate link-load information and compute new path-splitting parameters in real time [18, 23].

Network-wide optimization across failure scenarios (Section 3): We formulate and solve network-wide optimization problems for configuring the routers. Our algorithms compute (i) multiple paths that distribute traffic efficiently under a range of failure scenarios and (ii) the state for each ingress router to adapt to path failures. We present algorithms for two router designs that strike a different trade-off between router state and load-balancing performance.

Experiments with measurement data from a large ISP (Section 4): We evaluate our algorithms on measurement data from a tier-1 ISP network. Our simulation achieves a high degree of accuracy by utilizing the real topology, link capacities, link delays, hourly traffic matrices, and Shared Risk Link Groups (SRLGs) [13]. Our experiments show that one of our candidate router designs achieves near-optimal load balancing across a wide range of failure scenarios, even when the traffic demands change *dynamically*.

Deployability in ISP and data-center networks (Section 5): While our architecture enables simpler routers and switches, existing equipment can support our solutions. ISP backbones can use RSVP to signal multiple MPLS [29] paths, with hash-based splitting of traffic over the paths. In data centers, the fabric controller can configure multiple paths through the network, and the server machines can encapsulate packets to split traffic in the desired proportions.

The paper ends with related work in Section 6, conclusion in Section 7, and supporting proofs in an Appendix.

2. SIMPLE NETWORK ARCHITECTURE

Our architecture uses simple, cheap routers to balance load before, during, and after failures by placing most functionality in a management system that performs offline optimization. The network-management system computes multiple diverse paths between each pair of edge routers, and tells each ingress router how to split traffic over these paths under a range of failure scenarios. Each edge router simply

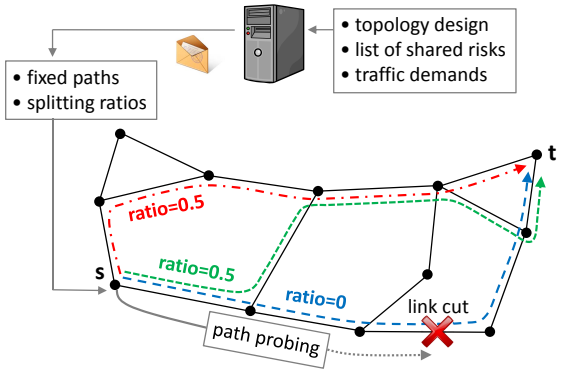

Figure 1: The management system calculates a fixed set of paths and splitting ratios, based on the topology, traffic demands, and potential failures. The ingress router learns about path failures and splits traffic over the remaining paths, based on preconfigured splitting ratios.

detects path-level failures and uses this information to adjust the splitting of traffic over the remaining paths, as shown in Figure 1. The main novel feature of our architecture is the way routers split traffic over the working paths; we propose two approaches that introduce a trade-off between router state and load-balancing performance.

2.1 Precomputed Multipath Routing

Many existing routing protocols compute a single path between each pair of routers, and change that path in response to topology changes. However, dynamic routing has many downsides, including the overhead on the routers (to disseminate topology information and compute paths) and the transient disruptions during routing-protocol convergence. Techniques for making convergence faster tend to increase the complexity of the routing software and the overhead on the routers, by disseminating more information or updating it more quickly. Rather than trying to reduce convergence time, or add mechanisms to detect transient loops and black-holes, we avoid dynamic routing protocols entirely [4].

Our architecture uses multiple *preconfigured* paths between each pair of edge routers, allowing ingress routers to adapt to failures by shifting traffic away from failed path(s). With multiple paths through the network, the routers do not need to recompute paths dynamically—they simply stop using the failed paths until they start working again. This substantially reduces router software complexity and protocol overheads (e.g., bandwidth and CPU resources), while entirely side-stepping the problem of convergence. Instead, the management system computes these paths, based on *both* traffic-engineering and failure-recovery goals, and installs the paths in the underlying routers. The management system can select diverse paths that ensure connectivity in the face of failures, including multiple correlated failures.

Using multiple paths also leads to better load balancing, whether or not failures occur. Today's shortest-path routing protocols (like OSPF and IS-IS) use a single path, or (at best) only support *even* splitting of traffic over multiple *shortest* paths. Our architecture (like other recent proposals for multipath load balancing [6, 14, 17, 37]) allows flexible splitting of traffic over multiple paths. However, we do

	Optimal (Baseline)	State-Dependent Splitting	State-Independent Splitting
Router state	Exponential in total # of links	Exponential in # of pre-configured paths between two routers	Linear in # of pre-configured paths between two routers
Failure information	Link level	Path level	Path level
Optimality	Optimal	Nearly-optimal	Good

Table 1: Properties of the candidate solutions. The solutions differ in the amount of configuration state that must be stored in the routers, the information the routers must obtain about each failure, and the achieved traffic-engineering performance.

not require dynamic adaptation of the traffic splitting. Instead, the ingress router has a simple static configuration that determines the splitting of traffic over the available paths, while intermediate routers merely forward packets over pre-established paths. The management system optimizes this configuration *in advance* based on a network-wide view of the expected traffic and likely failures. This avoids the protocol overhead and stability challenges of distributed, load-sensitive routing protocols. Also, the management system can use knowledge about shared risks and anticipated traffic demands—information the routers do not have.

2.2 Path-Level Failure Detection

Most routing protocols detect failures by exchanging "hello" messages between neighboring routers and flooding the topology changes through the network. This approach requires small timers for fast failure detection, imposing additional overhead on the routers. In addition, many failures are triggered by planned maintenance [21], leading to *two* convergence events—one for the link failure(s), and another for the recovery—that both cause transient disruptions. In addition, "hello" messages do not detect all kinds of failures—some misconfigurations (e.g., having a maximum packet size that is too small) and attacks (e.g., an adversary selectively dropping packets) do not lead to lost "hello" messages.

Instead, our architecture relies on *path-level* failure detection. Each ingress-egress router pair has a session to monitor each of its paths (e.g., as in BFD [15]). The probes can be piggybacked on existing data traffic, obviating the need for separate "hello" messages when the path is carrying regular data traffic. This enables fast failure detection without introducing extra probe traffic, and the "implicit probes" provide a more realistic view of the reliability of a path [3, 11], since the packets vary in size, addresses, and so on. Another advantage is that the packets are handled by the hardware interfaces and, as such, do not consume processing resources (or experience software processing delays) at intermediate routers. (Still, the propagation delay along a path does impose limits on detection time in large topologies, an issue we discuss in more detail in Section 5.)

Although the ingress router doesn't learn which *link* failed, knowledge of the *path* failures is sufficient to avoid the failed path. In fact, since the routers need not be aware of the topology, no control protocol is needed to exchange topology information. In fact, only some of the ingress routers need to learn about the failure—only the routers that have paths traversing the failed edge. The other ingress routers, and the intermediate routers, can remain unaware of the link failure. Of course, the *management system* ultimately needs to know about topology changes, so failed equipment can be fixed or replaced. But this detection problem can be handled on a much longer timescale since it does not affect the failure-recovery time for data traffic.

2.3 Local Adaptation to Path Failures

In our architecture, a router is a simple device that does not participate in a routing protocol, collect congestion feedback, or solve any computationally difficult problems. Still, the routers do play an important role in adapting the distribution of traffic when paths fail or recover, at the behest of the management system. We propose two different ways the routers can split traffic over the working paths: (i) state-independent splitting which has minimal router state and (ii) state-dependent splitting which introduces more state in exchange for near-optimal performance, as summarized (and compared to an idealized solution) in Table 1.

Optimal load balancing: This idealized solution calculates the optimal paths and splitting ratios separately for each possible failure state, i.e., for each combination of link failures. This approach achieves the best possible load balancing. However, the approach is impractical because the routers must (i) store far too much state and (ii) learn about all link failures—even on links the router's paths do not traverse. Therefore, this solution would violate our architecture. However, the solution is still interesting as it provides a lower bound on the amount of congestion achievable by the other two schemes.

State dependent splitting: In this solution, each ingress router has a separate configuration entry with path-splitting weights for each combination of *path* failures to a particular egress router. For example, suppose a router has three paths to an egress router. Then, the router configuration contains seven entries—one for each of the $2^3 - 1$ combinations of path failures. Each configuration entry, computed ahead of time by the management system, consists of three weights—one per path, with a 0 for any failed paths. Upon detecting path failures, the ingress router inspects a pre-configured table to select the appropriate weights for splitting the traffic destined to the egress router. Our experiments in Section 4 show that, even in a large ISP backbone, having three or four paths is sufficient, leading to modest state requirements on the router in exchange for near-optimal load balancing.

State independent splitting: This solution further simplifies the router configuration by having a *single* set of weights across all failure scenarios. So, an ingress router with three paths to an egress router would have only *three* weights, one for each path. If any paths fail, the ingress router simply renormalizes the traffic on the remaining paths. As such, the management system must perform a *robust* optimization of the limited configuration parameters to achieve

good load-balancing performance across a range of failure scenarios. Our experiments in Section 4 show that this simple approach can perform surprisingly well, but understandably not as well as state-dependent splitting.

3. NETWORK-WIDE OPTIMIZATION

In our architecture, the network-management system performs network-wide optimization to compute paths and traffic-splitting ratios that balance load effectively across a range of failure scenarios. In this section, we first discuss the information the management system has about the network topology, traffic demands, and shared risks. Then, we explain how the management system computes the multiple diverse paths and the traffic-splitting ratios, for both state-dependent and state-independent splitting. We solve all optimization problems either by formulating them as convex optimizations solvable in polynomial time, or by providing heuristics for solving NP-hard problems. Table 2 summarizes the notation.

3.1 Network-Wide Visibility and Control

The management system computes paths and splitting ratios based on a network-wide view:

Fixed topology: The management system makes decisions based on the designed topology of the network—the routers and links that have been deployed. The topology is represented by a graph $G(V, E)$ with a set of vertices V and directed edges E. The capacity of edge $e \in E$ is denoted by c_e, and the propagation delay on the edge is y_e.

Shared risk link groups: The management system knows which links share a common vulnerability, such as connecting to the same line card or router or traversing the same optical fiber or amplifier [13]. The shared risks are denoted by the set S, where each $s \in S$ consists of a set of edges that may fail together. For example, a router failure is represented by the set of its incident links, a fiber cut is represented by all links in the affected fiber bundle, and the failure-free case is represented by the empty set $\emptyset$. Operators also have measurement data from past failures to produce estimates for the likelihood of different failures (e.g., an optical amplifier may fail less often than a line card). As such, each failure state s has a weight w^s that represents its likelihood or importance.

Variable	Description
$G(V, E)$	network with vertices V and directed edges E
c_e	capacity of edge $e \in E$
y_e	propagation delay on edge $e \in E$
S	family of network failure states
s	network failure state (set of failed links)
w^s	weight of network failure state $s \in S$
D	set of demands
u_d	source of demand $d \in D$
v_d	destination of demand $d \in D$
h_d	flow requirement of demand $d \in D$
P_d	paths available to demand $d \in D$
α_p	fraction of the demand assigned to path p
O_d	family of observable failure states for node u_d
$o_d(s)$	state observable by u_d in failure state $s \in S$
P_d^o	paths available to u_d in failure state $o \in O_d$
f_p^s	flow on path p in failure state $s \in S$
f_p^o	flow on path p in failure state $o \in O_d$
l_e^s	total flow on edge e in failure state s
$l_{e,d}^s$	flow of demand d on edge e in failure state s

Table 2: Summary of notation

Expected traffic demands: The management system knows the anticipated traffic demands, based on past measurements and predictions of traffic changes. Each traffic demand $d \in D$ is represented by a triple (u_d, v_d, h_d), where $u_d \in V$ is the traffic source (ingress router), $v_d \in V$ is the destination (egress router), and h_d is the flow requirement (measured traffic). For simplicity, we assume that all demands remain connected for each failure scenario; alternatively, a demand can be omitted for each failure case that disconnects it. In practice, the management system may have a time sequence of traffic demands (e.g., for different hours in the day), and optimize the network configuration across all these demands, as we discuss in Section 4.3.

The management system's output is *set of paths* P_d for each demand d and the *splitting ratios* for each path. In each failure state s, the traffic splitting by ingress router u_d depends only on which *paths* have failed, not which failure scenario s has occurred; in fact, multiple failure scenarios may affect the same subset of paths in P_d. To reason about the handling of a particular demand d, we consider a set O_d of "observable" failure states, where each observable state $o \in O_d$ corresponds to a particular $P_d^o \subset P_d$ representing the available paths. For ease of expression, we let the function $o_d(s)$ map to the failure state observable by node u_d when the network is in failure state $s \in S$. The amount of flow assigned to path p in observable failure state $o \in O_d$ is f_p^o. The total flow on edge e in failure state s is l_e^s, and the flow on edge e corresponding to demand d is $l_{e,d}^s$.

The management system's goal is to compute paths and splitting ratios that minimize congestion over the range of possible failure states. A common traffic-engineering objective [9] is to minimize $\sum_{e \in E} \Phi(l_e^s/c_e)$ where l_e is the load on edge e and c_e is its capacity. $\Phi()$ could be a convex function of link load [9], to penalize the most congested links while still accounting for load on the remaining links. The final objective minimizing congestion across failure scenarios is

$$obj(l_{e_1}^{s_1}/c_{e_1}, ...) = \sum_{s \in S} w^s \sum_{e \in E} \Phi(l_e^s/c_e). \quad (1)$$

Minimizing this objective function is the goal of all the candidate solutions in the following section. The constraints that complete the problem formulation differ depending on the functionality placed in the underlying routers.

3.2 Computing Multiple Diverse Paths

The management system must compute multiple diverse paths that ensure good load balancing—and (most importantly) continued connectivity—across a range of failure scenarios. However, as shown later, computing the optimal paths for state-dependent and state-independent splitting is NP-hard. Instead, we propose a heuristic: using the collection of paths computed by the optimal solution that optimizes for each failure state independently. This guarantees that the paths are sufficiently diverse to ensure traffic de-

livery in all failure states, while also making efficient use of network resources.

The idealized optimal solution has a separate set of paths and splitting ratios in each failure state s. To avoid having variables for exponentially many paths, we formulate the problem in terms of the amount of flow $l^s_{e,d}$ from demand d traversing edge e for failure state s. The optimal edge loads are obtained by solving the convex optimization:

$$\begin{array}{lll} \min & obj(l^{s_1}_{e_1}/c_{e_1}, ...) & \\ \text{s.t.} & l^s_e = \sum_{d \in D} l^s_{e,d} & \forall s, e \\ & 0 = \sum_{i:e=(i,j)} l^s_{e,d} - \sum_{i:e=(j,i)} l^s_{e,d} & \forall d, s, j \neq u_d, v_d \\ & h_d = \sum_{i:e=(u_d,i)} l^s_{e,d} - \sum_{i:e=(i,u_d)} l^s_{e,d} & \forall d, s \\ & h_d = \sum_{i:e=(i,v_d)} l^s_{e,d} - \sum_{i:e=(v_d,i)} l^s_{e,d} & \forall d, s \\ & 0 \leq l^s_{e,d} & \forall d, s, e, \end{array} \quad (2)$$

where l^s_e and $l^s_{e,d}$ are variables. The first constraint defines the load on edge e, the second constraint ensures flow conservation, the third and fourth constraints ensure that the demands are met, and the last constraint guarantees flow non-negativity. An optimal solution can be found in polynomial time using conventional techniques for solving multi-commodity flow problems.

After obtaining the optimal flow on each edge for all the failure scenarios, we use a standard decomposition algorithm to determine the corresponding paths P_d and the flow f^s_p on each of them. The decomposition starts with a set P_d that is empty. New unique paths are added to the set by performing the following decomposition for each failure state s. First, annotate each edge e with the value $l^s_{e,d}$. Remove all edges that have 0 value. Then, find a path connecting u_d and v_d. Although we could choose any of the paths from u_d to v_d, our goal is to obtain paths that are as short as possible. So, if multiple such paths exist, we use the path p with the smallest propagation delay. Add this path p to the set P_d and assign to it flow f^s_p equal to the smallest value of the edges on path p. Reduce the values of these edges accordingly. Continue in this fashion, removing edges with zero value and finding new paths, until there are no remaining edges in the graph.

Note that we can show by induction that this process completely partitions the flow $l^s_{e,d}$ into paths. The decomposition yields at most $|E|$ paths for each network failure state s because the value of at least one edge becomes 0 whenever a new path is found. Hence the total size of the set P_d is at most $|E||S|$. It is difficult to obtain a solution that restricts the number of paths as we prove in the appendix that it is NP-hard to solve problem (2) when the number of allowed paths is bounded by a constant J. In practice, the algorithm produces a relatively small number of paths between each pair of edge routers, as shown later in Section 4.

3.3 Optimizing the Traffic-Splitting Ratios

Once the paths are computed, the network-management system can optimize the path-splitting ratios for each ingress-egress router pair. The optimization problem and the resulting solution depend on whether the routers perform state-dependent or state-independent splitting.

3.3.1 State-Dependent Splitting

In state-dependent splitting, each ingress router u_d has a set of splitting ratios for each *observable* failure state $o \in O_d$. Since the path-splitting ratios depend on *which* paths in P_d have failed, the ingress router must store splitting ratios for $min(|S|, 2^{|P_d|})$ scenarios; fortunately, the number of paths $|P_d|$ is typically small in practice. When the network performs such *state-dependent splitting*, the management system's goal is to find a set of paths P_d for each demand and the flows f^o_p on these paths in all observable states $o \in O_d$. If the paths P_d are known and fixed, the problem can be formulated as a convex optimization:

$$\begin{array}{lll} \min & obj(l^{s_1}_{e_1}/c_{e_1}, ...) & \\ \text{s.t.} & l^s_e = \sum_{d \in D} \sum_{p \in P^o_d, e \in p} f^o_p & \forall e, s, o = o_d(s) \\ & h_d = \sum_{p \in P^o_d} f^o_p & \forall d, o \in O_d \\ & 0 \leq f^o_p & \forall d, o \in O_d, p \in P_d, \end{array} \quad (3)$$

where l^s_e and f^o_p are variables. The first constraint defines the load on edge e, the second constraint guarantees that the demand d is satisfied in all observable failure states, and the last constraint ensures non-negativity of flows assigned to the paths. The solution of the optimization problem (3) can be found in polynomial time.

Finding the optimal set of paths $\{P_d\}$ in problem (3) is NP-hard. The Appendix shows that it is NP-hard to construct a path (if one exists) that allows the ingress router to distinguish the failure state s. This is required to decide how to best balance the load. All our formulations where the routers cannot directly observe link failures are NP-hard. Therefore, we use the paths that are found by the decomposition of the optimal solutions (2), as outlined in the previous subsection. Since these paths allow optimal load balancing for the optimal solutions (2), they are also likely to enable good load balancing for the optimization problem (3).

3.3.2 State-Independent Splitting

In state independent splitting, each ingress router has a *single* configuration entry containing the splitting ratios that are used under any combination of path failures. Each path p is associated with a splitting fraction α_p. When one or more paths fail, the ingress router u_d observes state o and uses $\frac{\alpha_p}{\sum_{q \in P^o_d} \alpha_q}$ as the splitting ratio for path p (and 0 for all the failed paths). If the network elements implement such *state-independent splitting*, and the paths P_d are known and fixed, the management system needs to solve the following non-convex optimization problem:

$$\begin{array}{lll} \min & obj(l^{s_1}_{e_1}/c_{e_1}, ...) & \\ \text{s.t.} & f^o_p = h_d \frac{\alpha_p}{\sum_{q \in P^o_d} \alpha_q} & \forall d, o \in O_d, p \in P_d \\ & l^s_e = \sum_{d \in D} \sum_{p \in P^o_d, e \in p} f^o_p & \forall e, s, o = o_d(s) \\ & 0 \leq f^o_p & \forall d, o \in O_d, p \in P_d, \end{array} \quad (4)$$

where l^s_e, f^o_p and α_p are variables. The first constraint ensures that the flow assigned to every available path p is proportional to α_p. The other three constraints are the same as in (3).

Unfortunately, no standard optimization techniques allow us to compute an optimal solution efficiently, even when the paths P_d are fixed. Therefore, we have to rely on heuristics to find both the candidate paths P_d and the splitting ratios α_p. To find the set of candidate paths P_d, we again use the optimal paths obtained by decomposing (2). To find

the splitting ratios we mimic the behavior of the optimal solution as closely as possible. We find the splitting ratios for all paths p by letting $\alpha_p = \sum_{s \in S} \frac{w^s f_p^s}{h_d}$ where f_p^s is the flow assigned by the optimal solution to path p in network failure state s. Since $\sum w^s = 1$, the calculated ratio is the weighted average of the splitting ratios used by the optimal solutions (2).

4. EXPERIMENTAL EVALUATION

To evaluate the algorithms described in the previous section, we wrote a simulator in C++ that calls the CPLEX linear program solver in AMPL and solves the optimization problems (2) and (3). We compare our two heuristics to the optimal solution, a simple "equal splitting" configuration, and OSPF with the link weights set using state-of-the-art optimization techniques. We show that our two heuristics require few paths resulting in compact routing tables, and the round-trip propagation delay does not increase. Finally, using real traffic traces obtained during a 24-hour measurement in the network of a tier-1 ISP we show that our solutions achieve excellent results without the need to perform any reoptimizations even in the presence of a changing traffic matrix.

Our experimental results show that the **objective value of state-dependent splitting very closely tracks the optimal objective**. For this reason, this solution is our favorite. Although state-independent splitting has somewhat worse performance especially as the network load increases beyond current levels, it is also attractive due to its simplicity.

4.1 Experimental Setup

Our simulations use a variety of synthetic topologies, the Abilene topology, as well as the city-level IP backbone topology of a tier-1 ISP with a set of failures provided by the network operator. The parameters of the topologies we used are summarized in Table 3.

Synthetic topologies: The synthetic topologies include 2-level hierarchical graphs, purely random graphs, and Waxman graphs. 2-level hierarchical graphs are produced using the generator GT-ITM [40], for random graphs the probability of two edges being connected is constant, and the probability of having an edge between two nodes in the Waxman graph decays exponentially with the distance of the nodes. These topologies also appear in [8].

Abilene topology: The topology of the Abilene network and a measured traffic matrix are used. We use the true edge capacities of 10 Gbps.

Tier-1 IP backbone: The city-level IP backbone of a tier-1 ISP is used. In our simulations, we use the real link capacities and measured traffic demands. We also obtained the link round-trip propagation delays.

The collection of network failures S for the synthetic topologies and Abilene contains single edge failures and the no-failure case. Two experiments with different collections of failures are performed on the tier-1 IP backbone. In the first experiment, single edge failures are used. In the second experiment, the collection of failures also contains Shared Risk Link Groups (SRLGs), link failures that occur simultaneously. SRLGs were obtained from the network operator's database that contains 954 failures with the largest failure affecting 20 links simultaneously. For each potential line card failure, a complete router failure, or a link cut there is a corresponding record in the SRLG database. Therefore, failures that do not appear in the database are rare. The weights w^s in the optimization objective (1) were set to 0.5 for the no-failure case, and all other failure weights are equal and sum to 0.5.

The set of demands D in the Abilene and tier-1 networks were obtained by sampling Netflow data measured on Nov. 15th 2005 and May 22nd 2009, respectively. For the synthetic topologies, we chose the same traffic demands as in [8].

To simulate the algorithms in environments with increasing congestion, we repeat all experiments several times while uniformly increasing the traffic demands. For the synthetic topologies we start with the original demands and scale them up to twice the original values. As the average link utilization in Abilene and the tier-1 topology is lower than in the synthetic topologies, we scale the demands in these realistic topologies up to three times the original value.

In our experiments we use the piecewise linear penalty function defined by $\Phi(0) = 0$ and its derivatives:

$$\Phi'(\ell) = \begin{cases} 1 & \text{for } 0 \leq \ell < 0.333 \\ 3 & \text{for } 0.333 \leq \ell < 0.667 \\ 10 & \text{for } 0.667 \leq \ell < 0.9 \\ 70 & \text{for } 0.9 \leq \ell < 1 \\ 500 & \text{for } 1 \leq \ell < 1.1 \\ 5000 & \text{for } 1.1 \leq \ell < \infty \end{cases}$$

This penalty function was introduced in [9], and allows one to formulate the optimizations (2) and (3) as linear programs by adding auxiliary variables. The function can be viewed as modeling retransmission delays caused by packet losses. The cost is small for low utilization, and increases steeply as the utilization exceeds 100%.

Our simulation calculates the objective value of the optimal solution, state-independent and state-dependent splitting, and equal splitting. Equal splitting is a variant of state-independent splitting that splits the flow evenly on the available paths. We also calculate the objective achieved by the shortest path routing of OSPF with optimized link weights. These link weights were calculated using the state-of-the-art optimizations of [8], and these optimizations take into consideration the set of failure states S and the corresponding failure weights w^s.

Our simulations were performed using CPLEX version 11.2 on a 1.5 GHz Intel Itanium 2 processor. Solving the linear program for (2) for a particular failure case in the tier-1 topology takes 4 seconds, and solving the linear pro-

Name	Topology	Nodes	Edges	Demands
hier50a	hierarchical	50	148	2,450
hier50b	hierarchical	50	212	2,450
rand50	random	50	228	2,450
rand50a	random	50	245	2,450
rand100	random	100	403	9,900
wax50	Waxman	50	169	2,450
wax50a	Waxman	50	230	2,450
Abilene	backbone	11	28	110
tier-1	backbone	50	180	625

Table 3: Synthetic and realistic network topologies.

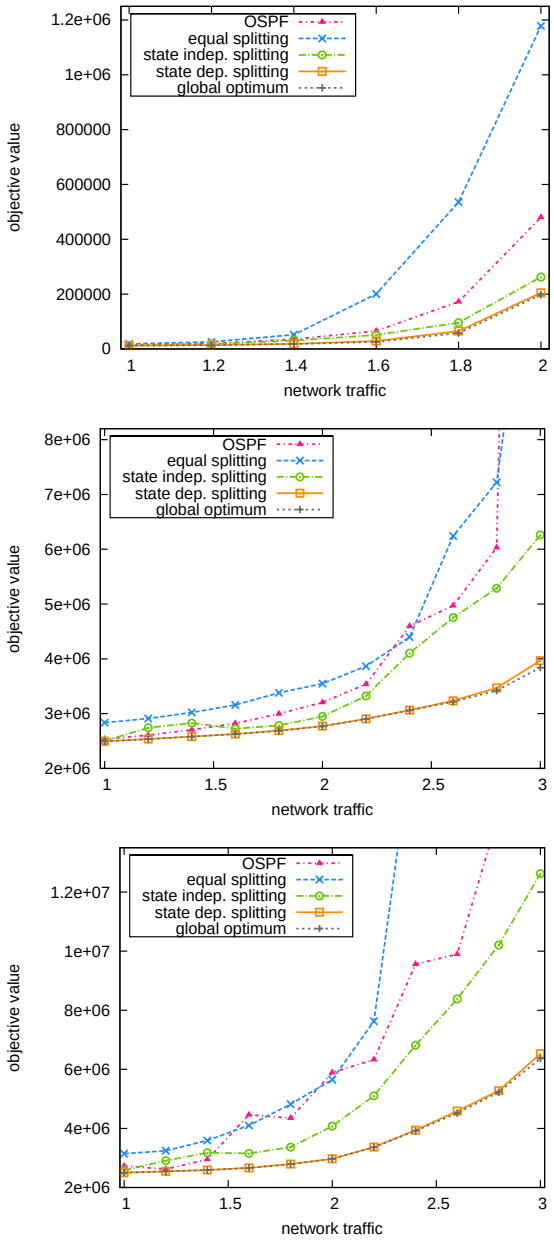

Figure 2: From top to bottom the traffic engineering objective as a function of an increasing traffic load in the hierarchical topology hier50a, tier-1 topology with single edge failures, and tier-1 topology with SRLGs, respectively. The performance of the optimal solution and state-dependent splitting is nearly identical.

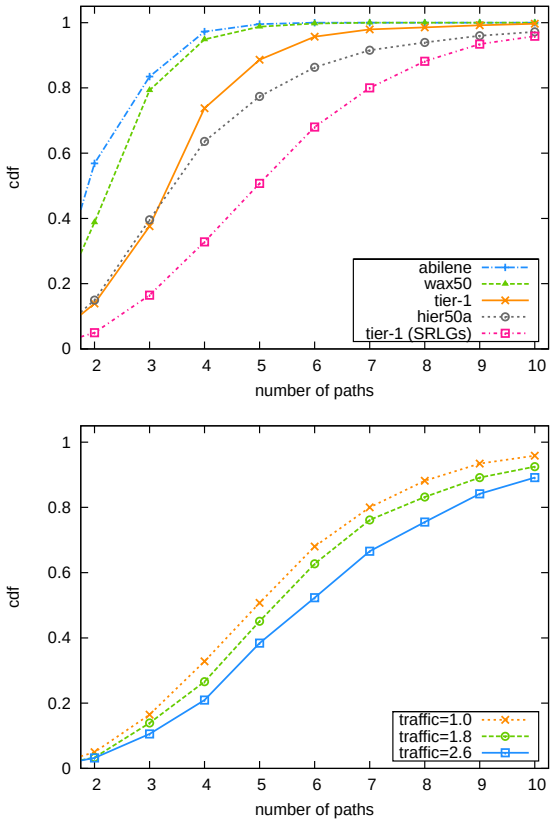

Figure 3: The number of paths used in various topologies at the top, and in the tier-1 topology with SRLGs at the bottom. The cumulative distribution function shows that the number of paths is almost independent of the traffic load in the network, but is larger for bigger, more well-connected topologies.

gram (3) takes about 16 minutes. A tier-1 network operator can perform calculations for its entire city-level topology in less than 2 hours.

4.2 Performance with Static Traffic

Avoiding congestion and packet losses during planned and unplanned failures is the central goal of traffic engineering. Our traffic engineering objective measures congestion across all the considered failure cases. The objective as a function of the scaled-up demands is depicted in Figure 2. The results which were obtained on the hierarchical and tier-1 topologies are representative, we made similar observations for all the other topologies. In Figure 2, the performance of state-dependent splitting and the optimal solution is virtually indistinguishable in all cases. State-independent splitting is less sophisticated and does not allow custom load balancing ratios for distinct failures, and therefore its performance is worse compared to the optimum. It is not surprising that the equal splitting algorithm achieves the worst performance.

We observe that OSPF achieves a somewhat worse performance than state-independent and state-dependent splitting as the load increases. However, we should note that in OSPF, each router is restricted to sending all its traffic on the single path with the smallest weight, or splitting the traffic evenly if multiple smallest-weight paths exist. This approach does not allow the same flexibility in choosing routes and splitting ratios as our solution, and, therefore, OSPF should not be expected to achieve the same performance even for an optimal choice of OSPF link weights.

Solutions with few paths are preferred as they decrease the number of tunnels that have to be managed, and reduce the size of the router configuration. However, a sufficient number of paths must be available to avoid failures and to

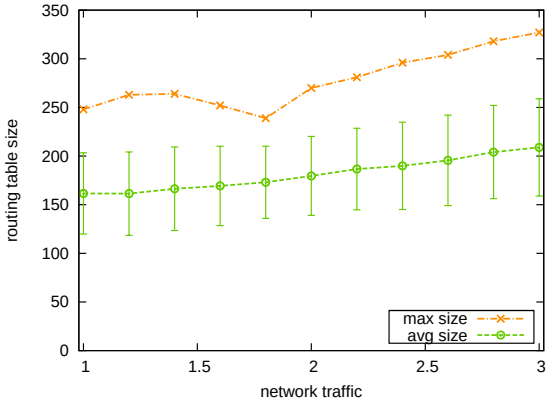

Figure 4: Size of the compressed routing tables in the tier-1 topology with SRLGs. The largest and average routing table sizes (± one standard deviation) in the backbone routers are shown.

reduce congestion. We observe that the number of paths used by our algorithms is small. We record the number of paths used by each demand, and plot the distribution in Figure 3. Not surprisingly, the number of paths is greater for larger and more diverse topologies. 92% of the demands in the hierarchical topology use 7 or fewer paths, and fewer than 10 paths are needed in the tier-1 backbone topology for almost all demands. Further, Figure 3 shows that the number of paths only increases slightly as we scale up the amount of traffic in the networks. This small increase is caused by shifting some traffic to longer paths as the short paths become congested.

A practical solution uses few MPLS labels in order to reduce the size of routing tables in the routers. Our experimental results reveal that when we use MPLS tunnels in the tier-1 topology, a few thousand tunnels can pass through a single router. However, a simple routing table compression technique allows us to reduce the routing table size to a few hundred entries in each router. Such compression is important because it reduces the memory requirements imposed on the simple routers whose use we advocate, and it improves the route lookup time.

Routing tables can be compressed by using the same MPLS labels for routes with a common path to the destination. Specifically, if two routes to destination t pass through router r, and these routes share the same path between the router r and the destination t, the same outbound label should be used in the routing table of router r. The resulting routing table sizes as a function of the network load are depicted in Figure 4. The curve on the top shows the size of the largest routing table, and the curve on the bottom shows the average routing table size among all the backbone routers.

Minimizing the delay experienced by the users is another important goal of network operators. We calculated the average round-trip propagation delays of all the evaluated algorithms. The calculated delays include delays in all failure states weighted by the corresponding likelihood of occurrence, but exclude congestion delay which is negligible. The delays are summarized in Table 4. We observe that the round-trip delay of all algorithms except equal splitting is almost identical at around 31 ms. These values would satisfy the 37 ms requirement specified in the SLAs of the tier-1 network. Moreover, these values are not higher than these experienced by the network users today. To demonstrate this, we repeated our simulation on the tier-1 topology using the real OSPF weights which are used by the network operator. These values are chosen to provide a tradeoff between traffic engineering and shortest delay routing. The results which appear in Table 4 in the row titled OSPF (current) show that the current delays are 31.38 ms for each of the two tier-1 failure sets.

Algorithm	Single edge	SRLGs
Optimal load balancing	31.75 ± 0.26	31.80 ± 0.25
State dep. splitting	31.51 ± 0.17	31.61 ± 0.16
State indep. splitting	31.76 ± 0.26	31.87 ± 0.25
Equal splitting	34.83 ± 0.33	40.85 ± 0.86
OSPF (optimized)	31.18 ± 0.40	31.23 ± 0.40
OSPF (current)	31.38 ± 0	31.38 ± 0

Table 4: Round-trip propagation delay in ms (average ± one standard deviation) in the tier-1 backbone network for single edge failures and SRLG failures.

4.3 Robust Optimization for Dynamic Traffic

Solving the optimization problems repeatedly as the traffic matrix changes is undesirable due to the need to update the router configurations with new paths and splitting ratios. We explore the possibility of using a single router configuration that is robust to diurnal changes of the demands.

To perform this study we collected hourly netflow traffic traces in the tier-1 network on September 29, 2009. We denote the resulting 24 hourly traffic matrices $D^0, D^1, ..., D^{23}$. Figure 5 depicts the aggregate traffic volume, as well as example of the traffic between three ingress-egress router pairs. The aggregate traffic volume is the lowest at 9 a.m. GMT and peaks with 2.5 times as much traffic at midnight and 8 p.m. GMT. Comparison to the three depicted ingress-egress router demands reveals that the traffic during a day cannot be obtained by simple scaling as the individual demands peak at different times. This makes the joint optimization challenging.

The first step in the joint optimization is to calculate a single set of paths that guarantee failure resilience and load balancing for each of the 24 traffic matrices. There are several approaches we can take. In the first approach, we solve the linear program for (2) for each traffic matrix D^i separately and use the union of the paths obtained for each matrix. The second approach is to calculate the average traffic matrix $D = \frac{1}{24}\sum_i D^i$. The linear program for (2) is then solved for the average traffic matrix. In the third approach we use the envelope of the 24 traffic matrices instead of the average, i.e., we let $D_{jk} = max_i D^i_{jk}$.

In our simulations we chose the last method. Compared to the first method, it results in fewer paths. Compared to the second method, it allows better load balancing because demands between ingress-egress pairs with high traffic variability throughout the day are represented by the peak traffic.

The second step is to calculate router configuration robust to traffic changes. We again use the envelope $D_{jk} = max_i D^i_{jk}$ as the input traffic matrix and repeat the opti-

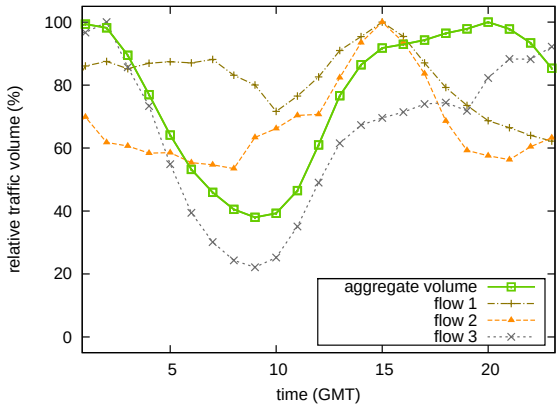

Figure 5: The aggregate traffic volume in the tier-1 network has peaks at midnight GMT and 8 p.m. GMT. Examples of three demands show that their peaks occur at different times of the day.

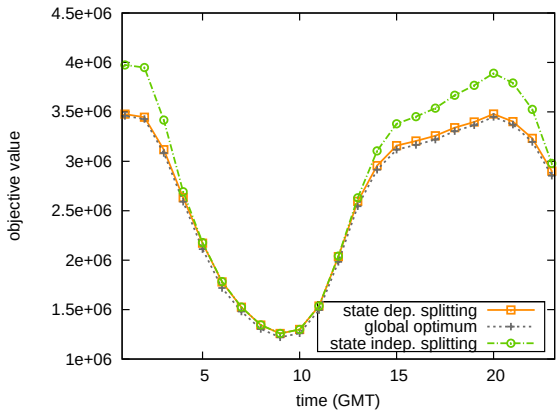

Figure 6: The traffic engineering objective in the tier-1 topology with SRLGs. The state dependent and state independent splitting algorithms use a single configuration throughout the day. The optimal solution uses a custom configuration for each hour.

mizations from the previous section. Then we test the solution by simulating the varying traffic demand during one day period. The resulting objective value of state dependent splitting and state independent splitting is depicted in Figure 6. The optimal objective in Figure 6 represents the performance of the best possible solution that uses custom configuration updated hourly. We observe that state dependent splitting with a single configuration is robust to diurnal traffic changes and the value of its objective closely tracks the optimum. State independent splitting is also close to optimal during low congestion periods, but becomes suboptimal during the peak hours.

5. DEPLOYMENT SCENARIOS

Although our architecture enables the use of new simpler routers, we can readily deploy our solutions using existing protocols and equipment, as summarized in Table 5. An ISP can deploy our architecture using Multi-Protocol Label Switching (MPLS) [29]. Data centers could use the same solution, or leverage existing Ethernet switches and move some functionality into the end-host machines.

	ISP Backbone	Data Center
Network element	MPLS router	Ethernet switch
Path installation	RSVP	VLAN trunking
Traffic splitting	Ingress router	End host
Failure detection	BFD	Host probing
Fast recovery	Ingress router	End host
Traffic demand	MPLS MIB	Host/VLAN counter

Table 5: Existing tools and protocols that can be used to deploy our architecture.

5.1 ISP Backbone Using MPLS

Installing MPLS paths with RSVP: MPLS is particularly suitable because ingress routers encapsulate packets with labels and direct them over pre-established Label-Switched Paths (LSPs). This enables flexible routing when multiple LSPs are established between each ingress-egress router pair. Our solution, then, could be viewed as a particular application of MPLS, where the management system computes the LSPs, instructs the ingress routers to establish the paths (say, using RSVP), and disables any dynamic recalculation of alternate paths when primary paths fail.

Hash-based splitting at ingress routers: Multipath forwarding is supported by commercial routers of both major vendors [2, 27]. The routers can be configured to hash packets based on port and address information in the headers into several groups and forward each group on a separate path. This provides path splitting with relatively fine granularity (e.g., at the 1/16th level), while preventing out-of-order packet delivery by ensuring that packets belonging to the same TCP or UDP flow traverse the same path.

Path-level failure detection using BFD: Fast failure detection can be done using Bidirectional Forwarding Detection (BFD) [15]. A BFD session can monitor each path between two routers, by piggybacking on the existing data traffic. (Backbones covering a large geographic region may also use existing link-level detection mechanisms for even faster recovery. For example, *local path protection* [28] installs a short alternate path between two adjacent routers, for temporary use after the direct link fails. However, local protection cannot fully exploit the available path diversity, leading to suboptimal load balancing; instead, local protection can be used in conjunction with our design.)

Failure recovery at ingress router: The ingress router adapts to path failures by splitting traffic over the remaining paths. In state-independent splitting, the ingress router has a single set of traffic-splitting weights, and automatically renormalizes to direct traffic over the working paths. State-dependent splitting requires modification to the router software to switch to alternate traffic-splitting weights in the data plane; no hardware modifications are required.

Measuring traffic demands using SNMP: MPLS has SNMP counters (called Management Information Bases) that measure the total traffic traversing each Label-Switched Path. The management system can poll these counters to measure the traffic demands. Alternative measurement techniques, such as Netflow or tomography, may also be used.

5.2 Data Center Using Hosts and Switches

While a data center could easily use the same MPLS-based solution, control over the end host and the availability of cheaper commodity switches enable another solution.

End-host support for monitoring and traffic splitting: The server machines in data centers can perform many of the path-level operations in our architecture. As in the VL2 [12] and SPAIN [24] architectures, the end host can encapsulate the packets (say, using a VLAN tag) to direct them over a specific path. This enables much finer-grain traffic splitting. In addition, the end host can perform path-level probing in the data plane, by piggybacking on existing data traffic and sending additional active probes when needed. Upon detecting path failures, the end host can change to new path-splitting percentages based on the precomputed configuration installed by the controller. The end host could also measure the traffic demands by keeping counts of the traffic destined to each egress switch. These functions can be implemented in the hypervisor, such as the virtual switch that often runs on server machines in data centers.

Multiple VLANs or OpenFlow rules for forwarding: The remaining functions can be performed by the underlying switches. For example, the management system can configure multiple paths by merging these paths into a set of trees, where each tree corresponds to a different VLAN [24]. Or, if the switches support the emerging OpenFlow standard [1, 22], the management system could install a forwarding-table rule for each hop in each path, where the rule matches on the VLAN tag, and forwards the packet to the appropriate output port. Since OpenFlow switches maintain traffic counters for each rule, the management system can measure the traffic demands by polling the switches, in lieu of the end hosts collecting these measurements.

6. RELATED WORK

Traffic engineering: Most of the related work treats failure recovery and traffic engineering independently. Traffic engineering without failure recovery in the context of MPLS is studied in [5, 6, 19, 32, 37]. The work in [5] utilizes traffic splitting to minimize end-to-end delay and loss rates; however, an algorithm for optimal path selection is not provided. The works in [19] and [32] minimize the maximum link utilization while satisfying the requested traffic demands. Other papers [6, 14, 17, 37] prevent congestion by adaptively balancing the load among multiple paths based on measurements of congestion, whereas our solution *precomputes* traffic-splitting configurations based on both the offered traffic and the likely failures.

Failure recovery: Local and global path protection are popular failure recovery mechanisms in MPLS. In local protection the backup path takes the shortest path that avoids the outage location from a point of local repair to the merge point with the primary path. The IETF RFC 4090 [28] focuses on defining signaling extensions to establish the backup paths, but leaves the issues of bandwidth reservation and optimal route selection open. In [35] the shortest path that avoids the failure is used. While [31] and [36] attempt to find optimal backup paths with the goal of reducing congestion, local path protection is less suitable for traffic engineering than global path protection, which allows rerouting on end-to-end paths [34]. Other work describes how to manage restoration bandwidth and select optimal paths [16, 20]. While our solution also uses global protection to reroute around failures, the biggest difference is that most of the related work distinguishes primary and backup paths and only uses a backup path when the primary path fails. In contrast, our solution balances the load across multiple paths even before failures occur, and simply adjusts the splitting ratios in response to failures.

Integrated failure recovery and TE: Some proposals only use alternate paths when primary routes fail [30], or they require explicit congestion feedback and do not provide algorithms to find the optimal paths [18, 23]. YAMR [10] constructs a set of diverse paths in the interdomain routing setting that are resilient against a specified set of failures, but without regard to load balancing. In [41] they integrate failure recovery with load balancing, but their focus is different—they guarantee delivery of a certain fraction of the traffic after a single edge failure, whereas our goal is to deliver *all* traffic for a known set of multi-edge failures. In [38] they propose an architecture that handles up to F link failures by using local rerouting, subject to link capacity constraints. Unlike [38], our work uses end-to-end routing, and does not require link state flooding and dynamic router reconfigurations. Proposals that optimize OSPF or IS-IS link weights with failures in mind, such as [8] and [26], must rely on shortest path IGP routing and therefore cannot fully utilize the path diversity in the network.

Failure recovery and TE with multiple spanning trees: Enterprise and data-center networks often use Ethernet switches, which do not scale well because all traffic flows over a single spanning tree, even if multiple paths exist. Several papers propose more scalable Ethernet designs that use multiple paths. The work of Sharma et al. uses VLANs to exploit multiple spanning trees to improve link utilization, and achieve improved fault recovery [33]. Most of the designs such as VL2 [12], and PortLand [25] rely on equal splitting of traffic on paths with the same cost. SPAIN [24] supports multipath routing through multiple spanning trees, with end hosts splitting traffic over the multiple paths. However, the algorithm for computing the paths does not consider the traffic demands, and the end hosts must play a stronger role in deciding which path to use for each individual flow based on the observed performance.

7. CONCLUSION

In this paper we propose a mechanism that combines path protection and traffic engineering to enable reliable data delivery in the presence of link failures. We formalize the problem by providing several optimization-theoretic formulations that differ in the capabilities they require of the network routers. For each of the formulations, we present algorithms and heuristics that allow the network operator to find a set of optimal end-to-end paths and load balancing rules.

Our extensive simulations on the IP backbone of a tier-1 ISP and on a range of synthetic topologies demonstrate the attractive properties of our solutions. First, state-dependent splitting achieves load balancing performance close to the theoretical optimum, while state-independent splitting often offers comparable performance and a very simple setup. Second, using our solutions does not significantly increase propagation delay compared to the shortest path routing of OSPF. Finally, our solution is robust to diurnal traffic changes and a single configuration suffices to provide good performance.

In addition to failure resilience and favorable traffic engineering properties, our architecture has the potential to simplify router design and reduce operation costs for ISPs as well as operators of data centers and enterprise networks.

8. ACKNOWLEDGMENTS

We thank Adel Saleh, DARPA Program Manager, for his guidance and support of our participation in the DARPA CORONET Program, Contract N00173-08-C-2011. We also acknowledge the suggestions received from Olivier Bonaventure, Quynh Nguyen, Kostas Oikonomou, Rakesh Sinha, Robert Tarjan, Kobus van der Merwe, and Jennifer Yates.

9. REFERENCES

[1] OpenFlow Switch Consortium. `http://www.openflowswitch.org/`.

[2] JUNOS: MPLS fast reroute solutions, network operations guide, 2007.

[3] I. Avramopoulos and J. Rexford. Stealth probing: Securing IP routing through data-plane security. In *Proc. USENIX Annual Technical Conference*, June 2006.

[4] M. Caesar, M. Casado, T. Koponen, J. Rexford, and S. Shenker. Dynamic route computation considered harmful. *SIGCOMM Comput. Commun. Rev.*, Apr. 2010.

[5] E. Dinan, D. Awduche, and B. Jabbari. Analytical framework for dynamic traffic partitioning in MPLS networks. In *IEEE International Conference on Communications*, volume 3, pages 1604–1608, 2000.

[6] A. Elwalid, C. Jin, S. Low, and I. Widjaja. MATE: MPLS adaptive traffic engineering. In *Proc. INFOCOM*, volume 3, pages 1300–1309, 2001.

[7] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10(2):111–121, 1980.

[8] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE Journal on Selected Areas in Communications*, 20(4):756–767, 2002.

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

[10] I. Ganichev, B. Dai, B. Godfrey, and S. Shenker. YAMR: Yet another multipath routing protocol. *SIGCOMM Comput. Commun. Rev.*, 40(5):14–19, 2010.

[11] S. Goldberg, D. Xiao, E. Tromer, B. Barak, and J. Rexford. Path-quality monitoring in the presence of adversaries. In *Proc. ACM SIGMETRICS*, June 2008.

[12] A. Greenberg et al. VL2: A scalable and flexible data center network. In *Proc. ACM SIGCOMM*, pages 51–62, 2009.

[13] I. P. Kaminow and T. L. Koch. *The Optical Fiber Telecommunications IIIA*. Academic Press, New York, 1997.

[14] S. Kandula, D. Katabi, B. Davie, and A. Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *Proc. ACM SIGCOMM*, pages 253–264, 2005.

[15] D. Katz and D. Ward. Bidirectional forwarding detection (BFD). IETF RFC 5880, 2010.

[16] M. Kodialam and T. V. Lakshman. Dynamic routing of restorable bandwidth-guaranteed tunnels using aggregated network resource usage information. *IEEE/ACM Trans. Netw.*, 11(3):399–410, 2003.

[17] A. Kvalbein, C. Dovrolis, and C. Muthu. Multipath load-adaptive routing: Putting the emphasis on robustness and simplicity. In *IEEE ICNP*, 2009.

[18] C. M. Lagoa, H. Che, and B. A. Movsichoff. Adaptive control algorithms for decentralized optimal traffic engineering in the Internet. *IEEE/ACM Trans. Netw.*, 12(3):415–428, 2004.

[19] Y. Lee, Y. Seok, Y. Choi, and C. Kim. A constrained multipath traffic engineering scheme for MPLS networks. In *IEEE International Conference on Communications*, volume 4, pages 2431–2436, 2002.

[20] Y. Liu, D. Tipper, and P. Siripongwutikorn. Approximating optimal spare capacity allocation by successive survivable routing. *IEEE/ACM Trans. Netw.*, 13(1):198–211, 2005.

[21] A. Markopoulou et al. Characterization of failures in an operational IP backbone network. *IEEE/ACM Trans. Netw.*, 16(4):749–762, 2008.

[22] N. McKeown et al. OpenFlow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38:69–74, 2008.

[23] B. A. Movsichoff, C. M. Lagoa, and H. Che. End-to-end optimal algorithms for integrated QoS, traffic engineering, and failure recovery. *IEEE/ACM Trans. Netw.*, 15(4):813–823, 2007.

[24] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies. In *Proc. Networked Systems Design and Implementation*, Apr. 2010.

[25] N. Mysore et al. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *Proc. ACM SIGCOMM*, pages 39–50, 2009.

[26] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot. IGP link weight assignment for operational tier-1 backbones. *IEEE/ACM Trans. Netw.*, 15(4):789–802, 2007.

[27] E. Osborne and A. Simha. *Traffic Engineering with MPLS*. Cisco Press, Indianapolis, IN, 2002.

[28] P. Pan, G. Swallow, and A. Atlas. Fast reroute extensions to RSVP-TE for LSP tunnels. IETF RFC 4090, 2005.

[29] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol label switching architecture. IETF RFC 3031, 2001.

[30] H. Saito, Y. Miyao, and M. Yoshida. Traffic engineering using multiple multipoint-to-point LSPs. In *Proc. INFOCOM*, volume 2, pages 894–901, 2000.

[31] H. Saito and M. Yoshida. An optimal recovery LSP assignment scheme for MPLS fast reroute. In *International Telecommunication Network Strategy and Planning Symposium (Networks)*, pages 229–234, 2002.

[32] Y. Seok, Y. Lee, Y. Choi, and C. Kim. Dynamic constrained multipath routing for MPLS networks. In *International Conference on Computer Communications and Networks*, pages 348–353, 2001.

[33] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh. Viking: A multi-spanning-tree ethernet architecture for metropolitan area and cluster networks. In *Proc. INFOCOM*, volume 4, pages 2283 – 2294, 2004.

[34] V. Sharma and F. Hellstrand. Framework for multi-protocol label switching (MPLS)-based recovery. IETF RFC 3469, 2003.

[35] J.-P. Vasseur, M. Pickavet, and P. Demeester. *Network Recovery: Protection and Restoration of Optical, SONET-SDH, IP, and MPLS*, pages 397–422. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2004.

[36] D. Wang and G. Li. Efficient distributed bandwidth management for MPLS fast reroute. *IEEE/ACM Trans. Netw.*, 16(2):486–495, 2008.

[37] J. Wang, S. Patek, H. Wang, and J. Liebeherr. Traffic engineering with AIMD in MPLS networks. In *IEEE International Workshop on Protocols for High Speed Networks*, pages 192–210, 2002.

[38] Y. Wang et al. R3: Resilient routing reconfiguration. In *Proc. ACM SIGCOMM*, pages 291–302, 2010.

[39] D. Wendlandt, I. Avramopoulos, D. Andersen, and J. Rexford. Don't secure routing protocols, secure data delivery. In *Proc. ACM SIGCOMM Workshop on Hot Topics in Networks*, Nov. 2006.

[40] E. W. Zegura. GT-ITM: Georgia Tech internetwork topology models (software), 1996.

[41] W. Zhang, J. Tang, C. Wang, and S. de Soysa. Reliable adaptive multipath provisioning with bandwidth and differential delay constraints. In *Proc. INFOCOM*, pages 2178–2186, 2010.

APPENDIX

A. PROOFS

This Appendix shows that two problems are NP-hard:

FAILURE STATE DISTINGUISHING
INSTANCE: A directed graph $G = (V, E)$, source and destination vertices $u, v \in V$, and a sets $s \subseteq E$.
QUESTION: Is there a simple directed path P from u to v that is up if the edges in s do not fail and down if they fail?

BOUNDED PATH LOAD BALANCING
INSTANCE: A directed graph $G = (V, E)$ with a positive rational capacity c_e for each edge $e \in E$, a collection S of subsets $s \subseteq E$ of *failure states* with a rational weight w^s for each $s \in S$, a set of triples (u_d, v_d, h_d), $1 \leq d \leq k$, corresponding to *demands*, where h_d units of demand d need to be sent from source vertex $u_d \in V$ to destination vertex $v_d \in V$, an integer bound J on the number of paths that can be used between any source-destination pair, a piecewise-linear increasing cost function $\Phi(\ell)$ mapping edge loads ℓ to rationals, and an overall cost bound B.
QUESTION: Are there J (or fewer) paths between each source-destination pair such that the given demands can be assigned to the paths so that the cost (sum of $\Phi(\ell)$ over all edges and weighted failure states as described in the text) is B or less?

To prove that a problem X is NP-hard, we must show that for some known NP-hard problem Y, any instance y of Y can be transformed into an instance x of X in polynomial time, with the property that the answer for y is yes if and only if the answer for x is yes. Both our problems can be proved NP-hard by transformations from the following problem, proved NP-hard by Fortune, Hopcroft, and Wyllie [7].

DISJOINT DIRECTED PATHS
INSTANCE: A directed graph $G(V, E)$ and distinguished vertices $u_1, v_1, u_2, v_2 \in V$.
QUESTION: Are there directed paths P_1 from u_1 to v_1 and P_2 from u_2 to v_2 such that P_1 and P_2 are vertex-disjoint?

THEOREM 1. *The* FAILURE STATE DISTINGUISHING *problem is NP-hard.*

Proof. Suppose we are given an instance $G = (V, E), u_1, v_1, u_2, v_2$ of DISJOINT DIRECTED PATHS. Our constructed instance of FAILURE STATE DISTINGUISHING consists of the graph $G' = (V, E')$, where $E' = E \cup \{(v_1, u_2)\}$, with $u = u_1$, $v = v_2$, and $s = \{(v_1, u_2)\}$.

Given this choice of s, a simple directed path from u to v that is up only if the edge (v_1, u_2) is up must contain that edge. We claim that such a path exists if and only if there are vertex-disjoint directed paths P_1 from u_1 to v_1 and P_2 from u_2 to v_2. Suppose a distinguishing path P exists. Then it must consist of of three segments: a path P_1 from $u = u_1$ to v_1, the edge (v_1, u_2), and then a path P_2 from u_2 to $v = v_2$. Since it is a simple path, P_1 and P_2 must be vertex-disjoint. Conversely, if vertex-disjoint paths P_1 from u_1 to v_1 and P_2 from u_2 to v_2 exist, then the path P that concatenates P_1 followed by (v_1, u_2) followed by P_2 is our desired distinguishing path. ■

THEOREM 2. *The* BOUNDED PATH LOAD BALANCING *problem is NP-hard even if there are only two commodities ($k = 2$), only one path is allowed for each ($J = 1$), and there is only one failure state s.*

Proof. For this result we use the variant of DISJOINT DIRECTED PATHS in which we ask for edge-disjoint rather than vertex-disjoint paths. The NP-hardness of this variant is easy to prove, using a construction in which each vertex x of G is replaced by a pair of new vertices in_x and out_x connected by the edge (in_x, out_x), and each edge (x, y) of G is replaced by the edge (out_x, in_y).

Suppose we are given an instance $G = (V, E), u_1, v_1, u_2, v_2$ of the edge-disjoint variant of DISJOINT DIRECTED PATHS. Our constructed instance of BOUNDED PATH LOAD BALANCING is based on the same graph, with each edge e given capacity $c_e = 1$, with the single failure state $s = \phi$ (i.e., the state with no failures), with $w^s = 1$, and with demands represented by $(u_1, v_1, 1)$ and $(u_2, v_2, 1)$. The cost function Φ has derivative $\Phi'(\ell) = 1$, $0 \leq \ell \leq 1$, and $\Phi'(\ell) = |E| + 1$, $\ell > 1$. Our target overall cost bound is $B = |E|$.

If the desired disjoint paths exist, we can use P_1 to send the required unit of traffic from u_1 to v_1, and P_2 to send the traffic from u_2 to v_2. Since the paths are edge-disjoint, no edge will carry more than one unit of traffic, so the cost per edge used is 1, and the total number of edges used is at most $|E|$. Thus the specified cost bound $B = |E|$ is met. On the other hand, if no such pair of paths exist, then we must choose paths P_1 and P_2 that share at least one edge, which will carry two units of flow, for a cost of at least $|E| + 1$, just for that edge. Thus if there is a solution with cost $|E|$ or less, the desired disjoint paths must exist. ■

Adding more paths, failure states, or commodities cannot make the problem easier. Note, however, that this does not imply that the problem for the precise cost function Φ presented in the text is NP-hard. It does, however, mean that, assuming P $\neq$ NP, any efficient algorithm for that Φ would have to exploit the particular features of that function.