# Balancing Performance, Robustness and Flexibility in Routing Systems

Kin-Wah Kwong, Roch Guérin, *Fellow, IEEE*, Anees Shaikh, *Senior Member, IEEE*, and Shu Tao, *Member, IEEE*

***Abstract*—Modern networks face the challenging task of handling increasingly diverse traffic that is displaying a growing intolerance to disruptions. This has given rise to many initiatives, and in this paper we focus on multiple topology routing as the primary vehicle for meeting those demands. Specifically, we seek routing solutions capable of not just accommodating different performance goals, but also preserving them in the presence of disruptions. The main challenge is *computational, i.e.*, to identify among the enormous number of possible routing solutions the one that yields the best compromise between performance and robustness. This is where our principal contribution lies, as we expand the definition of critical links – a key concept in improving the efficiency of routing computation – and develop a precise methodology to efficiently converge on those solutions. Using this new methodology, we demonstrate that one can compute routing solutions that are both flexible in accommodating different performance requirements and robust in maintaining them in the presence of failures and traffic fluctuations.**

***Index Terms*—Routing, optimization, robustness, multi-topology.**

## I. Introduction

IP NETWORKS now carry a wide range of traffic with performance needs not always served well by the traditional "one-size-fits-all" design. When it comes to routing, this has triggered interest in solutions that compute paths according to different criteria as exemplified in the Multi-Topology Routing (MTR) extensions [21], [22]. In contrast to traditional IP routing, MTR allows multiple weights on each link, and hence the corresponding routings can be assigned to separate traffic classes. MTR's main benefits stem, therefore, from its ability to route traffic classes independently to meet different performance goals. This added flexibility can be used to improve service differentiation for multiple performance criteria, *e.g.*, [13], or achieve fast local rerouting after failures, *e.g.*, [1], [12]. As detailed in Section II, each of these issues is reasonably well understood in isolation. However, the growing heterogeneity of IP traffic and its increasing sensitivity to disruptions, many of which can be attributed to routing [19], make exploring if and how routing can be optimized for performance and flexibility under both normal and failure conditions an important question.

We carry out such an investigation in its most basic setting, namely that of *two* independent routings, *i.e.*, Dual Topology Routing or DTR, with one routing targeting delay and the other throughput. We assume a well-provisioned network, where link delay characteristics are the dominant performance criterion for delay-sensitive traffic, but less so for throughput-sensitive traffic. Our goal goes beyond optimizing routing decisions according to each criterion, and extends to making them *robust* to failures. In other words, performance alone is not our sole target, and we are willing to "sacrifice" some of it in exchange for robustness in the face of common disruptions, *e.g.*, single link failures[1]. That such a trade-off is both attainable and beneficial is known under standard IP routing [10], [17], [23], [24], but its extension to multiple (two) routings has to the best of our knowledge not been explored before. The issue is of importance as although DTR, and more generally MTR, offer routings capable of meeting different performance criteria, the growing reliability requirements of today's traffic also call for ensuring that these criteria are still met in the presence of failures. Whether or not this can be realized is unclear. In particular, combining optimization criteria across multiple routings can often render a solution more fragile to small changes in parameters, and therefore make it less amenable to robust optimization. The lack of prior works exploring this issue and limited a priori understanding of the existence and feasibility of practical solutions for computing robust DTR are what motivated this work.

Furthermore, the additional computation complexity introduced by having multiple interacting routings also makes it unlikely that standard, or even similar solutions as used for a single routing, can be readily applied. Computational complexity arises from two main sources, one of which is already present in standard IP routing (single routing). Specifically, upon detecting failures, IP routing adjusts its packet forwarding decisions, *i.e.*, recomputes shortest paths based on the configured link weights[2], to better redistribute traffic around the failure. The goal of "robust" routing [10], [17], [23], [24] is then to identify a *single* set of link weights that optimizes this traffic redistribution in all failure scenarios under consideration, as well as under normal operating conditions. This calls for evaluating each routing solution against all possible

Manuscript received October 27, 2009; revised February 21, 2010 and April 29, 2010. The associate editor coordinating the review of this paper and approving it for publication was G. Pavlou.

The work of K.-W. Kwong and R. Guérin was supported by NSF grant CNS-0627004. Part of this work was presented at the ACM CoNEXT 2008 conference, Madrid, Spain, Dec. 2008.

K.-W. Kwong and R. Guérin are with the Department of Electrical and Systems Engineering, University of Pennsylvania (e-mail: {kkw@seas, guerin@ee}.upenn.edu).

A. Shaikh and S. Tao are with the IBM T. J. Watson Research Center (e-mail: {aashaikh@watson, shutao@us}.ibm.com).

Digital Object Identifier 10.1109/TNSM.2010.1009.I9P0355

[1]Single link failures are among the most frequent and can substantially impact network performance [11], [19]. They are also typically short enough not to warrant reconfiguring the network.

[2]Dynamic adjustment of link weights after a failure, while possible, is usually avoided as it can further slow-down routing convergence and increase traffic disruptions.

1932-4537/10/$25.00 © 2010 IEEE

failures. In DTR, this already complex problem is compounded because routing solutions now consist of all possible *pairs* of routings. The resulting combinatorial explosion makes a direct approach infeasible even for relatively small networks.

The first contribution of this paper is, therefore, to develop a computationally efficient, yet accurate, solution. The method is based on a novel definition of *link criticality* – a key metric for assessing the importance of a link to the robustness of routing solutions. The novelty is *not* in the concept of link criticality itself. It had already been proposed and used in the context of single routing [10], [23], [24], where in spite of the lower computational complexity, a brute force approach was still impractical for all but small networks. Instead, the novelty is in how critical links are *identified.* Specifically, the challenge is not so much in realizing that only a small subset of links may indeed be critical to the robustness of a routing solution. It is in determining *which* links actually belong to that subset. This is where the true challenge lies, and link criticality alone is not a particularly useful concept unless accompanied by a specific "identification" methodology. As a matter of fact, the relative contributions of previous works that tackled this problem for single routing [10], [23], [24] were in introducing progressively more accurate identification schemes. Unfortunately, as discussed in Section IV, those existing ad-hoc methods all failed to produce consistent results when applied to DTR. Overcoming this problem called for a re-evaluation of what it meant for a link to be "critical" and for devising an explicit and systematic methodology for deriving the criticality of a link from this basic understanding. The methodology developed in this paper provides an intuitive and principled discussion of how critical links should be selected; one that was not articulated in earlier works. The methodology is applicable to both single and multiple routings and not limited to a particular traffic engineering objective. As such, it can also be used to improve or generalize existing robust single-routing solutions. This generality is where the paper's contribution to the concept of link criticality lies.

In addition to the problem of devising an effective link criticality metric, another challenge faced in a DTR setting is that links can exhibit different criticality for each routing. Characterizing the *overall* (across both routings) criticality of a link then calls for "combining" those values so as to produce a global ordering of critical links. This ordering can then be used to reduce computational complexity by focusing on a small subset consisting of only the most critical links. This is what allows us to then explore the benefits of robust DTR solutions across a broad range of network topologies and traffic patterns, and identify *when*, *why* and *how* robust DTR solutions can help. We also explore the robustness of these benefits in the face of variations in traffic patterns and against failure scenarios that differ from those originally anticipated.

The rest of the paper is structured as follows. Section II reviews related works. Section III introduces our model and problem formulation. Section IV presents the approach used for identifying critical links and evaluates its effectiveness. Section V is devoted to demonstrating the benefits of robust DTR optimization across a broad range of network topologies, traffic patterns and uncertainties. Section VI concludes the paper.

## II. Related Works

Previous related works fall in two categories: Works tackling a similar robust optimization problem in the context of single routing; and works using MTR for traffic engineering and improving network robustness by computing multiple routings that serve as backups in case of failures.

Works from the first category are rooted in earlier efforts for finding optimal link weight settings in IP networks (shortest path and destination based), *e.g.*, [8]. Fortz *et al.* [9] was the first to consider extending the problem to include robustness to changes in either network topology (link failures) or traffic patterns. The work suggested that one could often mitigate the impact of such changes by adjusting just a few link weights. Subsequent studies, *e.g.*, [10], [17], [23], [24], established the feasibility of computing a single set of link weights that worked well under both normal conditions and all single link failures. The computational complexity of the problem was also identified, which led to various versions [10], [23], [24] of the previously mentioned critical link approach. These were, however, mostly ad-hoc and did not provide a principled, systematic investigation of the critical link selection process. Specifically, Yuan [24] proposed to select critical links randomly, while Fortz *et al.* [10] suggested to identify them based on their impact on network utilization. Sridharan *et al.* [23] proposed to use fluctuations in the performance of routing solutions that emulated link failures as a means for identifying critical links. Our method is partially inspired by this last approach, but differs significantly in how it determines link criticality, in part motivated by the fact that neither it nor the methods of [10], [24] were found to work well in a DTR setting (see Section IV-C for details).

In the second category, [1], [12] recently proposed to use MTR to improve network resiliency. Each routing protects against certain failure scenarios, with routers switching from one routing to another upon detecting failures. These works neither consider MTR as a means to support different traffic classes, nor do they focus on jointly optimizing routing to preserve performance across classes in the presence of failures. The use of MTR for traffic engineering was suggested in [2], [13]. [13] demonstrated the use of MTR to improve service differentiation among different traffic classes. [2] suggested to use MTR to approximate optimal routing by dividing traffic matrix into smaller "slices", each routed on a separate topology.

Another set of tangentially related works are studies on path restoration and backup routing in MPLS networks, *e.g.*, [4]. While those works share the goal of improving network robustness to failures, the MPLS forwarding paradigm makes it an altogether different problem.

## III. Robust Optimization Model

We model the network as a directed graph $G = (V, E)$, with node set $V$ and link set $E$, and with $C_l$ denoting the capacity of link $l \in E$. The network supports two traffic classes:

delay- and throughput-sensitive. The traffic matrices for the two classes are $R_D = [r_D(s,t)]$ and $R_T = [r_T(s,t)]$, respectively, where $r_D(s,t)$ and $r_T(s,t)$ are the traffic volumes in each class for source-destination (SD) pair $(s,t) \in V \times V$.

Each link $l$ in the network is assigned two weights, $W_l^D$ and $W_l^T$, for routing delay- and throughput-sensitive traffic, respectively. Hence, two logical topologies are formed for the two traffic classes. $W := \bigcup_{l \in E} \{W_l^D, W_l^T\}$ denotes a weight setting for the network. Our goal is to find a $W$ that works well in both normal (failure-free) and all single link failure scenarios. Traffic from different classes share link resources, *e.g.*, through a common FIFO queue at each link, so that they interact on an equal footing. However, in computing routing solutions, we give *precedence to delay-sensitive traffic* because of its inelastic nature, *i.e.*, we give preference to solutions that optimize performance for delay-sensitive traffic and among those seek the ones that yield the best possible performance for throughput-sensitive traffic (see the discussion and formulation below for details on how this affects the computation of routing solutions).

The cost function for delay-sensitive traffic is based on the notion of Service Level Agreements (SLA's) [17], typically defined in the form of an end-to-end delay bound for all SD pairs, which is particularly meaningful for real-time traffic, *e.g.*, VoIP [5]. End-to-end delays are computed by summing the delays of individual links on the path of each SD pair. The delay $D_l$ on link $l$ is computed as

$$D_l := \begin{cases} p_l, & \text{if } x_l/C_l \le \mu \quad (1a) \\ \dfrac{\kappa}{C_l}\left(\dfrac{x_l}{C_l - x_l} + 1\right) + p_l, & \text{otherwise} \quad (1b) \end{cases}$$

where $\kappa$ is the average packet size, $p_l$ and $x_l$ are the propagation delay and the total traffic on link $l$, respectively[3]. The end-to-end delay $\xi_{(s,t)}$ for SD pair $(s,t)$ routed on path $P$ is then $\xi_{(s,t)} = \sum_{l \in P} D_l$. The model assumes that queueing delay is negligible compared to propagation delay when link utilization is low ($\le \mu$) [17]. Hence, at low load, delay-sensitive traffic selects paths based on propagation delay only, without discriminating among them based on their utilization. At high load ($> \mu$), $D_l$ includes queueing delay, with Eq. (1b) using an M/M/1 model to approximate the average queueing delay on link $l$.

The cost function of delay-sensitive traffic for SD pair $(s,t)$ is then related to both $\xi_{(s,t)}$ and the SLA target delay bound $\theta > 0$. Specifically, the traffic incurs a cost of 0 when its delay is below $\theta$, followed by a sharp increase as soon as the delay exceeds the SLA threshold[4]. This is expressed as

$$\Lambda_{(s,t)} := \begin{cases} 0, & \xi_{(s,t)} \le \theta \quad (2a) \\ B_1 + B_2\left(\xi_{(s,t)} - \theta\right), & \text{otherwise} \quad (2b) \end{cases}$$

where $B_1$ and $B_2$ are positive parameters that determine the SLA penalty: $B_1$ is a constant penalty incurred for any SLA violation, while $B_2$ is a penalty proportional to the delay in excess of the SLA bound. Without loss of generality, we choose $B_1 = 100$ and $B_2 = 1$. This cost function captures the financial penalty commonly associated with SLA violations, and the fact that many real-time applications exhibit a threshold-based sensitivity to delay, *e.g.*, VoIP quality is relatively insensitive to delay as long as it remains below a certain threshold, but degrades very rapidly beyond that [7].

The overall cost function for delay-sensitive traffic is then defined as the total penalty imposed because of SLA violations, *i.e.*, $\Lambda := \sum_{(s,t) \in V \times V} \Lambda_{(s,t)}$.

For throughput-sensitive traffic, we reuse the load-based cost function $f(x_l)$ of [8], where $f(x_l)$ denotes the congestion cost of link $l$ as a function of its total traffic load $x_l$. The overall cost for throughput-sensitive traffic, $\Phi$, is then the sum of all link costs, *i.e.*, $\Phi := \sum_{l \in \mathcal{L}} f(x_l)$ where $\mathcal{L}$ is the set of links carrying throughput-sensitive traffic.

Based on $\Lambda$ and $\Phi$, we define a global cost function $K := \langle \Lambda, \Phi \rangle$ as well as an "ordering" that allows us to compare values $K_1$ and $K_2$ obtained by two routing solutions. In keeping with the focus on first meeting the requirements of delay-sensitive traffic, the cost function used to compute routing solutions reflects a "lexicographic" ordering of the two cost components, namely, $K_1 = \langle \Lambda_1, \Phi_1 \rangle > K_2 = \langle \Lambda_2, \Phi_2 \rangle$, if and only if $\Lambda_1 > \Lambda_2$, or $\Lambda_1 = \Lambda_2$ and $\Phi_1 > \Phi_2$. In other words, a routing is better only if it improves delay-sensitive traffic performance, or if it keeps delay-sensitive traffic performance essentially the same but improves throughput-sensitive traffic performance. Next, given $K$ and the associated ordering, we search for weight settings $W$ that work well in both normal and failure scenarios.

This search proceeds in two phases. The first phase, called *regular optimization*, targets normal conditions and minimizes $K_{\text{normal}} := \langle \Lambda_{\text{normal}}, \Phi_{\text{normal}} \rangle$, where $\Lambda_{\text{normal}}$ and $\Phi_{\text{normal}}$ are respectively the delay- and throughput-sensitive traffic costs in the absence of failures. In other words, the first phase seeks to

$$\underset{W}{\text{minimize}} \; K_{\text{normal}} \; . \quad (3)$$

The best costs $\Lambda^*_{\text{normal}}$ and $\Phi^*_{\text{normal}}$ obtained from this first phase are then used as benchmarks when optimizing for robustness in the second phase.

The second phase, called *robust optimization*, optimizes routing against link failures. Let $\Lambda_{\text{fail},l}$ and $\Phi_{\text{fail},l}$ denote the costs of the delay- and throughput-sensitive traffic, respectively, when link $l$ fails. To make routing robust against all single link failures, we search for weight settings as follows:

$$\underset{W}{\text{minimize}} \; K_{\text{fail}} := \langle \Lambda_{\text{fail}}, \Phi_{\text{fail}} \rangle \quad (4)$$

subject to

$$\Lambda_{\text{normal}} = \Lambda^*_{\text{normal}} \quad (5)$$

$$\Phi_{\text{normal}} \le (1+\chi)\Phi^*_{\text{normal}} \quad (6)$$

where $\Lambda_{\text{fail}} := \sum_{l \in E} \Lambda_{\text{fail},l}$ and $\Phi_{\text{fail}} := \sum_{l \in E} \Phi_{\text{fail},l}$ measure the compounded costs of the delay- and throughput-sensitive traffic, respectively, over all single link failures[5].

Eq. (5) states that we are not willing to degrade the performance of delay-sensitive traffic, *i.e.*, allow SLA violations in exchange for greater robustness, because applications associated with this traffic class experience sharp quality degradations when the end-to-end delay violates the target

[3]To prevent discontinuity of $x_l/(C_l - x_l)$ when $x_l \to C_l$, this function is approximated by a linear function when $x_l/C_l \ge 0.99$.

[4]Because SLAs are typically evaluated based on averaging multiple measurements (probes), we use a rather high load threshold of $\mu = 0.95$ in Section V to ensure that performance degradations are experienced by enough measurement packets to trigger an SLA violation.

[5]Other cost objectives can be used as well.

SLA bound. On the other hand, Eq. (6) states that such a trade-off is acceptable for throughput-sensitive traffic within a range specified by $\chi \geq 0$, because this traffic class is typically associated with elastic (*e.g.*, TCP-based) applications that can tolerate some degradations.

## IV. Reducing Computational Costs

The problem of computing optimal link weights in IP networks (even with a single traffic class and without optimizing for failure resiliency) is NP-hard [8]. Finding an exact solution to Eq. (3) that optimizes DTR under normal conditions is already a computationally formidable task, because the solution space is now exponentially larger[6]. Finding a solution to Eq. (4), which considers the impact of all possible single link failures, is even harder. Hence, computationally efficient heuristics are needed to make robust DTR optimization practical. This is the goal of this section.

### A. Heuristic outline

As discussed in Section III, our approach involves two phases illustrated in Fig. 1. The first phase builds on a local search heuristic to identify a good DTR link weight setting for Eq. (3), while the second phase focuses on finding a good robust routing solution that optimizes Eq. (4).

During the search of Phase 1, both weights (one for each traffic class) on each link are randomly perturbed. A new weight setting is accepted if it results in a lower network cost $K_{\text{normal}}$. This procedure is repeated across all links during each iteration, and stops[7] when the resulting cost reductions are less than $c\%$ after $P_1$ diversifications, where a diversification consists of restarting the search from a new random weight setting whenever the cost is not improved after a certain number of iterations. Following the motivations put forth in [23] and as detailed in Section IV-D1, we leverage Phase 1a to collect statistics on the impact of each link on the cost functions. This information will be used in Phase 1c to identify critical links. As detailed in Section IV-D1, Phase 1b may also be triggered by the heuristic to generate additional statistics if the information collected in Phase 1a is insufficient as required by the critical link identification process.

Critical links are used in Phase 2 of the heuristic, which relies on the weight settings produced during Phase 1. In particular, weight settings were recorded during Phase 1 each time one was found that satisfied the constraints of Eqs. (5) and (6). Starting from those weight settings, Phase 2 performs another local search to optimize Eq. (4). The search again terminates when cost reductions from new weight perturbations are less than $c\%$ after at least $P_2$ diversifications. The weight setting $W$ that results in the smallest value for $K_{\text{fail}}$ is chosen as the final solution. The complete pseudocode for the heuristic can be found in [15].

[6]Because traffic from the two classes interact (through a common FIFO queue in the setting chosen for this paper), they affect each other's performance. Therefore, the routing choices for the two traffic classes also affect each other. As a result, each routing for one class needs to in principle be evaluated for all possible routings of the other class. Hence, the solution space grows exponentially.

[7]We implicitly assume that Phase 1 produces a good enough solution when its stopping criterion is met.

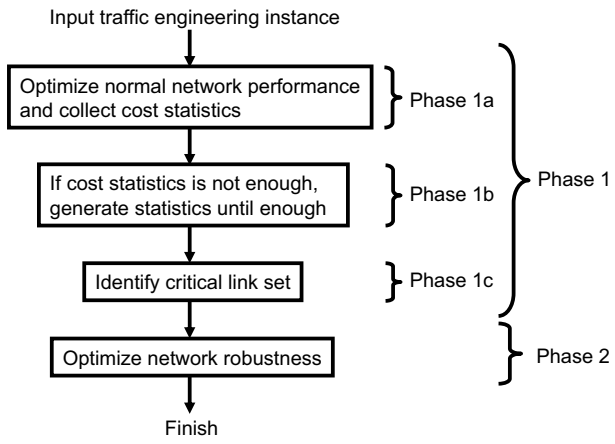



Fig. 1. The flow of the proposed heuristic.

### B. Critical links

The major complexity of the heuristic is in Phase 2, since each step involves computing network costs for all possible link failures. As in [10], [23], [24], this is what motivates the introduction of critical links, as once they have been identified, $K_{\text{fail}}$ only needs to be evaluated for the failure of these links, instead of all links. The resulting reduction in computations is then in direct proportion to $|E_c|/|E|$, where $E_c \subset E$ denotes the set of critical links. The smaller $|E_c|$, the greater the savings, and the optimization now becomes

$$\underset{W}{\text{minimize}} \ \overline{K}_{\text{fail}} := \left\langle \overline{\Lambda}_{\text{fail}}, \overline{\Phi}_{\text{fail}} \right\rangle \tag{7}$$

where $\overline{\Lambda}_{\text{fail}} := \sum_{l \in E_c} \Lambda_{\text{fail},l}$ and $\overline{\Phi}_{\text{fail}} := \sum_{l \in E_c} \Phi_{\text{fail},l}$.

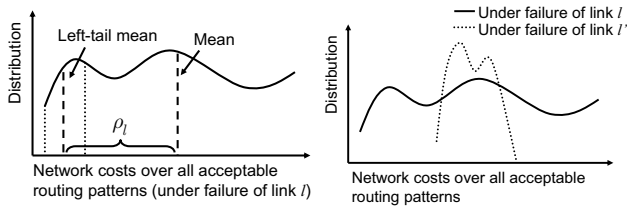
Given this, our goals are two-fold. For a given value of $|E_c|$, we want to explicitly identify which links to include in $E_c$ to minimize the resulting inaccuracy. In addition, we would like to develop an understanding of how small $|E_c|$ can be in practice, while preserving acceptable accuracy.

### C. Defining link criticality

Identifying critical links, *i.e.*, quantifying the criticality of a link, is an issue that earlier proposals [10], [23], [24] have tackled. Unfortunately, extending earlier approaches to DTR failed to generate quality solutions, and we briefly review the reasons behind this failure.

The introduction of two interacting routings as part of DTR resulted in an explosion in the size of the solution space. This made the random selection approach of [24] impractical. This was also an issue with the load-based criterion of [10], as the presence of two routings can result in a much wider range of load variations across routing solutions. In addition, link load is not the only or even the most critical performance metric for delay-sensitive traffic, which is one of the two traffic classes in our DTR model.

The reasons behind the failure of the approach of [23] in a DTR setting are more subtle. In [23], critical links are defined as links for which network costs vary wildly, when evaluated after link weight perturbations that emulate failures (assign large values to the link) during the initial phase of the optimization (our Phase 1a). The intuition is that the




(a) (b)

Fig. 2. (a) Defining link criticality. (b) Two link cost distributions.

wide cost fluctuations these links produce across failure-like weight settings, identify links for which selecting the "right" weights can make a significant difference. In translating this intuition into a procedure, [23] introduces two thresholds, which define regions of bad and good performance, and tracks how often they are crossed for each link across instances (weight settings) that emulate the failure of that link. The problem with extending this approach to DTR is that the greater range of performance variations present in DTR makes it difficult to define thresholds that are universally effective across network settings. In addition, because [23] is only concerned with a single routing, its methodology does not deal with reconciling different link criticalities, one for each routing.

Our failure to extend earlier definitions of link criticality to a DTR setting motivated a re-examination of what was behind the very notion of link criticality, and a search for a more systematic procedure to quantify it.

Specifically, we define the "criticality" of a link as the difference in the network costs produced by Phase 2 of our heuristic, with and without including the link. Thus, the question is how to estimate this difference without computing the best network cost if the link were included in the computation. For that purpose, let us *hypothetically* assume (see Fig. 2(a)) that we can construct the *distribution* of network costs, *i.e.*, $\Lambda_{\text{fail},l}$ or $\Phi_{\text{fail},l}$, under all "acceptable" routing solutions[8] when link $l$ fails. Assuming the availability of this distribution, it is then possible to infer the likely effect of including link $l$ or not in $E_c$, as we describe next.

Consider first the case where link $l$ is not in $E_c$. In such case, because the procedure of selecting link weights is oblivious to network performance under the failure of link $l$, our best estimate for the resulting network cost is simply the mean of the distribution of Fig. 2(a). In other words, the final weight setting is essentially random when it comes to the failure of link $l$. In contrast, when link $l$ is in $E_c$, the impact of its failure is explicitly incorporated in the weight selection. Hence, the selection process is biased against weight settings that generate high network costs when link $l$ fails (the r.h.s. of the curve in Fig. 2(a)), and favors weight settings that yield good performance (the l.h.s. of the curve in Fig. 2(a)). Choosing the weight setting that produces the best such performance may not be feasible, since the final solution has to be a compromise across all failure scenarios. However, it is reasonable to assume that the final choice falls somewhere in the "left-tail" of the distribution.

Based on the above observations, we propose to define link criticality as the difference between the mean value of network costs when link $l$ fails, and some estimate of the left-tail of this distribution. More formally, let $\widehat{\Lambda}_{\text{fail},l}$ and $\widehat{\Phi}_{\text{fail},l}$ denote the mean values of the distribution of Fig. 2(a) for the delay and throughput-sensitive cost functions, and $\widetilde{\Lambda}_{\text{fail},l}$ and $\widetilde{\Phi}_{\text{fail},l}$ the corresponding left-tail[9] mean values, we define the criticality of link $l$ for the two traffic classes as:

$$\rho_{\Lambda,l} := \widehat{\Lambda}_{\text{fail},l} - \widetilde{\Lambda}_{\text{fail},l} \quad (8)$$

$$\rho_{\Phi,l} := \widehat{\Phi}_{\text{fail},l} - \widetilde{\Phi}_{\text{fail},l} \quad (9)$$

The higher the value of $\rho_{\Lambda,l}$ or $\rho_{\Phi,l}$, the more critical link $l$ is. Figure 2(b) shows two representative distributions for links $l$ and $l'$. The network cost distribution for link $l'$ is relatively narrow, so that its mean and left-tail mean are close to each other. This indicates that even if we do not explicitly take link $l'$ into account during robust optimization, our selection of a routing solution, which is essentially random in its performance under the failure of link $l'$, will not perform too differently from one optimized for such a scenario. In contrast, the wider distribution for link $l$ translates into a much bigger difference between a random weight selection and one that explicitly seeks to optimize performance under the failure of link $l$.

### D. Identifying critical links

In order to use our proposed definition of link criticality, we need to return to our initial hypothesis and validate it, *i.e.*, obtain estimates of the distributions of network costs following the failure of each link across all acceptable routings (link weight settings). Obviously, this needs to be done at an acceptable computational cost. The approach we use is inspired from the methodology of [23], and extends it to reflect the larger solution space of DTR.

*1) Building cost distributions:* A naïve approach to building the cost distribution we seek would be to consider a very large set of weight settings, large enough to yield a reasonable sampling of the underlying distribution, and to proceed to then fail every link in succession and compute the resulting network costs. Clearly, this is not computationally feasible. Instead, as in [23], we take advantage of the fact that robust optimization first needs to compute the best network cost under normal conditions. This is Phase 1a in Fig. 1, which performs a local search that randomly perturbs individual link weights while seeking to improve network cost. The information gathered in this phase can be leveraged to build the distribution of network costs under link failures. This is because some weight perturbations closely resemble link failures, *i.e.*, assigning a large enough weight to a link has a similar impact on routing decisions as failing the link (the latter is equivalent to assigning it an "infinite" weight).

Specifically, cost estimates are recorded for every Phase 1a weight perturbation for which the values of *both* perturbed link weights are in the interval $[qw_{\max}, w_{\max}]$, where $0 < q < 1$ and $w_{\max}$ is the maximum allowable weight value (this emulates the failure of the link and affects both traffic classes), *and* the

[8] A routing is acceptable if it satisfies Eqs. (5) and (6).

[9] We define the left-tail as the smallest 10% costs.

network costs before such perturbation are "acceptable." We set $q = 0.7$ to realize a reasonable trade-off between closely emulating failures and ensuring a large enough number of failure-like perturbations by the end of Phase 1a. The focus on acceptable network costs is because the cost distribution we seek to estimate is in fact a *conditional* distribution, where the conditioning is that the original network cost be within a certain range of the optimal values, *i.e.*, as captured by Eqs. (5) and (6). These represent the only weight settings we are willing to consider, and therefore the one to focus on when seeking to identify critical links.

In practice, the criteria used to decide whether a perturbation is acceptable or not is slightly looser than the conditions expressed in Eqs. (5) and (6). This relaxation is to ensure that we collect a reasonable number of cost samples during Phase 1a. Specifically, samples are deemed acceptable as long as the cost of delay-sensitive traffic does not exceed the current best cost (discovered in Phase 1a so far) by more than $zB_1$ ($0 \leq z \leq 1$ and recall from Eq. (2b) that $B_1$ is a fixed cost penalty imposed on each SLA violation), and the cost of throughput-sensitive traffic is less than $(1+\chi)$ times the current best cost, where $\chi$ is the same relaxation parameter as in Eq. (6). In our experiments, we use $z = 0.5$ and $\chi = 0.2$. Pseudocode detailing the collection of cost samples during Phase 1a can be found in [15].

Even with the above relaxations of our sampling strategy, it is still possible that the number of valid samples generated in Phase 1a is insufficient to produce accurate estimates of the cost distributions of all links. This is because it requires weight perturbations that result in *both* weights being close to $w_{\max}$. Too few samples would produce inaccurate cost distributions, which could result in incorrect assessment of link criticality and compromise the quality of the final routing solution. Our approach to this problem is to add an optional phase, Phase 1b, that is carried out in case insufficient samples are collected in Phase 1a. Whether or not to trigger Phase 1b depends on the following considerations.

In Phase 1a, we continuously update the criticality of each link for the two traffic classes using the cost samples collected so far (based on Eqs. (8) and (9)), and maintain these values in two lists ($E_\Lambda$ and $E_\Phi$) sorted in descending order. $E_\Lambda$ and $E_\Phi$ are updated when on average $\tau$ (we set $\tau = 30$) additional cost samples per link are collected. After each update, we evaluate changes in the rank order of the criticality of all links as follows.

Let us take $E_\Lambda$ as an example. Between two updates at $t-1$ and $t$, we evaluate the following index for link $l$: $S_{\Lambda,l}(t) := |\text{Rank}_\Lambda(l,t) - \text{Rank}_\Lambda(l,t-1)|$ where $\text{Rank}_\Lambda(l,t)$ denotes its rank in $E_\Lambda(t)$. Then, we evaluate the overall change $\overline{S}_\Lambda$ of criticality index for delay-sensitive traffic: $\overline{S}_\Lambda := \sum_{l\in E}\gamma_l S_{\Lambda,l}(t)$ where $\gamma_l$ is a weighing factor satisfying $\sum_{l\in E}\gamma_l = 1$. We choose $\gamma_l \propto S_{\Lambda,l}(t)$ to ensure that greater weight is given to links whose criticality rank changes more. Similarly, $\overline{S}_\Phi$ for throughput-sensitive traffic is defined accordingly. Given $\overline{S}_\Lambda$ and $\overline{S}_\Phi$ as defined, we consider that link criticality estimates have converged if both $\overline{S}_\Lambda \leq e$ and $\overline{S}_\Phi \leq e$. In the evaluation of Section IV-E, we use $e = 2$ as our convergence threshold. If by the end of Phase 1a, convergence conditions are not satisfied, Phase 1b is initiated (see Fig. 1). Its purpose is to generate additional samples until convergence has been realized. Phase 1b obviously entails some computational overhead. However, as demonstrated in Section IV-E, this overhead (and that associated with the gathering of samples during Phase 1a) pales in comparison to the computational cost of Phase 2. Additional details about Phase 1b can again be found in [15].

**Algorithm 1**: Critical link identification process.

**Input**: Sorted $E_\Lambda$ and $E_\Phi$, target size $n$ of critical link set
**Result**: Critical link set $E_c$

```
n1 ← |E|, n2 ← |E|, Ec := E_{Λ,n1} ∪ E_{Φ,n2}
while |Ec| > n do
    if ρ̄_Λ(E_{Λ,n1−1}) ≥ ρ̄_Φ(E_{Φ,n2−1}) then n2 ← n2 − 1
    else n1 ← n1 − 1
end
return Ec := E_{Λ,n1} ∪ E_{Φ,n2}
```

*2) Selecting critical links:* After Phase 1a or 1b, the quantities $\rho_{\Lambda,l}$ and $\rho_{\Phi,l}$, defined in Eqs. (8) and (9) as the criticality of link $l$ for the two cost functions, have been estimated. It remains to use this information to decide which links belong to $E_c$. This depends on $\rho_{\Lambda,l}$ and $\rho_{\Phi,l}$ as well as the target size $n$ of $E_c$, and is carried out in Phase 1c. Because each link has two distinct criticality values, one for each traffic type, their orderings according to each may not be consistent. As a result, the first step of Phase 1c is to normalize link criticality values as follows:

$$\overline{\rho}_{\Lambda,l} := \rho_{\Lambda,l}\Big/\sum\nolimits_{j\in E}\widetilde{\Lambda}_{\text{fail},j}\ ,\quad \overline{\rho}_{\Phi,l} := \rho_{\Phi,l}\Big/\sum\nolimits_{j\in E}\widetilde{\Phi}_{\text{fail},j}$$

Note that the denominators in the expressions for $\overline{\rho}_{\Lambda,l}$ and $\overline{\rho}_{\Phi,l}$ represent lower bound estimates[10] of the network costs that any routing may be able to realize across all single link failures. The estimates are lower bounds as the best cost values achievable for each individual link failure may not be *jointly* realizable. Finding a routing capable of approaching those values is our goal, and doing so at an acceptable computational cost is the purpose behind identifying critical links. The quantities $\overline{\rho}_{\Lambda,l}$ and $\overline{\rho}_{\Phi,l}$ normalize the absolute criticality values for link $l$, $\rho_{\Lambda,l}$ and $\rho_{\Phi,l}$, so that it is then possible to rank links according to a global notion of criticality that accounts for both traffic classes.

Specifically, $\overline{\rho}_{\Lambda,l}$ and $\overline{\rho}_{\Phi,l}$ capture relative deviations incurred, for each traffic type, when link $l$ is not included in $E_c$. With these normalized criticality values in hand, Phase 1c proceeds to progressively eliminate links that have the least effect on the expected, normalized error of the optimization procedure. Specifically, links are first sorted in descending order of $\rho_{\Lambda,l}$ and $\rho_{\Phi,l}$ into two lists, $E_\Lambda$ and $E_\Phi$, respectively. The two lists are then used to estimate the normalized optimization errors, if only the top-$m$ links in a list are used in Phase 2 of the heuristic. These expected normalized errors are computed as follows:

$$\overline{\rho}_\Lambda(E_{\Lambda,m}) := \sum_{l\in E\setminus E_{\Lambda,m}}\overline{\rho}_{\Lambda,l}\ ,\quad \overline{\rho}_\Phi(E_{\Phi,m}) := \sum_{l\in E\setminus E_{\Phi,m}}\overline{\rho}_{\Phi,l}$$

where $E_{\Lambda,m}\subseteq E_\Lambda$ and $E_{\Phi,m}\subseteq E_\Phi$ denote the sets of the top-$m$ links, in order of criticality, in $E_\Lambda$ and $E_\Phi$. Given these

[10]Recall that $\widetilde{\Lambda}_{\text{fail},j}$ and $\widetilde{\Phi}_{\text{fail},j}$ represent estimates of the best possible network costs after the failure of link $j$.

TABLE I
CRITICAL VS. FULL SEARCH FOR DIFFERENT TOPOLOGIES.

| Topology type [# nodes, # links] | | RandTopo [30,180] | NearTopo [30,180] | PLTopo [30,162] | ISP [16,70] |
|---|---|---|---|---|---|
| Average link utilization | | 0.43 | 0.46 | 0.44 | 0.43 |
| $\beta_{\text{full}}$ | | 0.19 | 21.39 | 1.13 | 1.04 |
| $\frac{\lvert E_c\rvert}{\lvert E\rvert} = 5\%$ | $\beta_{\text{crt}}$ | 2.60 (0.82) | 25.17 (5.73) | 2.82 (0.28) | 3.32 (0.74) |
| | $\beta_{\Phi}$ (%) | 7.96 (8.71) | 24.55 (10.56) | 3.11 (2.28) | 14.22 (8.27) |
| $\frac{\lvert E_c\rvert}{\lvert E\rvert} = 10\%$ | $\beta_{\text{crt}}$ | 1.30 (0.35) | 25.31 (3.10) | 2.40 (0.13) | 3.22 (0.93) |
| | $\beta_{\Phi}$ (%) | 4.09 (0.32) | 18.59 (13.81) | 5.61 (3.07) | 18.01 (6.22) |
| $\frac{\lvert E_c\rvert}{\lvert E\rvert} = 15\%$ | $\beta_{\text{crt}}$ | 0.99 (0.15) | 22.33 (4.55) | 1.76 (0.21) | 1.99 (0.35) |
| | $\beta_{\Phi}$ (%) | 2.75 (1.84) | 19.42 (13.75) | 8.03 (4.19) | 10.08 (4.43) |

estimates, $E_c$ is initialized to $E$ and links are removed in reverse order of their effect on the normalized errors. This elimination process ends when the target size of $|E_c| = n$ is reached. The full procedure is detailed in Algorithm 1.

### E. Evaluating critical link selection

In this section, we demonstrate that the approach we have just described (denoted *critical search*) is successful in meeting our original goals, which we quantify by comparing it to a "brute-force" solution with $E_c = E$ (termed *full search*) in terms of both its accuracy and computational cost. The evaluation is carried out over a range of topologies and traffic loads (see Section V-A for details). Experiments are repeated 5 times for each network topology and traffic load configuration, and averages and standard deviations are reported.

*1) Accuracy:* To measure the accuracy of the critical search solution, we introduce the following metrics:

- $\beta_{\text{full}}, \beta_{\text{crt}}$: Average numbers of SD pairs that violate the SLA delay bound across all single link failures, under full (*full*) and critical searches (*crt*).
- $\beta_{\Phi}$ (%): Difference in network costs for throughput-sensitive traffic ($\Phi_{\text{fail}}$) between *full* and *crt*[11].

A good solution satisfies $\beta_{\text{crt}} \approx \beta_{\text{full}}$ and $\beta_{\Phi} \approx 0$.

The performance of critical search is summarized in Table I for different network topologies and different $|E_c|$ values that vary from 5% to 15% of $|E|$ (the values in brackets denote the standard deviations across the 5 runs of each experiment). The results demonstrate that critical search consistently produces a reasonable approximation of full search across different topologies, while considering only a small, albeit carefully selected, number of links. Similar results were obtained for different network sizes.

Another parameter, besides topology, that can influence the accuracy of critical search is network load. We investigate its impact in the context of a random topology with 30 nodes and 180 links. We set the maximum link utilization to 0.9 (the average utilization is 0.56, up from 0.43 in Table I). The results are as follows. $\beta_{\text{full}}$ is 1.80. $\beta_{\text{crt}}$ [resp. $\beta_{\Phi}$ (%)] is 5.77 [18.43], 2.23 [22.90], and 2.19 [24.98] when $|E_c|/|E|$ is 10%, 20%, and 25% respectively. The results illustrate that good accuracy can still be realized with only a slightly larger number of links ($\approx$ 20% vs. 10$\sim$15%) now in $E_c$. The impact of higher network load can be explained as follows. At high loads, the delay-sensitive traffic becomes more sensitive to queueing delays because of congestion. This amplifies the errors made by overlooking the cost impact of certain links. Similarly, the slope of the cost function of throughput-sensitive traffic increases with network load, so that at high loads a slight change in link load can significantly affect the cost. This in turn amplifies the magnitude of the errors incurred when omitting some links from the optimization. Both of those factors point to the need for some increase in the size of $E_c$ to maintain the accuracy of the critical search.

*2) Computational savings:* The other important aspect of critical search is the magnitude of the computational savings it yields. As discussed in Section IV-B, the bulk of the savings comes from reducing the time spent in Phase 2, and their magnitude should be approximately proportional to $1 - |E_c|/|E|$. For illustration purposes, we show results for a 30-node, 240-link RandTopo when $|E_c|/|E| = 0.1$. The average computation times (in hours) for the critical [resp. full] search in Phases 1 and 2 are 1.80 [1.32] and 4.27 [56.05] respectively. The results show that while the critical search approach slightly increases the time spent in Phase 1, this increase pales in comparison to the reduction of time spent in Phase 2. The results show a reduction in computation time from several days for the full search down to just a few hours for the critical search. These results were obtained on a 2.66 GHz Pentium Xeon machine, and all our experiments did not require more than 100 MB of memory.

In summary, in situations where computation time is not an issue, *e.g.*, small networks, access to substantial computational resources and/or loose time constraints for producing a solution, a full search is the approach of choice as it provides the highest accuracy. However, there are many settings where a full search will be either impractical or too expensive, especially because complexity grows very rapidly with network size. For example, extrapolating from the above results of computation times, a full search on a well-connected 100-node network will likely take several weeks versus about a day when using the critical link approach. Granted, those figures assume relatively low-end computational resources, but even in instances where more resources are available, both computational time and computational costs are likely to remain an issue. Network optimization may not be continuously performed in real-time, but new technologies, *e.g.*, virtualization, have considerably shortened the duration of provisioning cycles. Similarly, recent trends such as cloud computing have made computations more akin traditional utilities, where cost and consumption are closely linked. As a result, a solution that offers a tunable (through varying the size of the critical link set) trade-off between computational cost and accuracy is a useful tool.

[11]Critical search may produce a smaller cost for throughput-sensitive traffic than full search does, because of the non-linearity of the lexicographic cost function.

## V. Evaluating Robustness

The previous section addressed the feasibility of computing routing solutions capable of meeting delay and throughput requirements and maintaining them in the presence of all single link failures. Next, we turn to exploring the merits of this solution, and in particular assessing *when* it is of benefit, and how *big* those benefits are. Answering those questions calls for comparing the performance of routing solutions computed with and without taking robustness into account across different network topologies of varying sizes and carrying different traffic patterns and loads. Additionally, the choice of SLA target is also of interest, *e.g.*, to determine whether simply relaxing the target SLA can substitute for a robust routing solution.

Last but not least, the *sensitivity* of the solution to the assumptions on which it is predicated is of concern. In particular, we examine the sensitivity of robust routing solutions along two dimensions. The first one is inaccuracy in the "anticipated" traffic matrices used in computing a robust routing solution. Such inaccuracy is to be expected since traffic matrix estimation is rife with approximations, *e.g.*, [16]. The second dimension along which we assess the sensitivity of robust routing solutions is that of failure patterns. Specifically, routing solutions are computed to ensure robustness in the face of all single link failures, as such failures are more common. However, they are clearly not the only possible failure scenarios, and it is important to ensure that robustness to those failures is not at the cost of increased fragility to other types of failures. For illustration purposes, we focus on another important class of failures, namely, node failures that can arise because of software bugs or overload that affect an entire router.

In the rest of this section, we attempt to answer these questions, starting with an overview of the configurations (network topologies, traffic matrices, and other parameter settings) used in the evaluation. The results of the investigation can be summarized as follows:

- Robust optimization affords significant benefits across most network topologies, *i.e.*, minor loss in performance under normal conditions and much smaller performance degradations in the presence of failures.
  - These benefits grow as the path diversity (*i.e.*, the number of paths between nodes) offered by the network topology increases. This is because robust optimization explores additional paths besides the "shortest paths" considered by regular optimization. Those additional paths are then evaluated to determine if they afford greater robustness without meaningfully degrading performance under normal conditions. The range of this exploration is obviously bound by the number of alternate paths that are available. Hence, the benefits that robust optimization can offer, are typically in proportion to the number of paths it can explore.
- Network size and load do not significantly affect the benefits of robust optimization.
  - However, because high network loads can limit path diversity (*i.e.*, by reducing the number of paths that can accommodate delay-sensitive traffic without SLA violations after a failure), those benefits are somewhat lower at very high loads.
- Relaxing SLA delay bounds is not a substitute for robust optimization, *i.e.*, a looser SLA does not ensure greater robustness to failures.
- The benefits of using robust optimization are not overly sensitive to variations in the assumptions used to compute a solution. In particular, they remain even in the presence of variations in traffic patterns and loads, and for different types of failures, *e.g.*, link or node failures.

### A. Evaluation settings

*1) Network topologies:* Both real and synthesized topologies are used. Our real topology emulates a North American ISP backbone network of 16 nodes and 70 links. For synthesized topologies, we assume that nodes are randomly distributed in a unit square and connected using three different types of topologies:

- *RandTopo*: Random graph of given average node degree.
- *NearTopo*: Nodes connect to their closest neighbors.
- *PLTopo*: Power-law topology based on [3].

For synthesized topologies, link propagation delays are determined by the Euclidean distances between nodes and scaled proportionally to ensure a reasonable match between the target SLA bound $\theta$ and the network diameter. For the real ISP topology, link propagation delays are determined by the geographical distances between nodes. In both synthesized and real topologies, link propagation delays ranged roughly from 5ms to 20ms. Unless otherwise specified, a value of $\theta$=25ms is used, which approximates the U.S. coast-to-coast propagation delay. Link capacities were set at 500 Mbps, with different traffic patterns and intensities (see below) used to generate heterogeneous load levels.

*2) Traffic matrices:* The throughput- and delay-sensitive traffic matrices are generated using the models as in [13], where we assume that each SD pair generates delay-sensitive traffic. We assume that the total volume of delay-sensitive traffic is 30% of the total network traffic volume, but experiments with other fractions did not reveal strong sensitivity to this value. This is because both traffic classes share a common FIFO queue in the network, so that total load rather than the fraction of traffic from a given class determines queueing delays. Furthermore, as reflected in the cost function used for delay-sensitive traffic, propagation delay is the dominant component in determining its performance.

*3) Computational parameters:* Both the cost functions and the computational method of Section IV involve a number of parameters, and we briefly specify the values used in generating the results of Sections V-B to V-F. Experiments with other values produced some changes in the results, but none that significantly affected the conclusions of this investigation.

In estimating the network cost for delay-sensitive traffic, we used a packet size $\kappa = 1500$ bytes and a load threshold $\mu = 0.95$ in Eq. (1). This value of load threshold was used because we consider a backbone environment, where high link speeds make queueing delays negligible except at very high loads as reported in [20]. A similar assumption

TABLE II
NUMBER OF SLA VIOLATIONS ACROSS TOPOLOGIES.

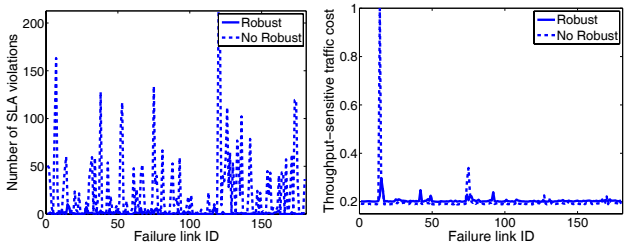
| Topology type [# nodes, # links] | | RandTopo [30,180] | NearTopo [30,180] | PLTopo [30,162] | ISP [16,70] |
|---|---|---|---|---|---|
| Average SLA violations | Robust | 0.99 (0.15) | 22.33 (4.55) | 1.76 (0.21) | 1.99 (0.35) |
| | No robust | 23.32 (8.07) | 40.17 (4.74) | 11.25 (2.05) | 4.49 (0.64) |
| Average top-10% SLA violations | Robust | 5.92 (0.70) | 121.81 (22.46) | 10.85 (1.81) | 10.93 (2.74) |
| | No robust | 139.01 (27.28) | 180.25 (25.20) | 72.58 (16.33) | 23.62 (3.17) |
| Cost degradation of throughput-sensitive traffic (%) | | 6.02 (1.45) | 18.93 (2.06) | 7.01 (1.61) | 12.35 (4.35) |

was also used in an earlier related work [17]. For example, given our setting, a 95% link load corresponds to an average queueing delay of less than 0.5ms which is about 10 times less than the smallest link propagation delay in our experiments. In allowing degradation of the performance experienced by throughput-sensitive traffic in exchange for greater robustness, we chose $\chi = 0.2$, *i.e.*, we allow a degradation of up to 20%. In Phase 1a of the optimization heuristic, the diversification interval is set to 100 iterations and the search stops when $P_1 = 20$ diversifications have all produced cost improvements below $c = 0.1\%$. In Phase 2, each diversification round starts with a weight setting close to one that already satisfies the constraints of Eqs. (5) and (6), so a smaller diversification interval of 30 iterations is used and the search terminates after $P_2 = 10$ diversifications produce cost improvements below $c = 0.1\%$. Unless otherwise specified, $|E_c|/|E| = 0.15$ was used. Each experiment was repeated 5 times and the average results are reported. In the tables, the values in the brackets denote the standard deviations in the 5 runs of each experiment.

### B. Effect of network topology

We evaluate the benefits of robust optimization across the topologies of Section V-A1. Two metrics of interest are (i) the number of SLA violations in the presence of failures (as a measure of robustness); and (ii) the impact of robust optimization on the network cost of throughput-sensitive traffic (as a measure of the cost of robust optimization). Note that implicit in our earlier choice of $\chi = 0.2$, we are willing to tolerate an increase in the network cost of throughput-sensitive traffic of 20% in exchange for robustness. The purpose of (ii) is to accurately ascertain the extent of the penalty that robust optimization imposes on throughput-sensitive traffic (recall that we are giving precedence to delay-sensitive traffic in the optimization).

The results of this investigation for metric (i) are reported in Table II, for scenarios where all topologies had an average link load around 0.43 under normal conditions. We note that the tables report the absolute number of SLA violations in each configuration. Because larger topologies (more nodes) have more SD pairs over which SLAs are measured[12], they often give rise to more SLA violations. This is in spite of the fact that a larger number of nodes typically improves path diversity, which can benefit robust optimization. The benefits from the latter can be readily assessed by dividing the number of SLA violations by the number of SD pairs to obtain a relative number of violations. From the table, we see that on average, robust optimization not only produces substantially fewer SLA violations across all failures (by factors ranging from 2 to 7 in most cases[13]), but more importantly it yields meaningful reductions when focusing on the "worst" top-10% of all failures, *i.e.*, those with the highest number of SLA violations.




(a) (b)

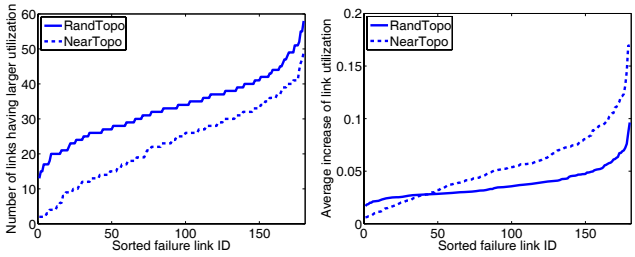
Fig. 3. Network performance with and without robust optimization.

We take a closer look at how those benefits are achieved in the case of RandTopo, for which detailed link-by-link results are presented in Fig. 3 that reports on both metrics (i) and (ii). Figure 3(a) displays the often dramatic reduction in the number of SLA violations that robust optimization affords, while Fig. 3(b) illustrates that throughput-sensitive traffic is also afforded some protection, especially during the worst failure patterns. In addition, the last row of Table II shows that although we were willing to tolerate a degradation of up to 20% in the cost of throughput-sensitive traffic under normal conditions in exchange for greater robustness, the actual degradation incurred is typically much smaller. In other words, the use of sub-optimal and potentially longer paths selected by robust optimization only had a small impact on the performance of throughput-sensitive traffic. This observation was consistent across all our experiments, so that from now on we focus on metric (i).

We turn next to NearTopo that was identified as somewhat of an outlier exhibiting smaller benefits from robust optimization than other topologies. Specifically, Table II shows a relatively large number of SLA violations even under robust optimization. This can be explained as follows: In NearTopo, nodes connect only to their nearest neighbors. Paths between pairs of nodes geographically far apart, *e.g.*, at opposite sides of the network, are not only long (in terms of hop counts), but share a small set of links in the "core" of the network. This limited path diversity means that core links are typically heavily loaded, and the associated long queueing delays can then induce SLA violations even in the absence of failures. Failures obviously make matters worse, and whenever a core

[12]The number of delay-sensitive SD pairs in a network with $|V|$ nodes is $|V|(|V|-1)$.

[13]NearTopo is somewhat of an outlier in that even if some reductions are seen, the number of SLA violations remains high even with robust optimization. We will shortly explain the reason behind this behavior.




(a) (b)

Fig. 4. Link loads after failure under robust optimization. (a) Number of links experiencing load increase. (b) Average increase in link load.

link fails, its traffic can only be redistributed on few other links that are already heavily loaded. This translates into even heavier congestion and longer queueing delays on those links, which then result in a large number of SLA violations. An obvious question is whether robust optimization would fare better, if links in the core of the network were resized to eliminate SLA violations at least under normal conditions. The resizing was done by increasing the capacity of those congested links so as to bring down their utilization below 90% under normal conditions. After performing such link resizing, the average number of SLA violations after failures decreases as expected (down to 8 when robust optimization is used and 18 when it is not). However, the marginal path diversity that is still the rule in NearTopo implies that even then the benefits of robust optimization remain limited.

The investigation of NearTopo illustrates that in general the benefits of robust optimization depend on its ability to discover and use additional paths that regular optimization would not consider, and do so without inducing severe congestion on these paths. This is not possible in NearTopo, where the limited number of routing options in the core means that both regular and robust optimizations consider essentially the same set of paths. To further illustrate the effect of path diversity on robust optimization, we compare link utilization levels in RandTopo and NearTopo after failures. Figure 4(a) shows that in RandTopo load increases after a failure are distributed over a much greater number of links than in NearTopo, Fig. 4(b) shows that the magnitudes of these increases are much smaller in RandTopo. The few large utilization increases in NearTopo are responsible for the comparatively larger number of SLA violations it experiences.

### C. Effect of network size

Network size is another factor that can affect the benefits of robust optimization, because varying the number of either nodes or links can affect the level of path diversity in the network (*i.e.*, the number of available paths between nodes). In investigating this aspect, we use the RandTopo for illustration purposes. We first vary the network size from 30 to 100 nodes, while keeping the mean node degree fixed at 5. Table III summarizes the number of SLA violations as the number of nodes varies. Next, we carry out a symmetric investigation where the number of nodes is kept constant at 30, and the number of links is increased by varying the average node degree from 4 to 8. The results are displayed in Table IV. The

TABLE III
SLA VIOLATIONS IN RANDTOPO (DIFFERENT NETWORK SIZES).

| Size | 30 nodes | | 50 nodes | | 100 nodes | |
|---|---|---|---|---|---|---|
| | R | NR | R | NR | R | NR |
| Average | 1.41 (0.18) | 5.95 (1.32) | 1.63 (0.11) | 23.35 (8.39) | 2.25 (0.23) | 30.62 (9.25) |
| Top-10% | 7.88 (1.72) | 33.20 (11.31) | 12.26 (1.14) | 177.03 (67.11) | 25.01 (2.36) | 339.10 (77.03) |

TABLE IV
SLA VIOLATIONS IN 30-NODE RANDTOPO (DIFFERENT MEAN DEGREES).

| Mean node degree | 4 | | 6 | | 8 | |
|---|---|---|---|---|---|---|
| | R | NR | R | NR | R | NR |
| Average | 1.88 (0.33) | 6.80 (1.15) | 0.99 (0.15) | 23.32 (8.07) | 0.87 (0.52) | 23.16 (5.16) |
| Top-10% | 7.83 (2.25) | 31.40 (7.91) | 5.92 (0.70) | 139.01 (27.28) | 7.44 (2.16) | 132.94 (21.92) |

average link utilization is roughly 0.43 across all topologies under normal conditions. In the tables, "R" and "NR" denote robust and regular optimizations respectively.

We find that the benefits of robust optimization persist or even increase as the network grows, whether by increasing the number of nodes or by increasing the average node degree. This is in part because larger networks typically offer greater path diversity and hence more alternate paths can be explored and leveraged by robust optimization. Furthermore, the greater network size does not preclude regular optimization from making locally bad decisions, *e.g.*, by re-routing delay-sensitive traffic over congested links in the presence of failures; hence triggering SLA violations.

### D. Effect of network load

Next, we investigate how network load affects the benefits of robust optimization. As mentioned earlier, higher link loads can reduce path diversity in the network. This is because when network load increases, queueing delay eventually becomes significant enough to reduce the number of alternate paths that can accommodate delay-sensitive traffic without SLA violations after a failure. The question is then whether enough alternate paths remain for robust optimization to discover and use, so as to offer meaningful improvements. In order to explore this issue, we take a 30-node, 180-link RandTopo as a representative example that in the absence of congestion offers a reasonable level of path diversity. We consider two levels of network load: medium and high, with maximum link utilization of 0.74 and 0.9, respectively, under normal conditions. In robust optimization for the highly-loaded network, we set $|E_c|/|E| = 0.25$ to achieve better accuracy of the *critical search*.

Figure 5(a) shows the number of SLA violations across all single link failures, with and without robust optimization. As network load increases, the higher link loads and associated higher queueing delays result in more paths with end-to-end delays at or close to the SLA bound. Thus, the lesser delay margins on many paths translate into more SLA violations after link failures, irrespective of whether robust optimization is used or not. In spite of this, we see that robust optimization still yields substantial improvements in minimizing SLA violations even at high loads. This indicates that at least in topologies with adequate path diversity, robust

TABLE V
SLA VIOLATIONS IN RANDTOPO AS A FUNCTION OF SLA BOUND.

| SLA bound (ms) | 25 | 30 | 45 | 60 | 100 |
|---|---|---|---|---|---|
| Regular optimization | | | | | |
| Average SLA violations | 23.32 (8.07) | 19.34 (9.34) | 26.75 (9.48) | 31.70 (4.29) | 33.72 (5.39) |
| Average link utilization | 0.44 | 0.46 | 0.48 | 0.49 | 0.50 |
| Average max utilization | 0.65 | 0.67 | 0.79 | 0.85 | 0.91 |
| Robust optimization | | | | | |
| Average SLA violations | 0.99 (0.15) | 0.62 (0.10) | 0.57 (0.07) | 0.33 (0.06) | 0.06 (0.05) |
| Average link utilization | 0.43 | 0.45 | 0.46 | 0.48 | 0.49 |
| Average max utilization | 0.56 | 0.60 | 0.64 | 0.65 | 0.72 |

optimization is still able to identify enough alternate paths to ensure robustness by allowing traffic to be redistributed on those paths in the presence of failures.

### *E. Effect of SLA delay constraint*

In this section, we address the question of whether and to what extent robustness is achievable simply by relaxing SLA delay bounds. In other words, is robust optimization useful only under the assumption of "tight" delay constraints? The results demonstrate that a looser SLA bound alone is not sufficient to ensure greater robustness to failures. As a matter of fact and somewhat counter-intuitively, it may make matters worse and actually strengthen the benefits of robust optimization.

We illustrate this using a 30-node, 180-link RandTopo as an example[14]. Table V shows the number of SLA violations under both regular and robust optimizations for different SLA bounds. Robust optimization consistently yields a significantly smaller number of SLA violations than regular optimization, even for loose SLA bounds. As a matter of fact, a looser SLA bound often results in *more* SLA violations under regular optimization. Both results, though not immediately intuitive, can be explained. Recall that the relative insensitivity to failures of robust optimization stems primarily from its selection of paths that are slightly sub-optimal under normal conditions, but capable of preserving performance in the presence of failures. Because of their sub-optimality, those paths are never considered by regular optimization. Hence, the benefits of robust optimization are primarily a reflection of its ability to consider a different (broader) set of paths than regular optimization. As long as this difference remains, so will these benefits. Consider now the effect of relaxing the SLA bound. A looser SLA bound means that more paths are eligible for routing delay-sensitive traffic. This in turn results in new path choices for improving the performance of throughput-sensitive traffic. However, it does little to change the fact that regular optimization will still not consider the sub-optimal paths that robust optimization does. In other words, both optimizations have more paths to choose from, but the *differences* in their choices remain.

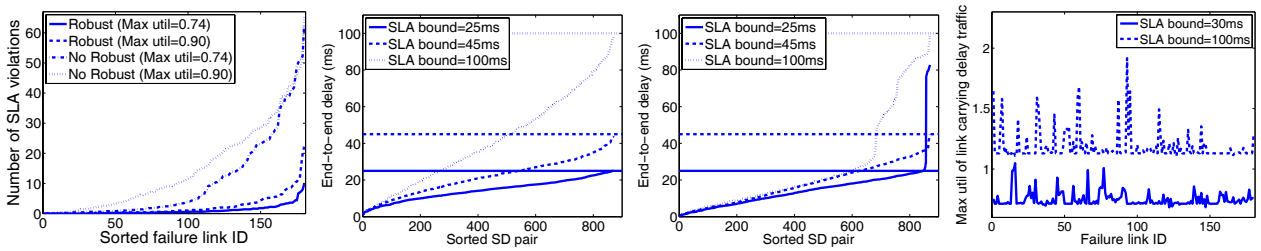
The reason behind the *increased* number of SLA violations under regular optimization in RandTopo as the SLA bound is relaxed is explored further in Fig. 5(b). The figure plots the distribution of end-to-end delays in the absence of failures for delay-sensitive traffic under regular optimization. The results suggest that as the SLA bound is relaxed, the end-to-end delays of delay-sensitive traffic increase commensurately. Hence, the number of flows close to the SLA bound and, therefore, at risk of violating it after a failure, remains roughly constant. In other words, the relaxation of the SLA bound is not used to improve the "failure-tolerance margin" of delay-sensitive flows, *i.e.*, increase the amount of additional delay they can tolerate after a failure. In addition, the ability to consider longer paths for delay-sensitive traffic to improve the performance of throughput-sensitive traffic increases link utilization (see Fig. 5(d) and Table V, where both the average link utilization of the network and the average maximum link utilization experienced by each SD pair on its path under normal conditions are reported). The average link utilization measures the global increase in link load produced by allowing delay-sensitive traffic to use longer paths, while the average maximum link utilization focuses on the effect on what is likely to be the most problematic link for each SD pair, *i.e.*, the most loaded link on that SD pair's path. This makes it more likely that after a failure, link loads, in particular on the most loaded link, will exceed the level where queueing delays become high enough to affect end-to-end delays. Hence, it is more likely that delay-sensitive flows, whose end-to-end delays are close to the SLA bound prior to a failure, will experience SLA violations after a failure. This explains the greater number of SLA violations under regular optimization as reported in Table V for looser SLA bounds. Similar results were also observed with the PLTopo and ISP topologies.

The previous results notwithstanding, there are cases where a loose SLA bound lessens the benefits of robust optimization. This typically occurs in networks with limited path diversity, where robust optimization has little potential in the first place. We illustrate this behavior using NearTopo which as discussed earlier has limited path diversity at least when it comes to crossing the network's core. By relaxing the SLA bound from 25ms to 100ms in NearTopo, the average number of SLA violations across all single link failures is decreased from 40 to 8 when regular optimization is used. In the mean time, the average link utilization is around 0.38 when relaxing the SLA bound. This observation is different from RandTopo (Table V) due to the limited path diversity in NearTopo. Because of this, relaxing the SLA bound does not trigger the exploration of longer paths (with more links), which could in turn result in higher link loads. Instead, as shown in Fig. 5(c), although end-to-end delays grow somewhat[15] as the SLA bound is relaxed, this growth is less marked than in RandTopo.

In summary, the conclusion that relaxing SLA bounds is not a substitute for robust optimization is valid, but only in topologies where robust optimization is effective in the first place.

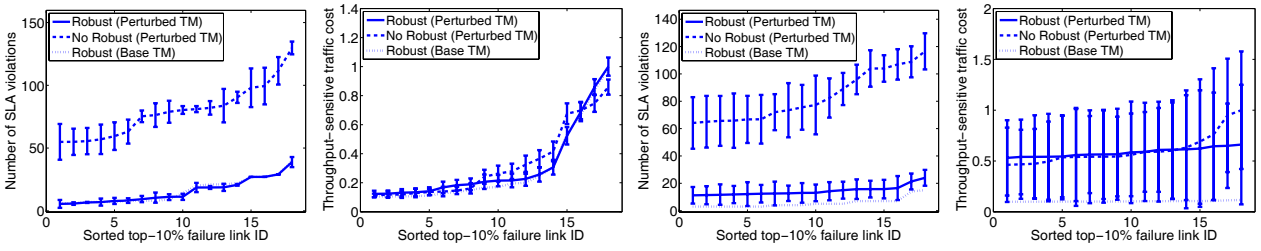
[14]Its maximum end-to-end propagation delay was fixed to 25ms.

[15]This growth is partly caused by assigning a queueing delay of zero as long as link load is below a certain threshold. This favors keeping the load of a few links below this threshold to give flows using them a better chance to meet the SLA target, even if it results in other links being more heavily loaded. As the SLA bound increases, this incentive disappears and flows are distributed more evenly across links, with more links now above the load threshold even if the average remains constant.



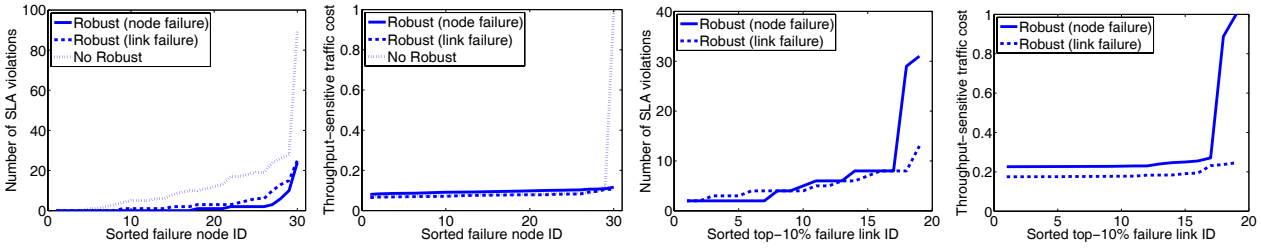

(a) (b) (c) (d)

Fig. 5. (a) SLA violations in medium- and highly-loaded networks. (b) and (c) End-to-end delays across SD pairs in the absence of failures under regular optimization in RandTopo and NearTopo respectively. (d) Max utilization of links carrying delay-sensitive traffic in RandTopo under regular optimization.




(a) (b) (c) (d)

Fig. 6. (a) and (b) Network performance under the random traffic fluctuation model. (c) and (d) Network performance under the download hot-spot model.




(a) (b) (c) (d)

Fig. 7. (a) and (b) Network performance under all single node failures. (c) and (d) Network performance under all single link failures.

### *F. Sensitivity to uncertainties*

The last aspect of our evaluation deals with the sensitivity of the routing solutions we produce to uncertainty in the inputs used to compute them. In particular, we examine their benefits when traffic patterns and loads differ from those used to compute them, and in the face of node failures as opposed to single link failures[16].

As alluded to at the beginning of Section V, traffic matrices are usually derived from combining and averaging a broad range of measurements. As a result, traffic matrices cannot be assumed to be accurate estimates of the actual traffic flowing through a network at a particular point in time. In addition to averaging and measurement inaccuracies, external factors, *e.g.*, flash crowds, BGP route changes, etc., can also contribute significant discrepancies between actual traffic and the traffic matrices used for optimization. The question is whether this affects the effectiveness of robust optimization.

[16]As we shall see, even if routing solutions computed to be robust against single link failures do not perform as well as solutions computed specifically for node failures, they still outperform routings that are oblivious to failures and simply seek to optimize performance under normal operating conditions. This outcome was also observed for other types of failure patterns, *e.g.*, multiple link failures. It establishes that robustness to single failures is not realized through an increased fragility to other failure types.

Let $R_D$ and $R_T$ denote the delay- and throughput-sensitive *base* traffic matrices used by the optimization, respectively, and $\widetilde{R}_D = [\widetilde{r}_D(s,t)]$ and $\widetilde{R}_T = [\widetilde{r}_T(s,t)]$ the traffic matrices representing the *actual* traffic carried by the network. In investigating the impact of differences between $R_D$ and $\widetilde{R}_D$, and $R_T$ and $\widetilde{R}_T$, we focus on two types of traffic uncertainties. The first emulates measurement errors and random fluctuations in traffic intensities. The second targets traffic variations caused by sporadic incidents that affect the traffic sunk or sourced by a few nodes.

To capture random fluctuations in the intensity of traffic between individual SD pairs, we rely on a Gaussian model that has been shown appropriate in modeling such estimation errors [6], [18]. This gives actual traffic intensities of the form: $\widetilde{r}_D(s,t) = r_D(s,t) + N\left(0, \varepsilon r_D\left(s,t\right)\right)$ and $\widetilde{r}_T(s,t) = r_T(s,t) + N\left(0, \varepsilon r_T\left(s,t\right)\right)$ where $N\left(0,\sigma\right)$ denotes a normally distributed random variable with zero mean and standard deviation $\sigma$. $\varepsilon$ controls the magnitude of possible traffic fluctuations. For example, with $\varepsilon = 0.2$, the actual traffic intensities can fluctuate by $\pm 40\%$ around the estimated mean value with a likelihood of about 95%.

The impact of sporadic incidents is captured by using a hot-spot model that allows traffic surges to (upload) or from

(download) a small set of (server) nodes. The hot-spot model involves selecting a small set of server nodes, assigning a number of "clients" to each one of them, and scaling the traffic intensities of the corresponding SD pairs by a factor greater than one. Specifically, in the upload scenario, assuming that client $i$ is assigned to server $j$, the corresponding traffic intensities are $\widetilde{r}_D(i,j) = \nu_{i,j} r_D(i,j)$ and $\widetilde{r}_T(i,j) = \mu_{i,j} r_T(i,j)$ for the delay- and throughput-sensitive traffic, respectively, where $\nu_{i,j} > 1$ and $\mu_{i,j} > 1$. Similarly, in the download scenario, $\widetilde{r}_D(j,i)$ and $\widetilde{r}_T(j,i)$ are set to $\nu_{j,i} r_D(j,i)$ and $\mu_{j,i} r_T(j,i)$, respectively.

We illustrate the sensitivity of routing solutions to the random fluctuation and download hot-spot models in Fig. 6 using a 30-node, 180-link RandTopo. Similar results for the upload hot-spot model were obtained and can be found in [14]. In the random fluctuation scenario, we used base traffic matrices for which robust optimization produced 90% maximum link utilization under normal conditions. $\varepsilon = 0.2$ was used to generate traffic fluctuation. In the download hot-spot scenario, we used base traffic matrices for which robust optimization produced 74% maximum link utilization under normal situations. We randomly selected 10% of the nodes as servers and 50% of nodes as clients, and $\nu_{j,i}$ and $\mu_{j,i}$ were uniformly distributed between 2 and 6, *i.e.*, the traffic volume could increase by 100-500% for those SD pairs. In each model, 100 testing instances were randomly generated. The figures focus on the top-10% worst failures to magnify possible differences. The vertical bars in the figures denote the standard deviations among the 100 testing instances. The main conclusions are that (i) the benefits of robust optimization remain even with reasonably large deviations between estimated and actual traffic matrices, *i.e.*, robust optimization still performs much better in the face of failures; (ii) Computing a routing robust to failures appears to also afford some level of robustness against unexpected traffic fluctuations, *i.e.*, routing performance is roughly equal for the estimated and actual traffic matrices.

Next, we investigate the benefits of robust routing in dealing with single node failures; a failure pattern that was not accounted for in the optimization. As discussed earlier, although node failures are less common than link failures, they can arise in the presence of malfunctions that affect an entire router, *e.g.*, overload or software bug. The investigation compares the performance of three routings in the face of all possible single node failures, where the failure of a node triggers the failure of all its links as well as the removal of all the traffic it originates. The three routings under comparison include a standard routing that seeks to optimize performance under normal conditions, our robust routing solution that considers all possible single link failures, and a robust routing solution that explicitly targets node failures. This latter solution is computed using what is essentially an "exhaustive" heuristic, which is computationally feasible in the case of node failures because of the smaller (linear) number of failure patterns under consideration.

For illustration purposes, we show the results for a 30-node, 180-link RandTopo. The traffic matrix produces roughly 80% maximum link utilization under normal conditions. We use the same set of parameters to optimize routing against all single link and all single node failures, and allow the throughput-sensitive traffic cost under normal conditions to be relaxed by at most 20% in exchange for robustness to failures. Figs. 7(a) and 7(b) summarize the results for delay- and throughput-sensitive traffic, and display network performance for each routing under the node failure scenario. As expected, although a routing optimized specifically for node failures outperforms a routing computed to handle single link failures, the latter still vastly outperforms a routing that is oblivious to failures. This is especially so for delay-sensitive traffic. This indicates that robust optimization, as defined in this paper, *i.e.*, to account for all single link failures, is not realized at the cost of greater fragility to other failure patterns and can actually mitigate the effect of a wide range of failure scenarios.

Conversely, while a routing optimized for node failures is successful at protecting against such failures, it is no replacement for a routing explicitly aimed at single link failures in terms of offering robust performance against those more common failure patterns. This is illustrated in Figs. 7(c) and 7(d), which show that a routing optimized for node failures can perform very poorly for certain link failure scenarios. This was observed across several topologies and traffic patterns. In other words, producing a routing that performs well against all single link failures cannot be realized by optimizing routing to only withstand node failures. Effectively handling such scenarios calls for using the robust optimization procedure introduced in this paper.

## VI. Conclusion

The paper explored the extent to which DTR-based solutions can offer both flexibility in supporting multiple traffic types, and robustness to failures. In carrying out this exploration, the paper develops a novel insight and methodology for identifying links that are *critical* to routing performance, thereby making the task of computing a robust routing feasible. The paper demonstrates that flexibility and robustness can be jointly realized across a broad range of topologies and traffic patterns. It also demonstrates that those benefits are relatively insensitive to variations in traffic patterns and loads, and to some extent remain present in the face of other types of failures. More importantly, they are not realized at the cost of greater fragility to those failures.

There are several possible interesting extensions to this work. For example, a probabilistic failure model can be formulated as part of a robust optimization framework, and we believe that the critical link technique developed in this paper can be extended to that model as well. Another interesting direction is to explore how to jointly design routing and network topology to maximize robustness.

## References

[1] G. Apostolopoulos, "Using multiple topologies for IP-only protection against network failures: a routing performance perspective," ICS-FORTH, Greece, Tech. Rep., 2006.

[2] S. Balon and G. Leduc, "Dividing the traffic matrix to approach optimal traffic engineering," in *Proc. IEEE ICON*, 2006.

[3] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509-512, Oct. 1999.

[4] Y. Bejerano, Y. Breitbart, A. Orda, R. Rastogi, and A. Sprintson, "Algorithms for computing QoS paths with restoration," in *Proc. IEEE INFOCOM*, 2003.

[5] C. Boutremans, G. Iannaccone, and C. Diot, "Impact of link failures on VoIP performance," in *Proc. ACM NOSSDAV*, 2002.
[6] J. Cao, D. Davis, S. V. Wiel, and B. Yu, "Time-varying network tomography: Router link data," *J. American Statistical Association*, Dec. 2000.
[7] R. G. Cole and J. H. Rosenbluth, "Voice over IP performance monitoring," *SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 2, pp. 9-24, Apr. 2001.
[8] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. IEEE INFOCOM*, 2000.
[9] ——, "Optimizing OSPF/IS-IS weights in a changing world," *IEEE J. Sel. Areas Commun.*, vol. 20, no. 4, pp. 756-767, May 2002.
[10] ——, "Robust optimization of OSPF/IS-IS weights," in *Proc. International Netw. Optimization Conf.*, 2003.
[11] G. Iannaccone, C. N. Chuah, R. Mortier, S. Bhattacharya, and C. Diot, "Analysis of link failures in an IP backbone," in *Proc. IMW*, 2002.
[12] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne, "Fast IP network recovery using multiple routing configurations," in *Proc. IEEE INFOCOM*, 2006.
[13] K.-W. Kwong, R. Guérin, A. Shaikh, and S. Tao, "Improving service differentiation in IP networks through dual topology routing," in *Proc. ACM CoNEXT*, 2007.
[14] ——, "Balancing performance, robustness and flexibility in routing systems," in *Proc. ACM CoNEXT*, 2008.
[15] ——, "Balancing performance, robustness and flexibility in routing systems," University of Pennsylvania, technical report, 2009.
[16] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: existing techniques and new directions," in *Proc. ACM SIGCOMM*, 2002.
[17] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot, "IGP link weight assignment for operational Tier-1 backbones," *IEEE/ACM Trans. Networking*, vol. 15, no. 4, pp. 789-802, Aug. 2007.
[18] A. Nucci, A. Sridharan, and N. Taft, "The problem of synthetically generating IP traffic matrices: initial recommendations," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 3, pp. 19-32, 2005.
[19] K. Papagiannaki, R. Cruz, and C. Diot, "Network performance monitoring at small time scales," in *Proc. ACM IMC*, 2003.
[20] K. Papagiannaki, S. Moon, C. Fraleigh, P. Thiran, F. Tobagi, and C. Diot, "Analysis of measured single-hop delay from an operational backbone network," in *Proc. IEEE INFOCOM*, 2002.
[21] T. Przygienda, N. Shen, and N. Sheth, "M-ISIS: multi topology (MT) routing in IS-IS," IETF RFC 5120, Feb. 2008.
[22] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault, "Multi-topology (MT) routing in OSPF," IETF RFC 4915, June 2007.
[23] A. Sridharan and R. Guérin, "Making IGP routing robust to link failures," in *Proc. Netw.*, 2005.
[24] D. Yuan, "A bicriteria optimization approach for robust OSPF routing," in *Proc. IEEE Workshop on IPOM*, 2003.

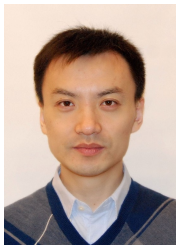
**Kin-Wah Kwong** received the B.E. degree (First Class Honors) and the M.Phil. degree in Electronic Engineering from the Hong Kong University of Science and Technology, in 2003 and 2005, respectively. He is now pursuing the Ph.D. degree at the University of Pennsylvania. His current research focus is networking, with an emphasis on understanding the performance and robustness issues in wired and wireless networks, and leveraging this understanding into engineering solutions for Internet applications and systems. He has been working on network routing, online social networks, and peer-to-peer systems. He was a co-recipient of the IEEE INFOCOM 2010 Best Paper Award.

**Roch Guérin** (F IEEE'01 / F ACM'06) received an engineer degree from ENST, Paris, France, and M.S. and Ph.D. degrees in EE from Caltech.

From 1986 till 1998 he was with the IBM T. J. Watson Research Center, Yorktown Heights, NY. In 1998 he joined the Electrical and Systems Engineering department of the University of Pennsylvania, as the Alfred Fitler Moore Professor of Telecommunications Networks. His research has spanned core networking issues, *e.g.*, routing, traffic engineering, quality-of-service, etc., as well as topics related to the use of networks by applications and the exploration of how economic factors affect the evolution of networked systems.

Dr. Guérin has been an editor for several ACM and IEEE publications, and is currently the Editor-in-Chief for the IEEE/ACM TRANSACTIONS ON NETWORKING. He has been involved in the organization of numerous ACM and IEEE sponsored activities. He was General Chair of the IEEE INFOCOM'98 conference, Technical Program co-chair of the ACM SIGCOMM'01 conference, General Chair of the ACM SIGCOMM'05 conference, and Program co-chair of the ACM CoNEXT'07 conference. He currently serves on the ACM CoNEXT steering committee. In 1994 he received an IBM Outstanding Innovation Award for his work on traffic management and the concept of equivalent bandwidth, and in 2010 he received the IEEE INFOCOM Achievement Award for "Pioneering Contributions to the Theory and Practice of QoS in Networks." He was on the Technical Advisory Board of France Telecom for two consecutive terms from 2001 to 2006, and on that of Samsung Electronics in 2003-2004. He joined the Scientific Advisory Board of Simula in 2010.

**Anees Shaikh** (M'99 / SM'09) received B.S. and M.S. degrees in Electrical Engineering in 1994 from the University of Virginia, and a Ph.D. from the University of Michigan in 1999. He currently leads a team of researchers and engineers focusing on systems and network management technologies in support of IBM's Global Technology Services business. Dr. Shaikh has contributed technologies to a number of IBM products and service offerings, such as the IBM Content Serving Utility, IBM's Global Service Delivery Centers, and the AIX 6.1 operating system. Most recently, he has been leading a number of cloud computing initiatives within IBM Research. Dr. Shaikh is active in the systems and networking research community, having co-authored nearly 40 technical articles, and served on over 20 conference and workshop technical program committees.



**Shu Tao** (S'03 / M'05) received B.S. and M.S. degrees in Electrical Engineering from Wuhan University, China, in 1997 and 2000, respectively. He received Ph.D. degree in Electrical Engineering from the University of Pennsylvania in 2005.

Dr. Tao is currently a Research Staff Member at the IBM T. J. Watson Research Center, where he works on IT systems and service management, proactive problem diagnosis, enterprise risk management and data mining. He has been granted the IBM Research Award for his work in these areas in 2006, 2007, and 2009, respectively. His other research interests include cloud computing, network and systems management, network measurement and performance evaluations.