Balancing Performance, Robustness and Flexibility in Routing Systems*

Kin-Wah Kwong, Roch Guérin
University of Pennsylvania
{kkw@seas, guerin@ee}.upenn.edu

Anees Shaikh, Shu Tao
IBM T.J. Watson Research Center
{aashaikh@watson, shutao@us}.ibm.com

ABSTRACT

Modern networks face the daunting 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.

1. 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 [18, 17]. This added flexibility can be used to improve service differentiation for multiple performance criteria, *e.g.*, [11], or improve robustness to failures, *e.g.*, [10, 8, 1]. As detailed in Section 2, 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 [16], make exploring if and how they can be jointly realized an important question.

We carry out such an investigation in its most basic instance, 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 [20, 7, 19, 14], but its extension to multiple (two) routings has to the best of our knowledge not been explored before. Such an extension is important as although DTR, and more generally MTR, significantly enhances the flexibility of IP routing, flexibility without robustness to disruptions is of limited benefits. Conversely, the sheer computational complexity of exploring the enormous routing space created by multiple (two) interacting routing schemes makes such an extension non-trivial.

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 [20, 7, 19, 14] 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

*The work of Kin-Wah Kwong and Roch Guérin was supported by NSF grant CNS-0627004.

ACM CoNEXT 2008, December 10-12, 2008, Madrid, SPAIN

[1]Single link failures are among the most frequent and can substantially impact network performance [16, 9]. They are also typically short enough not to warrant reconfiguring the network.

[2]Dynamically adjusting link weights after a failure is typically avoided to prevent even further traffic disruptions.

solution against all possible failures! In DTR, this already complex problem is compounded because routing solutions now consist of all possible *pairs* of routings. Clearly, 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. Note that the novelty we claim is *not* in the concept of link criticality itself. The concept was already proposed and used in the context of single routing [20, 7, 19, 14], where in spite of the lower computational complexity, a brute force approach was still impractical for all but small networks. The main challenge though is not in realizing that only a small subset of links may indeed be critical to the robustness of a routing solution. The challenge is in identifying *which* links actually belong to that subset. In other words, 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 [20, 7, 19, 14] were in introducing progressively more accurate identification schemes. Unfortunately, as discussed in Section 4, those existing 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. This 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.

The rest of the paper is structured as follows. Section 2 reviews related works. Section 3 introduces our model and problem formulation. Section 4 presents the approach used for identifying critical links and evaluates its effectiveness. Section 5 is devoted to demonstrating the benefits of robust DTR optimization across a broad range of network topologies and traffic patterns. Section 6 concludes the paper.

2. 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 to improve 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.*, [5]. Fortz *et al.* [6] was the first to consider extending the problem to include robustness to changes in either network topology (link failures) or traffic patterns. Their work suggested that one could often mitigate the impact of such changes by adjusting just a few link weights. Subsequent studies, *e.g.*, [7, 14, 20, 19], 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 [7, 20, 19] of the previously mentioned critical link approach. Specifically, Yuan [20] proposed to select critical links based on random sampling, while Fortz *et al.* [7] suggested to identify them based on their impact on network utilization. Sridharan *et al.* [19] 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 [7, 20] were found to work well in a DTR setting.

In the second category, [1, 8, 10] 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. Another set of tangentially related works are studies on path restoration and backup routing in MPLS networks. While those works share the goal of improving network robustness to failures, the MPLS forwarding paradigm makes it an altogether different problem.

3. ROBUST OPTIMIZATION MODEL

We model the network as a directed graph $G = (V, E)$ with node set V and link set E, and C_l denotes the capacity of link[3] $l \in E$. The network supports two traffic classes: delay- and throughput-sensitive. The traffic matrices for the two classes are $R_D = [r_D(s,t)]_{|V|\times|V|}$ and $R_T = [r_T(s,t)]_{|V|\times|V|}$, 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).

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. $W := \bigcup_{l\in E}\{W_l^D, W_l^T\}$ 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

[3]Unless otherwise specified, link means directed link.

interact on an equal footing. However, in computing routing solutions, we assume that *precedence is given to delay-sensitive traffic* (more on this below).

The cost function for delay-sensitive traffic is based on the notion of Service Level Agreements (SLA's) [14], typically defined in the form of an end-to-end delay bound for all SD pairs. 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

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

where κ is the average packet size, p_l and x_l are the propagation delay and the total traffic on link l, respectively[4]. The above model assumes that queueing delay is negligible compared to propagation delay when link utilization is low ($\leq \mu$) [14]. Hence, at low load, delay-sensitive traffic selects paths based on propagation delay only, without discriminating them by 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 utility or cost function of delay-sensitive traffic for SD pair (s,t) is then related to both the delay $\xi_{(s,t)}$ it experiences, and the SLA target delay bound $\theta > 0$. Specifically, the traffic incurs a cost of 0 when its delay is below θ, followed by a sharp increase as soon as the delay exceeds the SLA threshold. This is expressed as

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

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 [4].

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 of [5], which defines the cost $f(x_l)$ incurred by traffic on link l as follows

$$f(x_l) = \begin{cases} x_l, & \frac{x_l}{C_l} \leqslant \frac{1}{3} \quad (3a) \\ 3x_l - 2/3C_l, & \frac{1}{3} \leqslant \frac{x_l}{C_l} \leqslant \frac{2}{3} \quad (3b) \\ 10x_l - 16/3C_l, & \frac{2}{3} \leqslant \frac{x_l}{C_l} \leqslant \frac{9}{10} \quad (3c) \\ 70x_l - 178/3C_l, & \frac{9}{10} \leqslant \frac{x_l}{C_l} \leqslant 1 \quad (3d) \\ 500x_l - 1468/3C_l, & 1 \leqslant \frac{x_l}{C_l} \leqslant \frac{11}{10} \quad (3e) \\ 5000x_l - 16318/3C_l, & \frac{11}{10} \leqslant \frac{x_l}{C_l} \quad (3f) \end{cases}$$

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

where x_l denotes the total traffic on link l. The overall cost for throughput-sensitive traffic, Φ, 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 Λ and Φ, 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. Because of the precedence given to delay-sensitive traffic in computing routing solutions, the ordering we choose is "lexicographic," namely, $\langle a_1, b_1\rangle > \langle a_2, b_2\rangle$, if and only if $a_1 > a_2$, or $a_1 = a_2$ and $b_1 > b_2$. 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 Λ_{normal} and Φ_{normal} are the delay- and throughput-sensitive traffic costs, respectively, under normal conditions. In other words, it seeks to

$$\underset{W}{\text{minimize}}\ K_{\text{normal}} \quad (4)$$

The best costs $\Lambda^*_{\text{normal}}$ and Φ^*_{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:

$$\underset{W}{\text{minimize}}\ K_{\text{fail}} := \langle \Lambda_{\text{fail}}, \Phi_{\text{fail}} \rangle \quad (5)$$

subject to

$$\Lambda_{\text{normal}} = \Lambda^*_{\text{normal}} \quad (6)$$

$$\Phi_{\text{normal}} \leqslant (1+\chi)\Phi^*_{\text{normal}} \quad (7)$$

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. Eq. (6) states that we are not willing to degrade the performance of delay-sensitive traffic, *i.e.*, allow SLA violations in exchange for greater robustness, while Eq. (7) states that such a trade-off is acceptable for throughput-sensitive traffic within a range specified by $\chi \geqslant 0$. Eq. (6) reflects our earlier assumption regarding penalties for SLA violations, while Eq. (7) acknowledges the elastic nature of performance for most throughput-sensitive applications, *e.g.*, TCP.

4. REDUCING COMPUTATIONAL COSTS

The problem of computing optimal link weights in IP networks (even with a single traffic class and without optimizing for failure resilience) is NP-hard [5]. It is already a computationally formidable task to find an exact solution to Eq. (4) that optimizes DTR routing under normal conditions. It is even harder to find a solution to Eq. (5), which needs to consider the impact of all possible single link failures. Hence, to make robust DTR optimization practical, it is necessary to develop a computationally efficient heuristic.

4.1 Heuristic outline

As mentioned in the previous section and illustrated in Fig. 1, our approach involves two phases. The first phase builds on a tabu search heuristic to identify a good DTR link weight setting for Eq. (4). In each step of this search (Phase 1a in Fig. 1), both weights (one for each traffic class) of each link are randomly perturbed. The weights generated from this perturbation are accepted if they result in a lower network cost K_{normal}. This procedure is repeated across all links during each iteration, and stops[5] when the resulting cost reductions are less than $c\%$ after P_1 diversifications, where a diversification amounts to restarting the search from a new random weight setting whenever the cost is not improved after a certain number of iterations. Details about this basic building block can be found in [12]. In addition, following the motivations put forth in [19] and as detailed in Section 4.4.1, we also leverage this first phase to collect information for the identification of critical links.

Critical links are used in Phase 2 of the heuristic, which uses weight settings produced in Phase 1 and satisfying the constraints of Eqs. (6) and (7) – they can be gathered over the course of Phase 1. Starting from those weight settings, Phase 2 performs another tabu search to optimize Eq. (5). The search terminates when cost reductions from new weight perturbations are again less than $c\%$ after at least P_2 diversifications. The weight setting W that results in the smallest value for K_{fail} is chosen as the final solution.

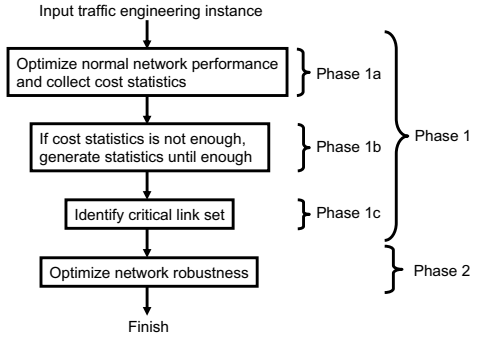


Figure 1: The flow of the proposed heuristic.

4.2 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 [20, 7, 19], this is what motivates the introduction of critical links, as once they have been identified, K_{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

$$\underset{W}{\text{minimize}}\ \overline{K}_{\text{fail}} := \left\langle \overline{\Lambda}_{\text{fail}}, \overline{\Phi}_{\text{fail}} \right\rangle \tag{8}$$

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.

4.3 Defining link criticality

How to identify critical links, or define the criticality of a link, is a question that earlier proposals [20, 7, 19] also faced. Unfortunately, extending these definitions to DTR failed to generate quality solutions, and we briefly highlight the reasons as follows.

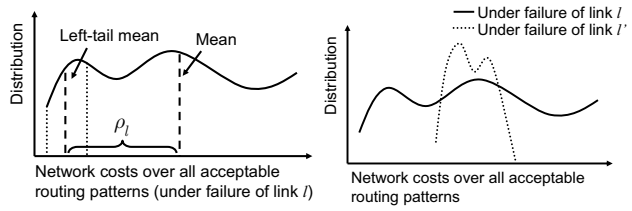
The explosion in the size of the solution space that resulted from the introduction of two routings made the random selection approach of [20] impractical. This was also an issue with the load-based criterion of [7], as the use of two routings can result in a much wider range of load variations across routing solutions. In addition, link load is not the most critical performance metric when considering the routing of delay-sensitive traffic, which is an important component in our DTR model.

The reasons behind the failure of the approach of [19] to the DTR setting are more subtle. Critical links in [19] are links for which network costs vary wildly during the initial phase of the optimization (our Phase 1a), when focusing on weight settings that emulate the failure of those links, *i.e.*, large values. The intuition is to focus on links for which selecting the right routing makes a significant difference, as indicated by the cost fluctuations they produce across failure-emulating weight settings. In translating this intuition into a procedure, [19] introduces two thresholds that define regions of bad and good performance and tracks how often they are crossed for each link in 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 made it difficult to define thresholds that were universally effective across network settings. Also the methodology of [19] is unable to deal with different link criticality for each routing. This motivated us to re-examine the notion of link criticality and seek to define a 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[6] when link l fails. Assuming the availability of

[5] We implicitly assume that a generated solution is good enough if the stopping criterion is met.

[6] A routing is acceptable if it satisfies Eqs. (6) and (7).



(a) (b)

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

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 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[7] mean values, we define the criticality of link l for the two traffic classes as:

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

$$\rho_{\Phi,l} := \widehat{\Phi}_{\text{fail},l} - \widetilde{\Phi}_{\text{fail},l} \quad (10)$$

The higher the value of $\rho_{\Lambda,l}$ or $\rho_{\Phi,l}$, the more critical link l is. Fig. 2(b) shows two representative distributions for, say, link 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.

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

4.4 Identifying critical links

In order to use our proposed definition of link criticality, we need to return to our initial hypothesis and 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 without carrying out exhaustive computations explicitly failing every link for each routing under consideration. The approach we use to construct our estimates of these distributions is inspired by the methodology of [19], and extends it to reflect the larger solution space we are dealing with.

4.4.1 Building cost distributions

As in [19], we take advantage of the fact that the optimization first needs to compute the best network cost under normal conditions. This is Phase 1a in Fig. 1, which performs a tabu 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 distributions of network costs under link failures. This can be accomplished by realizing that 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, if a Phase 1a weight perturbation results in weights for a link that are *both* in the interval $[qw_{\max}, w_{\max}]$, $0 < q < 1$, $w_{\max}$ is the maximum allowable weight value (*i.e.*, both traffic classes must be affected), *and* the network costs before such perturbation are "acceptable" (more on this below), then the cost samples from this weight perturbation can be used to build the estimated cost distributions of the link. To ensure that we gather a sufficient number of samples, our definition of "acceptable" network costs for deciding on the eligibility of a cost sample in Phase 1a is based on a slightly relaxed version of Eqs. (6) and (7). Specifically, the cost of delay-sensitive traffic should be no more than zB_1 higher than the current best cost (the lowest cost discovered in Phase 1a so far), and the cost of throughput-sensitive traffic should be no more than $(1 + \chi)$ times the current best cost. In our experiments, we let $z = 0.5$ and χ as defined in Eq. (7). We set $q = 0.7$ to realize a reasonable trade-off between closely emulating failures and ensuring the generation of a large enough number of samples by the end of Phase 1a.

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. [12] provides more details on how to decide to proceed with Phase 1b or not, but here we note that it was rarely triggered in our experiments, and when it was, the added computational overhead was marginal when compared to the overhead of Phase 2.

4.4.2 Selecting critical links

After Phase 1a or 1b, the quantities $\rho_{\Lambda,l}$ and $\rho_{\Phi,l}$, defined in Eqs. (9) and (10) 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:

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

The denominators in the above expressions represent our estimates of the best possible network costs across all single link failures, assuming it was possible to include all links in Phase 2 of the optimization. Thus, the above expressions capture relative deviations incurred, for each traffic type, when link l is not included in E_c. Note that although we have estimates for the best possible network costs under failures, we have not yet produced any routing capable of realizing them.

Once the normalized criticality values have been obtained, Phase 1c uses them to progressively eliminate from E_c 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_Λ and E_Φ, 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:

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

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_Λ and E_Φ.

Given these estimates, the next step is to remove links from E_c, starting with $E_c = E$ and until the target size of $|E_c| = n$ is reached, while minimizing the optimization error. This procedure is detailed in Algorithm 1.

Algorithm 1: Critical link identification process.

Input: Sorted E_Λ and E_Φ, target size n of critical link set
Result: Critical link set E_c

$n_1 \leftarrow |E|$, $n_2 \leftarrow |E|$, $E_c := E_{\Lambda,n_1} \cup E_{\Phi,n_2}$
while $|E_{\Lambda,n_1} \cup E_{\Phi,n_2}| > n$ **do**
  if $\overline{\rho}_\Lambda\left(E_{\Lambda,n_1-1}\right) \geqslant \overline{\rho}_\Phi\left(E_{\Phi,n_2-1}\right)$ **then**
    $n_2 \leftarrow n_2 - 1$
  else
    $n_1 \leftarrow n_1 - 1$
  end
end
return $E_c := E_{\Lambda,n_1} \cup E_{\Phi,n_2}$

4.5 Critical links-based search

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 5.1 for details). Because network performance can experience greater fluctuations when links not included in E_c are failed, each experiment is repeated 5 times, and we report averages and standard deviations.

4.5.1 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*).
- β_Φ (%): Difference in network costs for throughput-sensitive traffic (Φ_{fail}) between full and critical searches[8].

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 1 for different network topologies and different $|E_c|$ values that vary from 5% to 15% of $|E|$ (the values in the brackets denote the standard deviations in 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 observations were also found for different network sizes, and more results can be found in [12].

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 1). The results are shown in Table 2, and 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

[8]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.

Table 1: Critical vs. full search for different topologies.

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

Table 2: Critical vs. full search at high load.

β_{full}	1.80		
$\lvert E_c\rvert/\lvert E\rvert$	10%	20%	25%
β_{crt}	5.77 (2.91)	2.23 (0.94)	2.19 (0.73)
β_Φ (%)	18.43 (16.28)	22.90 (14.10)	24.98 (9.76)

increase in the size of E_c to maintain the accuracy of the critical search.

4.5.2 Computational savings

The other important aspect of critical search is the magnitude of computational savings it yields. As discussed in Section 4.2, the bulk of savings comes from reducing the time spent in Phase 2, and their magnitude should be approximately proportional to $1 - |E_c|/|E|$. This was validated across a range of topologies and is illustrated in Table 3 for a representative sample with $|E_c|/|E| = 0.1$. The table shows 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. Table 3 shows a reduction in computation time from several days for a full search down to just a few hours for a critical search. These results were obtained on a 2.66 GHz Pentium Xeon machine. The other metric of importance when assessing computational cost is memory consumption. We did not observe significant differences between the two approaches, with neither requiring more than 100 MB of memory throughout our experiments.

Table 3: Average computation times for critical vs. full search (in hours).

30-node, 240-link topologies				
Type	RandTopo		NearTopo	
Phase	1	2	1	2
Full	1.32	56.05	4.27	58.82
Crt	1.80	4.27	4.43	5.35

5. EVALUATING ROBUSTNESS

The previous section addressed the feasibility of computing routing solutions capable of both efficiently 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 it can substitute for a robust routing solution. Last but not least, the *sensitivity* of the solution to the accuracy of the "anticipated" traffic matrices is of concern. This is especially so since traffic matrix estimation is rife with potential inaccuracies, *e.g.*, [13]. As a result, it is also important to assess the extent to which the benefits of robust optimization remain in the presence of deviations between the anticipated and actual offered traffic.

In the rest of this section, we attempt to answer these questions, starting with a brief overview of the configurations (network topologies, traffic matrices, and other parameter settings) used in the evaluation. Because of space constraints, most configuration details are relegated to [12], and the focus is instead on the results of the investigation, which 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 offered by the network topology increases.
- Network size and load do not significantly affect the benefits of robust optimization.
  - However, because high network loads can limit path diversity, those benefits can be slightly 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 remain even when the actual offered traffic deviates from that used to compute the routing solution, *i.e.*, in some sense the robustness to topological changes (failures) also extends to traffic fluctuations.

5.1 Evaluation settings

5.1.1 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

Table 4: SLA violations across topologies.

Topology type [# nodes, # links]		RandTopo [30,120]	NearTopo [30,120]	PLTopo [30,162]	ISP [16,70]
Average SLA violations	Robust	1.88 (0.33)	126.09 (8.71)	1.76 (0.21)	1.99 (0.35)
	No robust	6.80 (1.15)	147.36 (19.77)	11.25 (2.05)	4.49 (0.64)
Average top-10% SLA violations	Robust	7.83 (2.25)	307.35 (21.59)	10.85 (1.81)	10.93 (2.74)
	No robust	31.40 (7.91)	379.53 (32.32)	72.58 (16.33)	23.62 (3.17)
Cost degradation of throughput-sensitive traffic (%)		3.24 (0.46)	5.89 (5.70)	7.01 (1.61)	12.35 (4.35)

and connected using three different types of topologies:

- *RandTopo*: Random graph of given average node degree.
- *NearTopo*: Nodes connect to their g closest neighbors[9].
- *PLTopo*: Power-law topology based on the preferential attachment model [2].

Link propagation delays are determined by the Euclidean (geographical for the real topology) distances between nodes and scaled to ensure a reasonable match between the target SLA bound θ and the network diameter (coast-to-coast propagation delay in the real topology). Unless otherwise specified, a value of $\theta = 25$ ms is used. Link capacities were set at 500 Mbps, with different traffic patterns and intensities (see below) used to generate heterogeneous load levels.

5.1.2 Traffic matrices

The throughput-sensitive traffic matrix is generated using a gravity model [13, 14] with three different demand levels across nodes. The delay-sensitive traffic matrix is generated using the random model of [11] where we assume that each SD pair generates delay-sensitive traffic. The total volume of delay-sensitive traffic is specified as a fraction $0 < f < 1$ of the total network traffic volume. We use $f = 30\%$ in the results we report on, but experiments with other values did not reveal strong sensitivity to this value.

5.1.3 Computational parameters

Both the cost functions and the computational method of Section 4 involve a number of parameters, and we briefly specify the values used in generating the results of Sections 5.2 to 5.6. 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 reflects a backbone environment, where high link speeds make queueing delays negligible except at very high loads. 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. (6) and (7), so a smaller diversification interval of 30 iterations is used and the search terminates after $P_2 = 10$ diversifications produce cost improvements below 0.1%. Unless otherwise specified, $|E_c|/|E| = 0.15$ was used. Traffic is split evenly over equal-cost paths. 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.

[9]A value g (> 1) is chosen such that the total number of links in the network matches a target value.

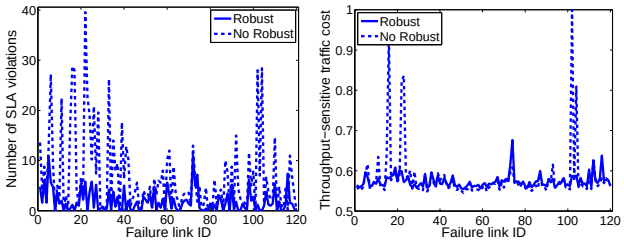
5.2 Effect of network topology

We evaluate the benefits of robust optimization on topologies introduced in Section 5.1.1. 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 ascertain whether throughput-sensitive traffic is incurring any additional penalty from robust optimization (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 4, for scenarios where all topologies had an average link load around 0.42 under normal conditions. 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[10]), but more importantly it yields drastic reductions when focusing on the "worst" top-10% of all failures, *i.e.*, those with the highest number of SLA violations.

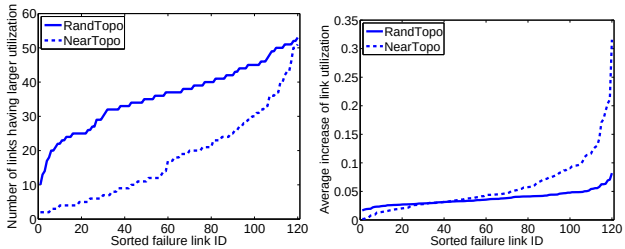
We take a closer look at how those benefits are achieved in the case of RandTopo, for which detailed link-by-link results are presented in Fig. 3 that reports on both metrics (i) and (ii). Fig. 3(a) displays the often dramatic reduction in the number of SLA violations that robust optimization affords, 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 4 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

[10]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) Number of SLA violations (b) Throughput-sensitive traffic cost

Figure 3: Network performance with and without robust optimization.



(a) (b)

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

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 now turn to the NearTopo topology that was identified as somewhat of an outlier exhibiting smaller benefits from robust optimization than other topologies. Specifically, Table 4 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 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. After performing such link resizing [12], the average number of SLA violations after failures decreases as expected (down to 18 when robust optimization is used and 38 when it is not). However, the limited path diversity that is still the rule in NearTopo implies that even then those benefits 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. Fig. 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.

5.3 Effect of network size

Network size is another factor that can affect the benefits of robust optimization. In investigating this possibility we present RandTopo as an example. The network size is varied from 30 to 50 nodes, while the mean node degree is fixed at 5. The average link utilization is roughly 0.43 across all topologies under normal conditions. Table 5 summarizes the number of SLA violations as the network size increases.

Table 5: Average number of SLA violations in RandTopo as a function of network size. ("R" and "NR" denote robust and regular optimizations, respectively).

Size	30 nodes		40 nodes		50 nodes	
	R	NR	R	NR	R	NR
Avg	2.70 (0.21)	13.80 (4.18)	0.92 (0.23)	34.56 (5.02)	1.63 (0.11)	23.35 (8.39)
Top-10%	16.58 (1.17)	84.05 (22.15)	6.52 (2.52)	200.25 (24.62)	12.26 (1.14)	177.03 (67.11)

We find that the benefits of robust optimization persist or even increase as the network grows. This is in part because larger networks typically offer greater path diversity, which robust optimization can leverage. Furthermore, the greater network size does not preclude regular optimization from making locally bad decisions, which re-route delay-sensitive traffic over congested links in the presence of failure, hence triggering SLA violations. Similar results were obtained when growing the network by adding links rather than nodes, and the details can be found in [12].

5.4 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, in that fewer alternate paths may be able to accommodate an increase in

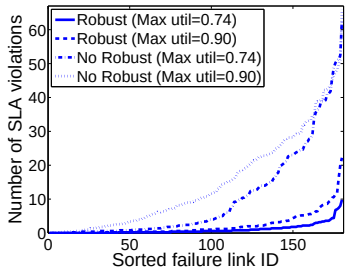


Figure 5: SLA violations in medium- and highly-loaded networks.

load, and therefore accessible to robust optimization to improve robustness. 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 it offers a reasonable level of path diversity. We consider two levels of network load: medium and high, with maximum link utilizations 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*.

Fig. 5 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 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.

5.5 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, it may make matters worse and actually strengthen the benefits of robust optimization.

We illustrate this using a 30-node, 120-link RandTopo as an example[11]. Table 6 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

[11]Its maximum end-to-end propagation delay was set to 25ms.

Table 6: Average number of SLA violations in RandTopo as a function of SLA bound under regular and robust optimizations.

SLA bound (ms)	25	30	45	60	100
Regular optimization					
Avg # SLA violations	6.80 (1.15)	4.51 (2.29)	16.22 (9.80)	23.94 (10.19)	30.24 (15.03)
Avg link util	0.42	0.44	0.46	0.48	0.50
Robust optimization					
Avg # SLA violations	1.88 (0.33)	0.49 (0.07)	0.20 (0.02)	<0.01 (<0.01)	0 (0)
Avg link util	0.42	0.43	0.45	0.46	0.48

optimization, even as the SLA bound becomes looser. As a matter of fact, a looser SLA bound often results in *more* SLA violations under regular optimization. Both results are intuitive, though not obvious. 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.

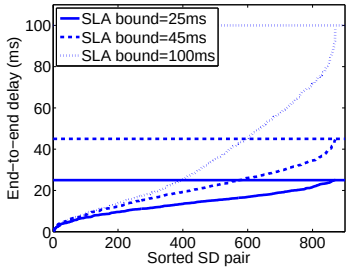


Figure 6: End-to-end delay distribution across SD pairs in the absence of failures under regular optimization.

The reason behind the *increased* number of SLA violations under regular optimization in RandTopo as the SLA bound is relaxed is itself explored further in Fig. 6. 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 flexibility to consider longer paths for delay-sensitive traffic to improve the performance of throughput-sensitive traffic also results in higher link utilization (see Table 6 and Fig. 7). This makes it more likely that after a failure, link loads will increase to a 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 6 for looser SLA bounds. Similar results were also observed with the PLTopo and ISP topologies [12].

The previous results notwithstanding, there are instances where a loose SLA bound lessens the benefits of robust optimization. This typically occurs in topologies with limited path diversity, where robust optimization has little potential in the first place[12]. In other words, the conclusion that relaxing SLA bounds is no replacement for robust optimization remains valid across topologies where it is effective.

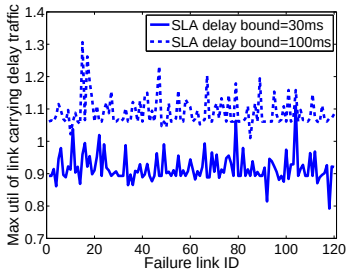


Figure 7: Maximum utilization of links carrying delay-sensitive traffic in RandTopo under regular optimization.

5.6 Sensitivity to uncertain traffic matrices

The last aspect of our evaluation deals with the sensitivity of the routing solutions we produce to errors/uncertainties in the traffic matrices used to compute them. As alluded to at the beginning of Section 5, 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.

Let R_D and R_T denote the delay- and throughput-sensitive *base* traffic matrices used by the optimization, respectively, and $\widetilde{R}_D = [\widetilde{r}_D(s,t)]_{|V|\times|V|}$ and $\widetilde{R}_T = [\widetilde{r}_T(s,t)]_{|V|\times|V|}$ 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 [3, 15]. This gives actual traffic intensities of the form

$$\widetilde{r}_D(s,t) = r_D(s,t) + N\left(0, \varepsilon r_D\left[s,t\right]\right) \quad (11)$$

$$\widetilde{r}_T(s,t) = r_T(s,t) + N\left(0, \varepsilon r_T\left[s,t\right]\right) \quad (12)$$

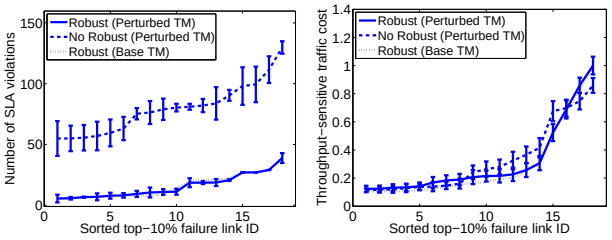
where $N(0,\sigma)$ denotes a normally distributed random variable with zero mean and standard deviation σ. ε controls the magnitude of possible traffic fluctuations. For example, with $\varepsilon = 0.2$, the actual traffic intensities can fluctuate by $\pm 40\%$ around the estimated mean value with a likelihood of about 95%. Conversely, 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 $\widetilde{r}_D(i,j) = \nu_{i,j} r_D\left[i,j\right]$ and $\widetilde{r}_T(i,j) = \mu_{i,j} r_T\left[i,j\right]$ for the delay- and throughput-sensitive traffic, respectively, where $\nu_{i,j} > 1$ and $\mu_{i,j} > 1$. Similar symmetric expressions hold for the download case.

The results of our investigation on the impact of traffic variations from the random and hot-spot models are reported in Figs. 8 and 9, respectively, for a 30-node, 180-link RandTopo. In each model, 100 testing instances were randomly generated. Details on the exact simulation settings can again be found in [12]. In the case of random fluctuations, $\varepsilon = 0.2$ was used. For hot-spots, we randomly selected 10% of the nodes as servers and 50% of nodes as clients, and $\nu_{i,j}$ and $\mu_{i,j}$ were uniformly distributed between 2 and 6, *i.e.*, the traffic volume could increase by 100-500% for those SD pairs. 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.

6. CONCLUSION

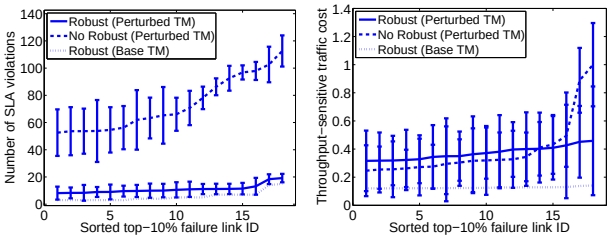
The paper explores the extent to which DTR-based routing solutions can offer both flexibility in supporting mul-

[12]See [12] for an illustration of this behavior on NearTopo.

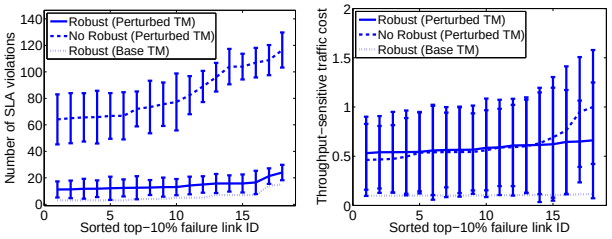


(a) SLA violation (b) Throughput-sensitive traffic cost

Figure 8: Impact of traffic uncertainty: random fluctuation scenario.



(a) SLA violation, upload hot-spot (b) Throughput-sensitive traffic cost, upload hot-spot



(c) SLA violation, download hot-spot (d) Throughput-sensitive traffic cost, download hot-spot

Figure 9: Impact of traffic uncertainty: hot-spot scenario. (a) and (b): Upload. (c) and (d): Download.

tiple 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 remain even in the presence of uncertainty in the traffic estimates used when computing routing solutions.

7. REFERENCES

[1] G. Apostolopoulos. Using multiple topologies for IP-only protection against network failures: A routing performance perspective. Technical report, ICS-FORTH, Greece, 2006.

[2] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, October 1999.

[3] J. Cao, D. Davis, S. V. Wiel, and B. Yu. Time-varying network tomography: Router link data. *Journal of the American Statistical Association*, Dec 2000.

[4] R. G. Cole and J. H. Rosenbluth. Voice over IP performance monitoring. *SIGCOMM Comput. Commun. Rev.*, 31(2):9–24, April 2001.

[5] B. Fortz and M. Thorup. Internet traffic engineering by optimizing OSPF weights. In *IEEE INFOCOM*, 2000.

[6] B. Fortz and M. Thorup. Optimizing OSPF/IS-IS weights in a changing world. *IEEE JSAC*, May 2002.

[7] B. Fortz and M. Thorup. Robust optimization of OSPF/IS-IS weights. In *International Network Optimization Conference*, 2003.

[8] S. Gjessing and O. Norway. Implementation of two resilience mechanisms using multi topology routing and stub routers. In *Proc. AICT-ICIW*, 2006.

[9] G. Iannaccone, C. N. Chuah, R. Mortier, S. Bhattacharya, and C. Diot. Analysis of link failures in an IP backbone. In *IMW*, 2002.

[10] A. Kvalbein, A. F. Hansen, T. Cicic, S. Gjessing, and O. Lysne. Fast IP network recovery using multiple routing configurations. In *IEEE INFOCOM*, 2006.

[11] K.-W. Kwong, R. Guérin, A. Shaikh, and S. Tao. Improving service differentiation in IP networks through dual topology routing. In *ACM CoNEXT*, 2007.

[12] K.-W. Kwong, R. Guérin, A. Shaikh, and S. Tao. Balancing performance, robustness and flexibility in routing systems. Technical report, University of Pennsylvania, 2008.

[13] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. In *ACM SIGCOMM*, 2002.

[14] A. Nucci, S. Bhattacharyya, N. Taft, and C. Diot. IGP link weight assignment for operational Tier-1 backbones. *IEEE/ACM ToN*, August 2007.

[15] A. Nucci, A. Sridharan, and N. Taft. The problem of synthetically generating IP traffic matrices: Initial recommendations. *SIGCOMM Comput. Commun. Rev.*, 35(3):19–32, 2005.

[16] K. Papagiannaki, R. Cruz, and C. Diot. Network performance monitoring at small time scales. In *ACM IMC*, 2003.

[17] T. Przygienda, N. Shen, and N. Sheth. M-ISIS: Multi topology (MT) routing in IS-IS. IETF RFC 5120, February 2008.

[18] P. Psenak, S. Mirtorabi, A. Roy, L. Nguyen, and P. Pillay-Esnault. Multi-topology (MT) routing in OSPF. IETF RFC 4915, June 2007.

[19] A. Sridharan and R. Guérin. Making IGP routing robust to link failures. In *Networking*, 2005.

[20] D. Yuan. A bicriteria optimization approach for robust OSPF routing. In *IEEE Workshop on IPOM*, 2003.