ENSURING NETWORK DESIGNS MEET PERFORMANCE REQUIREMENTS UNDER FAILURES

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Yiyang Chang

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

August 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Sanjay G. Rao, Chair

School of Electrical and Computer Engineering

Dr. T. N. Vijaykumar

School of Electrical and Computer Engineering

Dr. Xiaojun Lin

School of Electrical and Computer Engineering

Dr. Mohit Tawarmalani

Krannert School of Management

Approved by:

Dr. Dimitrios Peroulis

Head of the School Graduate Program

For my parents and my spouse.

ACKNOWLEDGMENTS

I would like to thank my advisor, Prof. Sanjay Rao, who has been a great mentor and a sincere friend. I am indebted to Prof. Rao for the helpful guidance he gave me and the opportunities he created to help me grow my research skills. I have learned a lot from Prof. Rao, not only the way to conduct first-class research, but also the way to pursue success and to handle failure.

I am grateful to my Committee members, Prof. Mohit Tawarmalani, Prof. T. N. Vijaykumar, and Prof. Xiaojun Lin, for the valuable advice and feedback they gave on my research and dissertation.

I am thankful to my research collaborators, Ashish Chandra, Dr. Mohammad Hajjat, Dr. Jahangir Hasan, Chuan Jiang, Ruiqi Liu, Prof. T. S. Eugene Ng, Prof. Gustavo Petri, Ashkan Rezaei, Prof. Tiark Rompf, Prof. Mohit Tawarmalani, Prof. Balajee Vamanan, and Prof. T. N. Vijaykumar, for their hard work and contributions to my research.

I am fortunate to have worked with my colleagues at Internet Systems Lab: Ashiwan, Chuan, Ehab, Harsh, Mohammad, Russ, Shankar, Srivats, Sruthi, Yun, Zaiwei, and Zisheng, for those enlightening discussions and kind support.

Last but not least, I would like to thank School of Electrical and Computer Engineering, Purdue University for funding me as a teaching assistant; I would like to thank Graduate School of Purdue University for awarding me Bilsland Dissertation Fellowship; and I would like to thank NSF for funding my research.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Chang, Yiyang Ph.D., Purdue University, August 2019. Ensuring Network Designs Meet Performance Requirements under Failures. Major Professor: Sanjay G. Rao.

With the prevalence of web and cloud-based services, there is an ever growing requirement on the underlying network infrastructure to ensure that business critical traffic is continually serviced with acceptable performance. Networks must meet their performance requirements under failures. The global scale of cloud provider networks and the rapid evolution of these networks imply that failures are the norm in production networks today. Unplanned downtime can cost billions of dollars, and cause catastrophic consequences. The thesis is motivated by these challenges and aims to provide a principled solution to certifying network performance under failures. Network performance certification is complicated, due to both the variety of ways a network can fail, and the rich ways a network can respond to failures. The key contributions of this thesis are: (i) a general framework for robustly certifying the worst-case performance of a network across a given set of uncertain scenarios. A key novelty is that the framework models flexible network response enabled by recent emerging trends such as Software-Defined Networking; (ii) a toolkit which automates the key steps needed in robust certification making it suitable for use by a network architect, and which enables experimentation on a wide range of robust certification of practical interest; (iii) Slice, a general framework which efficiently classifies failure scenarios based on whether network performance is acceptable for those scenarios, and which allows reasoning about performance requirements that must be met over a given percentage of scenarios. We also show applications of our frameworks in synthesizing designs that are guaranteed to meet a performance goal over all or a desired

percentage of a given set of scenarios. The thesis focuses on wide-area networks, but the approaches apply to data-center networks as well.

1. INTRODUCTION

With the advent of online and cloud-based services, society has become critically dependent on the Internet for all its needs. The Internet has seen a traffic growth of more than five orders of magnitude over the past few decades, and the trend is expected to continue [1]. Not only is the Internet traffic increasing, but also the expectation on network performance. For example, a recent paper from Google [2] indicates that not only has traffic increased by 100x over the last five years, but also bandwidth requirements must be met 99.99% of the time compared to 99% in the past. The dramatic growth of the Internet traffic and the increasing pressure to reduce costs have motivated both online service companies [3–5], and more recently, large ISPs [6, 7] to operate networks closer to capacity, in contrast to the 2x–3x over-provisioned networks in the past. Networks must achieve such performance goals under uncertainty. Failures of network components are routine [8–12], which can be caused by software and hardware bugs, configuration errors, and natural disasters (e.g., Hurricane Sandy). Network traffic patterns can also change significantly.

We refer to the task of certifying whether a network design meets the performance goal under uncertainty as *network performance certification*. This thesis solves network performance certification under flexible network response, which is enabled by Software-Defined Networks (SDNs, Figure 1.1), which can route traffic using global network-wide views, and with priorities [3, 4]. SDNs bring both challenges and opportunities: SDNs allow network response mechanisms to be expressed as well-defined optimization problems, but it is challenging to take the flexibility into account when certifying networks, as we will see later. To the best of the author's knowledge, this thesis is the first to propose a principled solution to network performance certification in an SDN context.

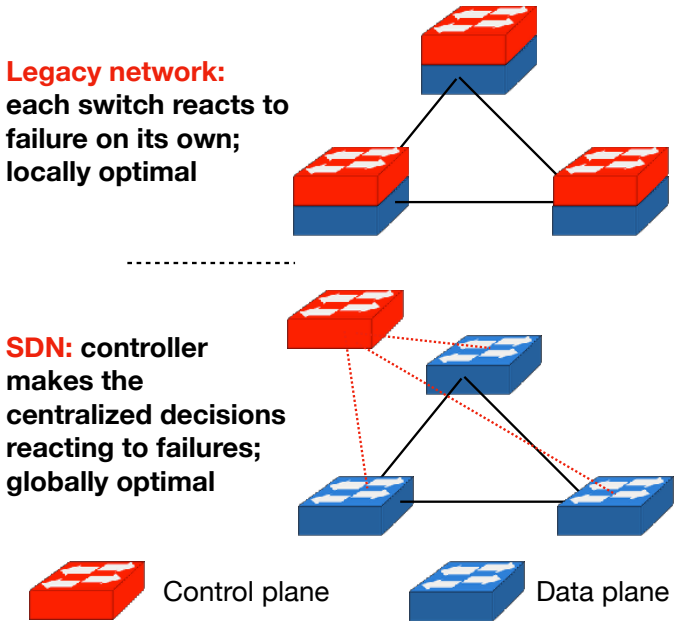


Fig. 1.1.: SDN vs. legacy network.

Unfortunately, many existing approaches to designing networks for failures (i) only focus on availability [13–17] resulting in poor performance on failures [12, 18]; (ii) only consider a small number of failure states [19–25], and do not scale as the number of possible failure states increases; or (iii) rely on ad-hoc simulation-based testing [26, 27]. This thesis is distinguished by the focus on performance, designing for multiple concurrent failures and designs with provable performance guarantees for known failures.

The rest of this chapter illustrates the background in the research area of designing networks resilient to failures in §1.1, recent work and their limitations in §1.2, the contributions of this thesis in §1.3, and the main results in §1.4. Finally, §1.5 provides a road map of the whole thesis.

1.1 Background

Many mechanisms focus on restoring or verifying connectivity but do not address performance (e.g., congestion, packet losses, and delays) under failures. Researchers have long recognized the need to recover quickly from network failures and have proposed to do so by moving traffic away from a failed network device or link. For instance, in Multiprotocol Label Switching (MPLS) settings, two classes of recovery mechanisms have been explored [13]. In a *link-based protection* scheme, upon the failure of a link l, traffic previously on l is rerouted along pre-computed detour paths that do not include l. In a *path-based protection* scheme, backup paths are calculated in advance for traffic from each source s to each destination t, and when a particular path fails (e.g., an underlying link or node fails), the traffic is diverted to those backup paths. Fast recovery mechanisms have also been studied in IP settings [14–17]. Nevertheless, fast recovery schemes are insufficient to prevent congestion (and consequently, packet losses and delays) under failures – e.g., recent work from Microsoft [18] shows that failures can frequently lead to links getting $10\% - 20\%$ more traffic than their capacity which can negatively impact the performance of demanding applications such as online retail, Web search, and video streaming.

Many works on resilient network design only consider a small number of failure states (e.g., single-link or node failures). A comprehensive survey of early work in the area is provided in [19], while recent representative work includes [20–25]. The approach in this literature is to arrive at resilient designs by explicitly enumerating all failure scenarios in the formulation of an optimization problem, which, for example, may determine how to provision spare capacity to handle those failures. While such an approach is tractable when the number of possible failure states is small, it does not scale well for the challenging performance requirements of modern networks that may experience multiple failures simultaneously [18, 28]. The intractability arises from the fact that naively enumerating failure states leads to a formulation that simultaneously models exponentially many routing problems, one for each failure state (e.g., for a

300 link network, 44,850 routing problems must be simultaneously modeled even for handling all two failure scenarios).

Simulation-based testing is inadequate for designing resilient networks. The state-of-practice in checking whether networks conform to performance requirements involves simulation-based testing [26, 27]. Unfortunately, the number of scenarios to consider is prohibitively large even for moderate sized networks. For instance, verifying that a network with 200 links performs acceptably under all 3 simultaneous link failures [18, 22, 28] considering all traffic matrices collected at 10 minute intervals over a week's period requires testing over a billion scenarios. Many more tests are required if partial link failures are also considered. Even if such arduous testing can provide assurance that a given network design complies with a specific performance goal, it remains challenging to use such tests for designing networks that meet a performance requirement [29, 30]. Given the large space of possible designs and policies, and since exhaustively testing any one of them is prohibitive, architects today use ad-hoc design techniques that may lead to overly conservative solutions, or fall short of meeting performance requirements, and lack provable guarantees.

Besides resilient design, network verification [31–34] has become an important area of research in recent years. These works seek to verify the correctness of network dataplane (e.g., connectivity, routing loops, black holes, etc.) and network configuration, but provide little insight on network performance under failures.

1.2 Recent work and challenges

Researchers have only recently [18, 28] started considering combinatorially many failure states in designing network mechanisms. While these efforts offer a promising start, the state-of-the-art in this area is at an early stage. Next, we elaborate on the challenges that remain, and why these initial efforts have limitations.

R3 [28] considers networks under f simultaneous link failures and attempts to produce a link-based protection mechanism to ensure that the network is congestion-

free under all such failures. Unfortunately, R3 only models restricted forms of network adaptation, and overestimates the impact of failures, as we will see later. On the other hand, FFC [18], a path protection scheme, only considers traffic recovery called rescaling, which requires that, upon failure, the source diverts the traffic from failed paths to other operational paths, and this diversion of traffic is done so that, on the operational paths, the relative ratio of similar traffic is preserved.

Despite these advances, two challenges remain. First, networks increasingly respond in more flexible ways, further facilitated by the emergence of SDNs [3, 4, 6, 7] which allow architects more explicit control on how traffic is managed. To the best of the author's knowledge, no previous work has provided a solution to certifying network performance under failures with flexible network response. Second, worst-case design may be unduly conservative since a small number of bad failure scenarios may be expensive or even infeasible to design for. Further, worst-case approaches do not provide the architect with an understanding of the distribution of performance across failure scenarios, and an understanding of which scenarios and what fraction lead to unacceptable performance.

1.3 Contributions

Motivated by the aforementioned challenges, this thesis makes the following contributions:

- Robust certification with flexible network response. We develop the first general and formal framework to certify worst-case performance of network across wide range of scenarios with flexible network response. Our framework combines nonlinear optimization techniques with network-inspired mechanisms to tackle the intractability of modeling flexible network response. In addition to certifying a network design, the framework also applies to network synthesis, and we demonstrate with a link capacity augmentation case study.

- Generalized robust certification and automation toolkit. Based on the previous robust certification framework, we generalize it to apply to many more practical situations: multiple traffic classes with different priorities, richer failure patterns, routing restriction, and various performance metrics. We also develop a toolkit to automate the reformulation and linearization process, which is often cumbersome and error-prone to do manually.

- The Slice framework. We have developed a formal framework certifying network for more general objectives than worst-case performance. The framework efficiently classifies failure scenarios based on whether the network performs acceptably, which can later enable designs optimized for a given percentage of scenarios.

1.4 Results

We evaluate our solutions with multiple real topologies [35] and public traffic data [36], as well as synthetic traffic data generated using gravity model [37]. The results show the promise of our approaches. First, the robust certification framework is effective in reasoning about network performance under uncertainty, while modeling flexible network response. For instance, when considering f-link failures, our framework achieves better certification bounds than state-of-the-art [28], while surprisingly matching the optimal in all the experiments for the failure case study. Further, when considering variable traffic within weighted averages of historical demands, our framework also achieves tighter bounds than existing approaches. Second, the generalized robust certification framework performs well across various failure models, including partial failures and heterogeneous link failures where it continues to match the optimal. The toolkit we build to automate the critical steps needed by robust certification performs reasonably well, and solving the automatically generated model takes $23\% - 40\%$ of extra time compared to solving the manually generated model for GEANT, a moderate-size network which has 50 edges. Third, Slice is effective in clas-

sifying scenarios that can be supported by different routing schemes, revealing large performance gaps between these routing schemes. Slice also effectively aids design for more generic objectives than the worst-case, and doing so offers substantial benefits over worst-case design. For instance, with the GEANT network, existing approaches are unable to guarantee performance is acceptable for any 2-failure scenario because of a small number of bad scenarios. In contrast, Slice can handle most 2-failure scenarios by discovering which ones to design for. Slice performs well with reasonable certification time: on a single core 3.00GHz CPU, it takes 2.1 seconds for Slice to certify a link-based protection design for GEANT, and 80.8 seconds for Deltacom, a large network with 150 edges.

1.5 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 presents the robust certification framework to solve the worst-case network performance certification under failures and various traffic demands. Chapter 3 generalizes Chapter 2 to a variety of scenarios of practical interest, and presents an automation toolkit that we have developed. Chapter 4 presents Slice, a framework for analyzing network performance and designing networks for more general objectives than worst-case performance under failures. Finally, Chapter 5 summarizes, and discusses future directions.

2. ROBUST VALIDATION OF NETWORK DESIGNS UNDER UNCERTAIN DEMANDS AND FAILURES

2.1 Introduction

In designing wide-area networks for ISPs and cloud service providers, it is critical to ensure predictable performance at acceptable costs. However, achieving this goal is challenging because links fail (both owing to planned maintenance, and unplanned events such as fiber cuts and equipment failures) [12, 28, 38], and network traffic is variable [39] and constantly evolving [12].

Validating that a network can cope with a range of traffic conditions and failure scenarios is challenging because the number of scenarios to consider are typically exponentially many, and may even be non-enumerable. For instance, a common requirement is to verify that a network with N links can service demand for all combinations of f simultaneous link failures [18,22,28]. The number of failure scenarios to consider is $\binom{N}{f}$ for each demand. Further, the set of traffic matrices are not even enumerable, so naively considering all traffic matrices and failure scenarios is prohibitive. There is a huge gap between practice and existing theoretical tools. Oblivious routing [39–42], and more generally, robust optimization [43,44] allow bounding worst-case performance across multiple scenarios of interest. However, to ensure tractability of the problem, these techniques make the *conservative* assumption that the network cannot adapt to changes in demands by re-routing traffic [39–42], or admit limited forms of adaptation [28, 45]. In practice, networks do adapt by re-routing traffic as demands shift or failures occur, and such adaptation can make network operations much more efficient. Further, the advent of Software-Defined Networking (SDN) allows for network-wide optimization, and facilitates the deployment of flexible re-routing strategies [3, 4].

Given the large gap between theory and practice, the process of validating network designs today is ad-hoc, often requiring extensive simulations, which can be highly time consuming as well as fall short of guaranteeing provable bounds on network performance. In this chapter, we take a first step towards tackling this by presenting a formal framework to provide performance bounds on a network design across a set of scenarios (demands, failures). The key novelty in our framework is that it can accommodate a richer set of adaptation mechanisms, used in practice today, for re-routing traffic on failures and changes in demands.

When flexible routing strategies are considered, providing robust performance guarantees typically requires solving intractable non-convex (and often non-linear) optimization problems. We address these difficulties by leveraging cutting-edge techniques in the non-linear optimization literature [46]. An attractive aspect of these techniques is their generality, which allows them to be applied to a wide range of network validation problems. We show that these techniques lead to tighter bounds on the validation problem than existing state-of-the-art approaches in robust optimization, a finding that has applications beyond networking. Further, the bounds are tight in practical settings of interest - e.g., when demands are expressed as a convex combination of known historical demands [42]. Finally, we show how the techniques may be augmented with analysis of individual problem structure to substantially improve the quality of bounds.

For concreteness, we focus on link utilization, a widely accepted traffic engineering metric [28, 39, 42], which impacts application latency and throughput. We apply our framework to two contrasting, yet practical case studies to illustrate key aspects of our framework. The case studies differ in the type of uncertainty (failures and demands), and the type of adaptation. Specifically, we consider (i) multi-commodity flow (MCF) routing [28, 39, 47] which provides the most flexibility and efficiency, and (ii) MPLS-style tunneling [4, 48] which has more limited flexibility in routing.

While we focus on validation, our framework can enable the synthesis of designs with performance guarantees under uncertainty. We demonstrate this by showing how

our approach can aid operators in determining the most effective ways to augment link capacities while ensuring acceptable link utilizations under failures.

We evaluate our approach using multiple real topologies [35] and public traffic data [36]. Our framework performs better than oblivious formulations for both case studies, while surprisingly matching optimal in all the experiments for the failure case study. Further, we show our framework aids in (i) identifying bad failure scenarios; (ii) determining how to best augment link capacity to handle failures; and (iii) evaluating design heuristics – e.g., we show the potential for poor performance with common tunnel selection heuristics.

2.2 Motivation

2.2.1 Robust validation applications

A network design consists of (i) *invariant parameters,* which cannot be changed (or are costly to change) across failures and/or demands; and (ii) *adaptable parameters,* which may be flexibly chosen for any scenario. Our framework ensures that the choice of invariant parameters is acceptable across a set of demands and/or failures. Below, we present motivating examples.

Topology Design. In designing network topologies, operators must determine what links to lease and how much capacity to provision. While the set of links and their capacities is difficult to change across failures and demands, the network may adapt by re-routing traffic.

MPLS Tunnel Selection. A common traffic engineering practice is to use tunnels (e.g., MPLS [49]) between each ingress and egress switch, to ensure a core network that does not need to run the BGP protocol. In such settings, a light-weight adaptation mechanism is to switch traffic across k pre-selected tunnels between each source destination pair, which only involves changing flow tables in appropriate ingress switches [4, 48]. Changing tunnels is more heavy-weight since the flow tables of in-

ternal switches also need to be modified. A good choice of pre-selected tunnels can lower the frequency of changing tunnels in response to fluctuations in demand.

Middlebox placement. Network policy may require that some of the flows traverse a set of middleboxes such as firewalls and intrusion detection systems (IDS) [50, 51]. While the placement of network middleboxes typically occurs over relatively longer time-scales, traffic may be re-routed to handle normal traffic fluctuations or failures.

In these examples, the topology itself, and the set of tunnels and placement of middleboxes as applicable are *invariant* parameters, while the fraction of traffic sent along a given tunnel is an *adaptable* parameter.

Robust validation may be performed at initial design time, as well as in a *continual* fashion as the network evolves, and new projections on demands are available. Robust validation may indicate the network is no longer able to cope with the scenarios of interest, requiring the operator to consider changes to the design (e.g., by provisioning more capacity on links). Further, it can provide information on which scenario causes the network requirements to be violated, and aid in determining design changes to address the violations.

2.2.2 Robust validation framework

Our framework is closely related to robust optimization. In traditional robust optimization, input parameters belong to an uncertainty set, and the objective is minimized across any parameter choice in the set [43, 44]. Further, recourse actions may be considered that depend on the specific parameter value. In the networking context, a typical recourse action involves rerouting traffic to handle a change in traffic matrix or failure. The robust optimization literature considers limited forms of recourse actions, primarily for tractability reasons, which may lead to more conservative estimates of worst-case performance (§2.4.4). In contrast, we model richer

network adaptation, and tackle the resulting intractable problems. Prior approaches can be seen as special cases of our more general framework discussed below:

Metrics to capture performance of network design. Our framework can validate a variety of network metrics such as link utilizations, and bandwidth assigned to latency sensitive flows. For concreteness, in this chapter, we focus on the utilization of the most congested link (which we will refer to as *Maximum Link Utilization (MLU)*, a widely used objective function [28, 39, 42]. Though we do not discuss this extensively, our framework also applies to other common metrics of link utilizations (e.g., sum of penalties assigned to individual links, where penalties are convex functions of link utilizations [22, 25, 52]). We focus on utilizations given their extensive use in the traffic engineering literature, and since they reflect application performance (e.g., throughput for bandwidth sensitive applications is inversely related to utilizations).

Characterizing uncertainty in network conditions. We seek to validate that a network design performs well across demands and failure scenarios of interest. A typical set of failure scenarios to consider is all simultaneous failures of F or fewer links [22, 28]. The range of demands may be specified in multiple ways. A common model is to specify a set of historical traffic matrices, and require that all demands based on standard prediction models are considered. We formally discuss this model as well as other models in §2.4.3 and §2.5.2.

Modeling how networks adapt. Networks may respond to failures, and changes in demand by rerouting traffic in the best possible fashion to keep utilizations low. This can be achieved by determining the optimal routing (MCF) for a given scenario. This design point is becoming increasingly practical with the adoption of SDNs, given that periodic reoptimization for network state is feasible. Other models may allow adaptation, but with constraints. For instance, in the MPLS tunneling example, the network may adapt by changing how traffic is split across pre-selected tunnels between each ingress and egress pair, though the tunnels themselves do not change. This corresponds well to SDN deployments where only edge routers are SDN en-

abled [53]. Finally, policy constraints (e.g., a requirement that a set of middleboxes be traversed) may constrain how networks may adapt [50, 51, 54].

2.3 Formalizing robust validation

2.3.1 General problem structure

Let X denote the uncertainty set (possibly continuous and non-enumerable) of demands, or failures over which a given network design must be validated. The design includes all parameters that must remain invariant with changes in demands and failures (e.g., network topology, selection of tunnels, placement of middleboxes). For any given scenario $x \in X$, the network may adapt by routing traffic appropriately as described in §2.2.2.

Let y denote the parameters determined by the network when adapting to scenario x. This includes how traffic is routed – e.g., in the tunneling context, y includes parameters that capture how traffic must be split across tunnels – though there may be additional variables determined as we discuss in §2.3.2. Formally, the network validation problem may be written as:

$$F^* = \max_{x \in X} \min_{y \in Y(x)} F(x, y) \tag{2.1}$$

The *inner* minimization captures that for any given scenario $x \in X$, the network determines y in a manner that minimizes an objective function $F(x, y)$ from a set of permissible strategies $Y(x)$. For the fully flexible routing model, $Y(x)$ corresponds to strategies permitted by the standard MCF constraints [47], while for routing with middlebox policies, only strategies that ensure the desired set of middleboxes are traversed are permitted. The *outer* maximization robustly captures the worst-case performance across the set of scenarios X, assuming the network adapts in the best possible fashion for each x.

In this chapter, we focus on objective functions $F(x, y)$ that minimize the MLU as discussed in §2.2.2. We refer to (2.1) as the validation problem, since it can be

used to verify that a chosen design meets a desired utilization goal. For instance, when applied to topology design, $F^* > 1$ indicates the network is not sufficiently provisioned to handle all failures and demands of interest.

For any given scenario x, the inner problem is typically easy to solve (a linear program (LP)), since the network must compute y online to adapt to any failure or shift in demand. The validation problem is however challenging since exponentially many (and potentially non-enumerable) scenarios x must be considered.

2.3.2 Concrete validation problems

We next relate the general formulation (2.1) to two concrete case studies, chosen both for their practical importance and to illustrate key ideas of the framework.

- The first case study validates topology design against failures, with the most flexible network adaptation.
- The second example validates tunnel selection across variable demands, with network adaptivity constrained to splitting traffic across pre-selected tunnels.

The examples illustrate the generality of our framework in terms of its ability to consider both failures and demands (discrete and continuous uncertainty sets), and different types of adaptivity models (flexible and more constrained). However, our framework applies to a wider range of applications including simultaneously varying demands and failures, other adaptation models such as middlebox constraints, and other ways of combining adaptation models and uncertainty sets (§2.5).

We use the notation $x = (x^f, x^d)$ where x^f denotes a failure scenario and x^d denotes a particular demand, dropping superscripts when the context is clear. Likewise, we use $y = (r, U)$ where r denotes how traffic is routed, and U denotes utilization metrics computed as a result. Since our focus is on minimizing MLU, the inner problem may be expressed as $\min_{y \in Y(x)} U$, with constraints in $Y(x)$ which express the requirement that the utilization of every link is at most U. We now discuss how constraints $Y(x)$ are specified for our case studies.

Fully flexible routing under uncertain failures. Let x_{ij}^f be a binary variable which is 1 if link $\langle i,j\rangle \in E$ (the set of links) has failed, and 0 otherwise. Since we do not consider variable demands in this case study, we let d_{it} denote the known demand from source i to destination t. Let r_{ijt} denote the total traffic to t carried on link $\langle i,j\rangle$. Let c_{ij} denote the capacity of link $\langle i,j\rangle$. Then, $Y(x)$ corresponds to the standard MCF constraints [47], and may be expressed as:

$$\begin{aligned} & Uc_{ij}(1-x_{ij}^f) \geq \sum\nolimits_t r_{ijt} \quad \langle i,j\rangle \in E \\ & \sum\nolimits_j r_{ijt} - \sum\nolimits_j r_{jit} = \begin{cases} d_{it} & \forall t, i \neq t \\ -\sum_j d_{jt} & \forall t, i = t \end{cases} \\ & r_{ijt} \geq 0 \quad \forall i,j,t \end{aligned} \tag{2.2}$$

The first constraint ensures that (i) the utilization of link $\langle i,j\rangle$ is at most U for all non-failed links; and (ii) no traffic is carried on a failed link. The second constraint captures flow balance requirements. Specifically, the net outflow from node i to destination t is the total traffic destined to t when $i = t$, and d_{it} otherwise.

Tunnel constraints and uncertain demands. Given a set of pre-selected tunnels, let T_{ijstk} be a binary parameter that denotes whether the link $\langle i,j\rangle$ is on tunnel k for traffic from the source s to destination t. Let x_{st}^d denote the total $s-t$ traffic, and r_{stk} the subset of this traffic on tunnel k. Then, $Y(x)$ may be expressed as:

$$\begin{aligned} & Uc_{ij} \geq \sum\nolimits_{s,t,k} r_{stk} T_{ijstk} \quad \langle i,j\rangle \in E \\ & \sum\nolimits_k r_{stk} = x_{st}^d \quad \forall s,t; \quad r_{stk} \geq 0 \quad \forall s,t,k \end{aligned} \tag{2.3}$$

The first constraint ensures that the utilization of every link is bounded by U. The second constraint captures that the sum of the traffic on all tunnels k for each $s-t$ pair must add up to the total demand of that pair.

$$Y(x) = \left\{ (r, U) \;\middle|\; \begin{array}{ll} \gamma_k(x)U \geq \sum_{i \in I} \beta_{ik}(x) r_i & k \in K \\ \sum_{i \in I} \alpha_{ij}(x) r_i \geq \delta_j(x) & j \in J \\ r \geq 0 & \end{array} \right\}$$

$$F(x, r, U) = U$$

Fig. 2.1.: General structure of determining routes (r) for scenario x to minimize MLU (U).

$$\begin{array}{lll} (W) \max\limits_{x,v,\lambda} & \sum_{j \in J} \delta_j(x) v_j & \\ \text{s.t.} & \sum_{j \in J} \alpha_{ij}(x) v_j \leq \sum_{k \in K} \beta_{ik}(x) \lambda_k & i \in I \\ & \sum_{k \in K} \gamma_k(x) \lambda_k = 1 & \\ & x \in X, (v_j)_{j \in J} \geq 0, (\lambda_k)_{k \in K} \geq 0 & \end{array}$$

Fig. 2.2.: General structure of validation problem derived from Figure 2.1 as a single-stage formulation.

2.3.3 Non-linear reformulation

The validation problem in (2.1) has been represented in a form referred to as a *two-stage formulation* (e.g., [45]). In the two-stage problem, the optimal second-stage variables (y) depend on the first-stage (x). We simplify this problem by re-expressing it as a single-stage problem, where all the variables are determined simultaneously.

In many network validation problems, including our case studies, the inner problem $\min_{y \in Y(x)} F(x, y)$ is an LP in variable $y = (r, U)$ for a fixed scenario x. This is reasonable because online adaptations of y must be computationally efficient. Figure 2.1 shows the general structure of the LP. Notice that the coefficients depend on scenario x. For example, in the failure validation case study, $\alpha_{ij}(x)$, $\beta_{ik}(x)$, and $\delta_j(x)$

$$
\begin{aligned}
(F)\max_{v,\lambda,x} \quad & \sum\nolimits_{t,i\neq t} d_{it}(v_{it}-v_{tt}) \\
\text{s.t.} \quad & v_{it}-v_{jt}\leq \lambda_{ij} \quad \forall t, \langle i,j\rangle \in E \\
& \sum\nolimits_{\langle i,j\rangle \in E} \lambda_{ij} c_{ij}(1-x^f_{ij}) = 1 \\
& x^f \in X; \quad x^f_{ij}\in\{0,1\}; \quad \lambda_{ij}\geq 0, \quad \langle i,j\rangle \in E
\end{aligned}
$$

$$
\begin{aligned}
(V)\max_{v,\lambda,x} \quad & \sum\nolimits_{s,t} x^d_{st} v_{st} \\
\text{s.t.} \quad & v_{st} \leq \sum\nolimits_{\langle i,j\rangle \in E} T_{ijstk}\lambda_{ij} \quad \forall s,t,k \\
& \sum\nolimits_{\langle i,j\rangle \in E} \lambda_{ij} c_{ij} = 1 \\
& x^d \in X; \quad \lambda_{ij}\geq 0, \quad \langle i,j\rangle \in E
\end{aligned}
$$

Fig. 2.3.: Formulations of validation problems for failure case study (F), and tunnel selection case study (V).

are constants while $\gamma_k(x)$ is a linear function of x. For a specific value of x, the inner problem is an LP.

It is well known that every LP (referred to as a primal form) involving a minimization objective may be converted into an equivalent maximization LP (referred to as a dual form) which achieves the same objective (assuming the dual is feasible) [55]. The validation problem can then be expressed as a single-stage formulation by:

1. Rewriting $\min_{y\in Y(x)} \ F(x,y)$ as an equivalent maximization problem using LP duality.

2. Adding the constraints $x \in X$ to the dual form to capture the set of demands or failure scenarios of interest.

Figure 2.2 shows the general structure of the validation problem as a single-stage formulation. Notice that variables r and U in Figure 2.1 have been replaced by the

dual variables λ and v. Moreover, x is now a variable since the problem validates utilization over all uncertain scenarios.

Formulations **(F)** and **(V)** in Figure 2.3 capture the validation problem for our case studies involving failures (2.2) and variable demands (2.3) respectively. At first glance, both formulations appear non-linear – the objective in (V) involves products of x^d and u variables, while the second constraint of (F) involves a product of variables x^f_{ij} and λ_{ij}. In §2.4.2, we show that (F) can be written as an integer program (IP) when X is the set of scenarios involving the failure of f or fewer links simultaneously. Regardless, both (V) and (F) are hard problems (non-linear non-convex and IP respectively).

2.4 Making validation tractable

§2.3.3 has shown that the validation problems, including our case studies, are typically intractable, Given the intractable nature of the problems, we do not solve them to optimality, rather seek ways to obtain upper bounds on the true optimal of (2.1). Since the purpose of validation is to ensure a design is acceptable, an upper bound that satisfies the design criteria is sufficient.

We aim for a general approach to tackle a wide range of validation problems. In the optimization literature, problems such as (2.1) are referred to as robust optimization problems and have been tackled mostly for limited adaptations. Instead, we use non-linear programming techniques, and show they achieve better bounds, a finding that has applications beyond networking.

We introduce the approach in §2.4.1, and how it applies to our case studies involving failures and variable demands in §2.4.2 and §2.4.3 respectively. Although our framework is general, analysis of problem structure can substantially improve the quality of bounds, as we will show for the failure case study in §2.4.2. Finally, in §2.4.4, we compare our techniques with benchmarks drawn from the network man-

agement and robust optimization literature, and show that our techniques can obtain tighter bounds than these approaches.

2.4.1 Relaxing validation problems

Our approach works by relaxing the validation problems into more tractable LPs, and obtaining an upper bound on the worst-case link utilizations across scenarios. An optimization problem L is a relaxation of a problem N if every feasible solution in N can be mapped to a feasible solution in L, and the mapped solution's objective value in N is no better than that of its mapping in L.

Reformulation-Linearization Technique (RLT) [46] is a general approach to relax non-linear integer problems. The technique reformulates the problem by (i) adding new constraints obtained by taking products of existing constraints; and (ii) linearizing the resulting formulation by replacing monomials with new variables. For our problem (W), RLT can be constructed as long as $\alpha_{ij}(x)$, $\beta_{ik}(x)$ and $\gamma_k(x)$ are polynomial functions.

For example, consider a non-linear optimization problem where the objective is to minimize $xy - x + y$ subject to the constraints: (i) $(x - 2) \geq 0$; (ii) $(3 - x) \geq 0$; (iii) $(y - 3) \geq 0$; and (iv) $(4 - y) \geq 0$. Products of pairs of constraints are taken – e.g., the product of constraints (i) and (iii) results in a new derived constraint $(x - 2)(y - 3) \geq 0$, i.e., $xy - 3x - 2y + 6 \geq 0$. The product term xy is replaced by a new variable z. The objective is rewritten as $z - x + y$, and the derived constraint in the previous step expressed as $z - 3x - 2y + 6 \geq 0$. The resulting problem is linear, as it no longer has product terms. However, it is a relaxation in the sense that constraints (e.g., $z = xy$) that must be present to accurately capture the original problem are not included in the new problem.

The above represents the first step in a hierarchy of relaxations and the next steps involve multiplying more than two constraints and linearizing as discussed above. Further, the RLT hierarchy can be tightened using convex relaxations of monomials,

which yield other well known hierarchies. As long as the set of inequalities in the verification problem define a bounded set, higher levels of this hierarchy of relaxations converge to the optimal value of the non-linear or integer program [46, 56]. Since the generated LPs can be large (more variables and constraints), we restrict attention to the first level of this hierarchy. Further, in practice, it often suffices to consider a subset of products even for the first level, which keeps the complexity of the resulting program manageable.

2.4.2 Validation across failure scenarios

Here, we discuss the RLT relaxation technique for our failure case study (formulation (F)). For concreteness, we consider all failure scenarios involving the simultaneous failure of f or fewer links. This failure model is used commonly in practice [28]. We discuss how to generalize the failure model later (§2.5). Incorporating this model results in replacing the constraint $x_{ij}^f \in X$ in (F) with the constraints $\sum_{\langle i,j\rangle \in E} x_{ij}^f \leq f$, and $x_{ij}^f \in \{0,1\}$.

Empirically, a simple RLT relaxation of the formulation does not yield a sufficiently tight upper bound to the validation problem. Instead, we reformulate the validation problem (F), and consequently derive constraints for the RLT relaxation, as described below:

Reformulating the validation problem. We add variables to (2.2), in a way that gives more flexibility in choosing solutions, but does not change the optimum. Adding variables to a primal results in additional constraints to the dual. Consequently, we derive constraints for (F) and the associated RLT relaxation LP, which

are derived from the LP dual of (2.2), thus improving the bound on utilization. Specifically, we reformulate (2.2) as follows:

$$
\begin{aligned}
& Uc_{ij}(1-x_{ij}^f) + a_{ij} \geq \sum\nolimits_t r_{ijt} \quad \langle i,j\rangle \in E \\
& \sum\nolimits_j r_{ijt} - \sum\nolimits_j r_{jit} = \begin{cases} d'_{it} & \forall t, i \neq t \\ -\sum_j d'_{jt} & \forall t, i = t \end{cases} \\
& r_{ijt}, a_{ij} \geq 0 \quad \forall i,j,t \\
& d'_{it} = \begin{cases} d_{ij} + a_{ij} & \langle i,j\rangle \in E \\ d_{ij} & \langle i,t\rangle \notin E \end{cases}
\end{aligned}
\tag{2.4}
$$

We augment each link $\langle i,j\rangle$'s capacity with the extra (variable) slack capacity a_{ij} for which we reserve the capacity along alternate paths in the network. In particular, the first constraint allows up to a_{ij} of the traffic on link $\langle i,j\rangle$ to be bypassed on the associated virtual link without counting it against the utilization of link $\langle i,j\rangle$. To compensate for this, we increase the total traffic that must be routed from i to j by a_{ij}, as indicated by the last constraint. It can be shown that (2.4) achieves the same optimal as (2.2). Further, because any feasible solution to (2.2) is also feasible to (2.4) (with slack variables a_{ij} being 0), (2.4) is more flexible in that it admits additional solutions.

Following the procedure outlined in Figures 2.1 and 2.2, this reformulated primal yields a reformulated validation problem (F') which consists of (F) with constraints $\lambda_{ij} \leq v_{ij} - v_{jj},\ \forall \langle i,j\rangle \in E$. Then, (F') simplifies to:

$$
\begin{aligned}
& (G)\max \sum\nolimits_{i,t} d_{it} v_{it} \\
& v_{it} - v_{jt} \leq v_{ij} \quad \forall t, \langle i,j\rangle \in E && (2.5) \\
& \sum\nolimits_{\langle i,j\rangle \in E} v_{ij} c_{ij}(1-x_{ij}^f) = 1 \\
& \sum\nolimits_{\langle i,j\rangle \in E} x_{ij}^f = f && (2.6) \\
& v_{it} \geq 0, v_{tt} = 0 \quad \forall i,t && (2.7) \\
& x_{ij}^f \in \{0,1\}, \quad \langle i,j\rangle \in E && (2.8)
\end{aligned}
$$

Proposition 2.4.1 *Reformulation (G) achieves the same optimal value as the original validation problem (F).*

Proof Clearly, the optimal value of (G) is no more than that of (F') because (G) has the following additional constraints (i) for all $\langle i,j\rangle \in E$, $\lambda_{ij} = v_{ij}$, and (ii) and for all nodes t, $v_{tt} = 0$. Therefore, we only need to show that the optimal value of (F') is no more than that of (G). Let $(\lambda^*, v^*, x^{f^*})$ be optimal in (F'). Denote by $\mathrm{SP}_{it}(\lambda)$ the shortest path between i and t with edge-lengths λ. For any path P_{it} connecting nodes i and t, it follows from the first constraint in (F) that $v^*_{it} - v^*_{tt} \leq \sum_{\langle i,t\rangle P_{it}} \lambda^*_{ij}$ and, so, minimizing rhs over paths yields $v^*_{it} - v^*_{tt} \leq \mathrm{SP}_{it}(\lambda^*)$. For any link $\langle i,j\rangle$ this implies that $\lambda^*_{ij} \leq v^*_{ij} - v^*_{jj} \leq \mathrm{SP}_{ij}(\lambda^*) \leq \lambda^*_{ij}$, where the first inequality is from the slack-induced constraint, the second inequality follows from discussion above, and the third inequality because $\langle i,j\rangle$ is a valid path from i to j. Therefore, equality holds throughout. Now, consider the solution (v', x^{f^*}) such that $v'_{it} = \mathrm{SP}_{it}(\lambda^*)$. We show that this solution is feasible to (G). Clearly, $v'_{it} - v'_{jt} \leq v'_{ij}$ because the shortest path from j to t can be augmented with $\langle i,j\rangle$ to yield a path from i to t. Next, because $v'_{ij} = \mathrm{SP}_{ij}(\lambda^*) = \lambda^*_{ij}$, where the last equality was shown above, it follows that $\sum_{\langle i,j\rangle} v'_{ij} c_{ij}(1 - x^{f^*}_{ij}) = 1$. Moreover, $v'_{it} = \mathrm{SP}_{it}(\lambda^*) \geq 0$ because $\lambda^*_{ij} \geq 0$ and, trivially, $v'_{tt} = \mathrm{SP}_{tt}(\lambda^*) = 0$. Therefore, (v', x^{f^*}) is feasible to (G). Finally, $\sum_{i,t} d_{it} v'_{it} = \sum_{i,t} d_{it} \mathrm{SP}_{it}(\lambda^*) \geq \sum_{i,t} d_{it}(v^*_{it} - v^*_{tt})$, where the equality follows from the definition of v' and the inequality by summing products of $\mathrm{SP}_{it}(\lambda^*) \geq (v^*_{it} - v^*_{tt})$ with $d_{it} \geq 0$. Therefore, the optimal value of (G) is at least as large as that of (F'). ■

The proof shows that an optimal solution of (F') satisfies $v_{tt} = 0$, $\forall t$ and $v_{ij} = \lambda_{ij}$, $\forall \langle i,j\rangle \in E$. The proposition then follows since (F) and (F') achieve the same optimal value having been derived respectively from primals (2.2) and (2.4) that achieve the same optimal.

Although (G) is non-linear because the product $v_{ij} x^f_{ij}$ is in the second constraint, we note that (G) has a finite objective only if the minimum cardinality edge-cut set

of the topology contains more than f links, a condition that can be verified in polynomial time [57]. Moreover, we prove in the following proof that if f failures cannot disconnect the nodes of the network, v_{ij} is bounded. Then, standard linearization of $v_{ij}x_{ij}^f$ that uses bounds on v_{ij} and $x_{ij}^f \in \{0,1\}$ reduces (G) to a mixed-integer linear program.

(G) can be formulated as an Integer Program after a polynomial time verification of graph connectivity:

Proof The objective of (G) is not finite if the minimum edge-cut set contains f or fewer links, a fact that can be verified in polynomial time [57]. Now consider that the topology is not disconnected after any simultaneous set of f link failures. We show that $v_{it} \leq \frac{1}{c_{min}}$, where $c_{min} = \min_{\langle i,j\rangle \in E} c_{ij}$. To prove the bounds, let NF denote the set of links that do not fail when the optimal value of (G) is achieved. For any pair of nodes i and t, there exists a path (whose edges we denote as P) on the failure of this set of links. By adding the first constraint of (G) for all edges along P, $v_{it} = \sum_{\langle i,j\rangle \in P} v_{ij} \leq \sum_{\langle i,j\rangle \in NF} v_{ij}$. From the second constraint of (G), $\sum_{\langle i,j\rangle \in NF} v_{ij}c_{ij} = 1$, and hence $\sum_{\langle i,j\rangle \in NF} v_{ij} \leq 1/c_{min}$. The bounds follow.

Multiplying the bound constraints $0 \leq v_{ij} \leq \frac{1}{c_{min}}$ with x_{ij}^f and $1 - x_{ij}^f$ allows us to linearize the above mixed-integer non-linear program into an integer program. This is achieved by replacing $v_{ij}x_{ij}^f$ with a new variable vx_{ij}^f and observing that it is automatically constrained to be $v_{ij}x_{ij}^f$ when $x_{ij}^f \in \{0,1\}$. ■

Relaxing the validation problem. Since the validation problem (G) is still intractable, we derive its first-level RLT relaxation as follows. First, the binary requirement $x_{ij}^f \in \{0,1\}$ is replaced by bound constraints, $x_{ij}^f \geq 0$ and $(1 - x_{ij}^f) \geq 0$. Next, the product of these bound constraints is taken with (2.5) and (2.7) and the product of (2.6) and (2.7) is taken. Finally, the nonlinear constraints are relaxed by introducing $vx_{iji'j'}^f$ to denote $v_{ij}x_{i'j'}^f$.

2.4.3 Validation across traffic demands

We now consider the tunnel selection case study (formulation (V)) and the problem of verifying utilization against uncertain demands. We discuss two models for specifying demands, and discuss the RLT relaxations.

Specifying demands. We consider two models:

• *Predicted demand:* This corresponds to scenarios when demands may be predicted from past history, a commonly used practice today. Consider optimizing the system for a set of known historical traffic matrices $\{d^h\}_{h\in H}$. As observed in [42], many predictors including the exponential moving average estimate the traffic matrix for a given interval as a convex combination of previously seen matrices. It may be desirable to verify the system for the convex hull of $\{d^1, d^2, \ldots d^h\}$, which ensures that all such predictors can be serviced with reasonable utilization. Specifically, this may be modeled by replacing the constraint $x^d \in X$ in (V) by the constraints $x^d = \sum_{h\in H} x_h d^h$, $x_h \geq 0$ and $\sum_{h\in H} x_h = 1$.

• *All demands that can be handled by the topology:* It may be desirable to understand the extent to which a topology must be over-provisioned if a tunneling solution is used compared to using an optimal MCF solution. This may be modeled by replacing the constraint $x^d \in X$ in (V) by the standard MCF constraints with x^d_{st} denoting demand from source s to destination t, and x^g_{ijt} a flow variable denoting traffic to t on link $\langle i, j\rangle$.

Obtaining the RLT relaxation. We obtain the RLT relaxation by taking the product of (i) inequalities involving v and λ variables with constraints of the form $x \geq 0$; (ii) inequalities involving x variables with constraints of the for $\lambda \geq 0$; (iii) inequalities involving v or λ with inequalities involving x; and (iv) equalities involving x variables with v variables.

2.4.4 Comparisons to alternate approaches

A key novelty of our framework is that it provides theoretical bounds on network performance across failures/demands, while allowing flexible adaptation. We can show that each RLT constraint we introduce in the problem makes the adaptations more flexible in a specific way. In contrast, prior theoretical work has focused on limited forms of adaptivity and we use them as benchmarks for our RLT relaxation approach. We show that our approach provides bounds that are at least as tight as these prior theoretical works, and later show empirically (§2.6) that the bounds are better in practice.

Oblivious approaches and generalizations. Oblivious routing [39, 42, 58–60] bounds utilizations across all links for a set of demands, while limiting how the network adapts to any given demand. While oblivious routing has mainly been considered in the context of MCF [39,42], the oblivious approach applies to other networking contexts. For instance, in our tunneling case study, an *Oblivious Tunneling* formulation constrains y_{stk} (traffic on tunnel k from s to t) to be of the form $y_{stk} = \alpha_{stk} x^d_{st}$, where α_{stk} is invariant across demands.

The robust optimization literature has considered a more general form of adaptation than an oblivious approach, which can enable tighter bounds on worst-case link utilization [45]. Here, every variable y_i (e.g., each y_{stk} variable in our tunneling example) that a network determines for a given scenario x, is constrained to have the form $y_i = \alpha_{i0} + \sum_j \alpha_{ij} x_j$ where all α_{ij} coefficients must be invariant with x. Note that x_j variables capture scenario x (e.g., in our tunneling example, x is a traffic matrix, and each x_j is a cell in the matrix). In optimization terminology, y_i is an affine function of x. Note that an oblivious approach is a special case of affine policies where many of the α coefficients are zero.

We say the *linearity requirement* has been met when constraints $Y(x)$, and objective $F(x, y)$ are linear in (x, y). For example, in the tunneling case study, the constraints (2.3) and the objective, U, are linear in U, r, and x^d. Further, the conditions

are satisfied by the original oblivious routing [39, 42], and while we do not elaborate, by other case studies such as routing with middleboxes. When network adaptation is restricted to affine policies, and the linearity requirement is met, an optimal set of α_{ij} coefficients may be computed efficiently using LP to minimize worst-case link utilizations [61]. We now state our result:

Proposition 2.4.2 *When the linearity requirement is met, an optimal affine policy can be efficiently computed. Under these circumstances, the first-level RLT relaxation for a validation problem is at least as tight as the bound from the optimal affine policy.*

The proof involves taking duals of the RLT relaxation. We do not elaborate on the technical details, and focus on the implications for validation. Further, for predicted demand (§2.4.3), Proposition 2.4.2 already implies that the first-level RLT can provide as tight a bound as an oblivious approach. However, we have shown a stronger result:

Proposition 2.4.3 *For the predicted demand case, the first-level RLT relaxation is an exact solution, while the oblivious solution may not always be exact.*

Some of our case studies do not satisfy the linearity requirement. In particular, the requirement is not satisfied for our case study involving failures (2.2) because the first constraint in (2.2) involves a non-linear term (product of U and x). Under these circumstances, an optimal affine policy may not be efficiently computable, and is thus not a viable benchmark. However, our framework is still applicable (as our failure case study has shown), since it only requires that the weaker condition that $Y(x)$ is linear in y variables for fixed x needs to be satisfied.

Benchmark for failure case study. R3 [28] tackles the validation problem under failures, but with the more limited goal of determining whether a network can handle all failures scenarios without congestion (i.e., whether $MLU \leq 1$), and with restrictions on how the network can adapt. R3 replaces failures with virtual demands (the traffic to be rerouted on failures) and computes an oblivious protection routing (MCF) for the virtual demand associated with each link. The formulation is only valid

when $MLU \leq 1$, since the virtual demand on each link is assumed to not exceed the link capacity. In contrast, our formulation (G), and the associated first-level RLT relaxation is valid for any MLU, which can aid in tasks such as determining which failure scenarios are bad when the network is not sufficiently provisioned, and how best to augment link capacities to handle failures (§2.6.3). When $MLU \leq 1$, the bounds from R3 are conservative for our validation problem owing to the restriction on adaptations and since the impact of the failures is over-estimated. We have been able to show:

Proposition 2.4.4 *The first-level RLT relaxation of (G) provides at least as tight a bound as R3, whenever R3 provides a valid bound.*

In fact, we can impose similar restrictions as R3 on how traffic is rerouted in response to failures by appropriately choosing a subset of RLT constraints. Yet, the MLU will reduce because we optimally chose slack a_{ij} instead of assuming it is c_{ij}. The proof of Proposition 2.4.4 considers a special affine policy for $y = (r, U, a)$ in (2.4), where U does not adapt with x and $a_{ij} = \alpha_{ij} x_{ij}$. We show that all such policies that yield $U \leq 1$ can be made feasible to R3, and, therefore, the bound for R3 is no better than the one obtained with this policy restriction. Since RLT encompasses search over these policies, the result follows. We will show in §2.6 that RLT yields tighter bounds than R3 whenever the network utilization is less than 1.

2.5 Aiding synthesis and generalizations

§2.4 has shown how our framework applies to two validation case studies. We next discuss applications to robust design (§2.5.1), and to other validation problems (§2.5.2).

2.5.1 Augmenting capacities to bound utilization

To see how our validation framework can help in robust design, consider the problem of incrementally adding capacity to existing links to ensure all failure scenarios of interest can be handled (with $U \leq 1$), while minimizing the costs of augmented capacity. We can extend (2.1) to model the capacity augmentation problem as follows:

$$\min_{\delta \geq 0} \max_{x^f \in X} \min \left\{ \sum_{\langle i,j \rangle \in E} w_{ij}\delta_{ij} \;\middle|\; \begin{array}{c} (c_{ij} + \delta_{ij})(1 - x^f_{ij}) \geq \sum_t r_{ijt} \\ r \;\text{ is a routing for }\; d \end{array} \right\}$$

where X is the set of failure scenarios, and δ_{ij} and w_{ij} are respectively the incremental capacity added to link $\langle i, j \rangle$, and the cost per unit capacity. Further, r is a routing for d if r satisfies the flow balance constraints of an MCF formulation. Then, dualizing the inner minimization problem results in a two-stage formulation whose inner problem is an IP since X is a discrete set. However, using the RLT relaxation technique presented in our framework, we replace the inner problem by an upper-bounding LP which can be dualized to upper-bound the cost of augmentation. This yields an LP based approach to conservatively augmenting capacity.

The above discussion also motivates an iterative approach to design. At each iteration, we solve a capacity augmentation problem considering failure scenarios identified in earlier rounds. Then, with the new capacities, we solve the failure validation problem to identify additional failure scenarios and iterate. At any stage, this provides a lower bound on the optimal capacity augmentation. Although the iterative procedure works well empirically for capacity augmentation, in other robust design problems, finding the worst uncertainty may be hard and the procedure may require too many iterations. In contrast, the LP based approach presented above always yields a conservative robust design quickly.

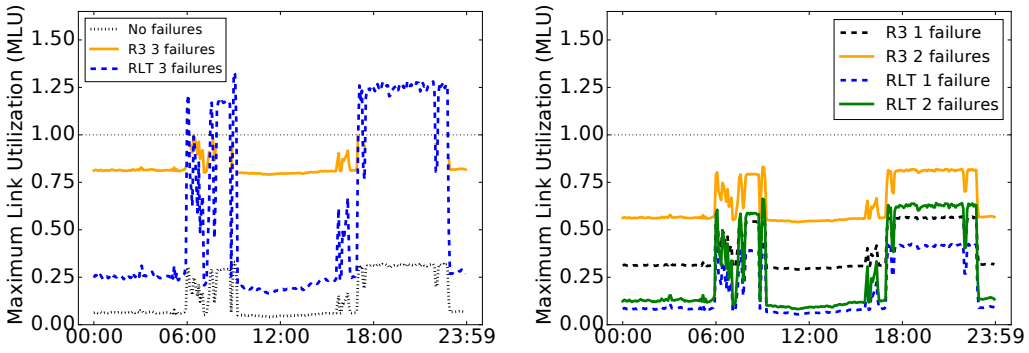
2.5.2 More general validation problems

In this section, we discuss how our framework can tackle other validation problems beyond our case studies.

Routing with middlebox constraints. Our framework may be used to obtain bounds on MLU when routing is constrained to satisfy middlebox policies [50,51,54]. The requirement that traffic from s to t be routed across a series of middleboxes can be modeled by associating each flow with a state variable which indicates a given middlebox has been traversed. The state is modified by each middlebox on the path. (2.2) is reformulated by introducing variables $r_{ijst\phi}$ which denote the flow on link $\langle i,j\rangle$ from s to t and for packets with state ϕ, and appropriately modifying the flow balance equations, and capacity constraints. The validation problem may now be formulated and solved across failures, or demands using the same approach as our two case studies.

Simultaneously varying failures and demands. We may desire to ensure utilizations are acceptable across any combination of failures and demands. This can be achieved by directly taking (F), and replacing demand variables d_{st} with variables x^d_{st}, and adding constraints for both x^d and x^f using previously studied models. A similar RLT relaxation applies in this case as well.

Handling shared risk link groups (SRLGs). We have considered a model where at most f links fail simultaneously. In practice, multiple links may fail together (e.g., a fiber cut may impact all links in the affected fiber bundle) [22]. The set of link groups G is considered, and each group g is associated with a set of links that may fail together. We introduce variables x^f_g which indicates whether a particular link group has failed. The validation problem is modeled by considering formulation (F), and replacing the constraints $x^f \in X$ with the constraints $x^f_{ij} = 1 - \prod_{\langle i,j\rangle \in g, g\in G}(1 - x^f_g)$, where all x^f_{ij} and x^f_g variables are binary, and $\sum_{g\in G} x^f_g \leq f$. This captures that link $\langle i,j\rangle$ has failed iff any group that it belongs to has failed, and at most f link groups may fail simultaneously. To eliminate the product terms, the first constraint can be linearized with the constraints $x^f_{ij} \geq x^f_g, \langle i,j\rangle \in g, g \in G$, and the constraint $x^f_{ij} \leq \sum_{\langle i,j\rangle \in g, g\in G} x^f_g$. An RLT relaxation may now be applied as normal. Alternately, other linearized constraints can be derived from exploiting this relationship within the RLT scheme that we do not detail.



(a) Abilene: baseline and 3 failures (b) Abilene: 1 and 2 failures

Fig. 2.4.: Validation across failure scenarios, comparing RLT and R3, for various topologies.

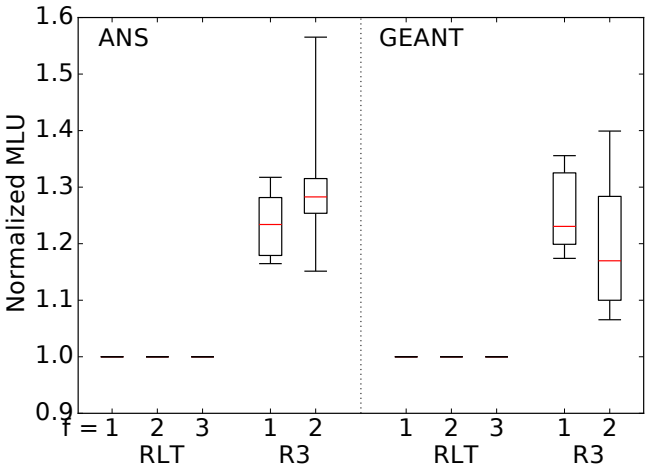


Fig. 2.5.: Latency CDF of different failure scenarios.

2.6 Evaluation

We evaluate the effectiveness of our framework in validating topology design under failures (§2.6.1), and tunnel selection under variable demands (§2.6.4). We compare our performance bounds with those obtained using existing approaches. Further, we show we can (i) identify bad failure scenarios (§2.6.2), (ii) optimally augment network capacity to handle failures (§2.6.3), and (iii) evaluate common design heuristics for tunnel selection (§2.6.4).

Table 2.1.: Topologies

Network	Nodes	Edges	Date	Link Capacity
Abilene	11	28	2004	homogeneous
ANS	18	50	2011	homogeneous
GEANT	41	118	2014	heterogeneous

We evaluate our work using real topologies obtained from the Internet Topology-Zoo [35]. We focus on three topologies: Abilene, ANS and GEANT [62] (Table 2.1), where Abilene and ANS have homogeneous link capacities, and GEANT has heterogeneous link capacities. All our LPs and IPs were run using CPLEX [63] (version 12.5.1.0). Our primary performance metric is MLU (§2.2.2) though we also consider how MLU impacts latency through emulation on an SDN testbed (§2.6.2).

2.6.1 Validation across failure scenarios

We evaluate the efficacy of our approach for determining MLU across failure scenarios, comparing MLU bounds produced by our RLT-based LP (§2.4.2) with (i) the IP (G) which can determine the optimal MLU value (§2.4.2); and (ii) R3 [28] (§2.4.4), the best known current approach. (G) is an intractable problem used only for comparison, and the running time of both our RLT relaxation and (G) is shown at the end of this section. We report the MLU returned by the R3 formulation instead of just the binary decision of whether $MLU \leq 1$ used in the original work. Recall R3 only provides valid bounds on MLU when $MLU \leq 1$ (§2.4.4). We study failure scenarios involving f arbitrary link failures, f ranging from 1 to 3, which practitioners indicated were important to consider. To ensure connectivity after multiple failures, we eliminated one-degree nodes from ANS and GEANT topologies, and modeled each edge as consisting of 2 sub-links of equal capacity for all topologies. The resulting ANS (GEANT) network has 17 (32) nodes and 96 (200) edges.

Table 2.2.: Average running time of the RLT scheme and the optimal IP for GEANT. For the optimal IP, the average running time is computed with instances that completed in 2 hours.

# of Failures	RLT (sec)	IP (sec)	% IP Completed
1	640.58	60.50	100
2	622.60	394.97	100
3	607.68	3890.16	60
4	598.31	–	0
5	586.79	–	0

We begin by presenting results with the Abilene topology using real traffic data [36]. Figure 2.4a shows the MLU for $f = 3$, for the RLT and R3 schemes for all traffic matrices measured on April 15th, 2004, a day which experienced a wide variety of traffic patterns. The MLU under normal conditions (no failures) is shown as a baseline. The RLT scheme matches the optimal IP scheme for all traffic matrices, and hence we do not present the IP scheme. The graph shows that several traffic matrices stress the network to achieve $MLU > 1$, indicating it is not provisioned to handle all three simultaneous link failures. Further, the RLT scheme achieves a tighter bound than R3 for all cases where $MLU \leq 1$, and unlike R3, it can provide valid bounds even when $MLU \geq 1$.

Figure 2.4b presents results for Abilene, but for $f = 1$ and 2. Again, the optimal IP is not shown, since RLT matches optimal. The graph shows the MLU is under 1 for all matrices, indicating the network can handle all possible 2 link failures. Moreover, RLT achieves a tighter bound on MLU than R3 for all matrices. We repeat the experiments with ANS and GEANT topologies. Since actual traffic matrices were not available to us, we generated multiple traffic matrices for each topology using the gravity model [37]. The traffic matrices were chosen so as to keep the link utilizations between 0.3 and 0.45 under normal conditions. Figures 2.5 presents the normalized MLU for

R3 and RLT, relative to the optimal IP for each f. Boxplots depict variation across the matrices. The graph shows that for all f and all traffic matrices, RLT always achieves a normalized MLU of 1, indicating it always matches optimal. The normalized MLU with R3 is higher, e.g., ranging from 1.15 to 1.57 for ANS $f = 2$. Note that results for R3 are not shown for $f = 3$ because all traffic matrices with GEANT, and all but 2 matrices with ANS achieved an optimal MLU above 1, indicating the network was not sufficiently provisioned for them. In contrast, the optimal MLU was under 1 for $f = 1$ and 2, for both topologies, and all traffic matrices.

A surprising aspect of our results is that across all topologies and traffic matrices, the RLT scheme matches the optimal IP. We have also investigated this further for other synthetic topologies and other settings, and have found RLT to match optimal across all the examples. We leave to future work further investigation of whether the first-level RLT in fact can be proven to match the optimal for this case study, or if counter-examples exist.

Running time. We report the running time (Table 2.2) from experiments with GEANT, the largest topology in our set, on a machine with 8-core 3.00 GHz Intel Xeon CPU and 94 GB memory. To create an even larger topology, we modeled each edge as consisting of 10 sub-links of equal capacity. The resulting network has 32 nodes and 1000 edges. Table 2.2 shows the average running time of RLT and the optimal IP using 10 traffic matrices generated by the gravity model. Since many IP instances didn't finish even after several hours, we set a 2-hour limit to the solver. Results show that the running time stays stable for RLT, but explodes for the optimal IP, as the number of failures increases. At $f = 3$, 40% of the IP instances did not converge. At $f = 4$ and 5, none of the IP instances converged, and the gaps[1] are larger than 0.5 in all the cases, indicating that the IP solutions found by the solver within 2 hours are still far from the optimal.

[1]gap = (UB - LB) / UB, where UB and LB denote the upper bound and the lower bound of the optimal objective value.

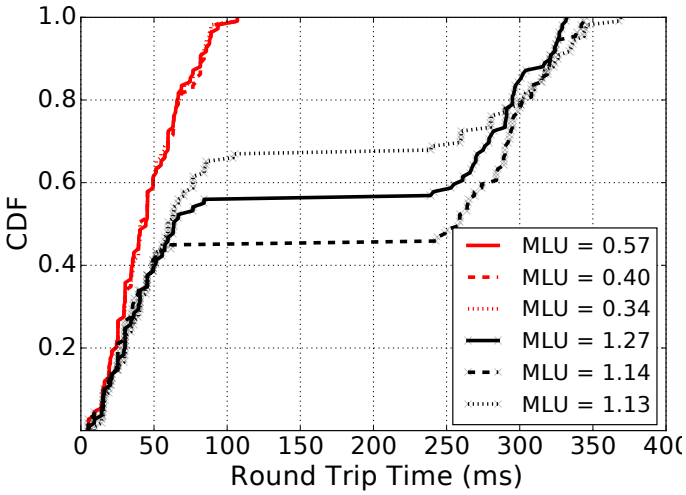


Fig. 2.6.: Latency CDF of different failure scenarios.

2.6.2 Impact of failures on application performance

Our validation framework can be used to identify failure scenarios that result in high MLU, which could then be emulated on a network testbed to study application performance metrics such as latency under such scenarios.

Finding bad failure scenarios. In general, it is hard to find failure scenarios for the original validation problem (G) that result in a high MLU since it is an IP. A random search is inefficient – e.g., for a certain Abilene traffic matrix, a brute-force search revealed only 0.05% of 3-failure scenarios achieved $MLU > 1$, while 0.08% cases achieved $MLU > 0.8$.

We use a branch and bound algorithm leveraging our RLT LP relaxation. At each exploration step, the failure status of a subset of links is fixed at each node (in the initial step, none of the links are fixed), and the relaxation LP is run to determine a (possibly fractional) solution that results in the highest MLU for the LP. The link with the highest fractional failure (say $\langle i, j \rangle$) is considered, and the LP is rerun fixing x_{ij}^f as each of 0 and 1. Branches where the $MLU < 1$ are pruned. Of the remaining candidate unexplored nodes, the node with the highest MLU is visited. Ties are broken by picking the node at the lowest level in the search tree. The process is

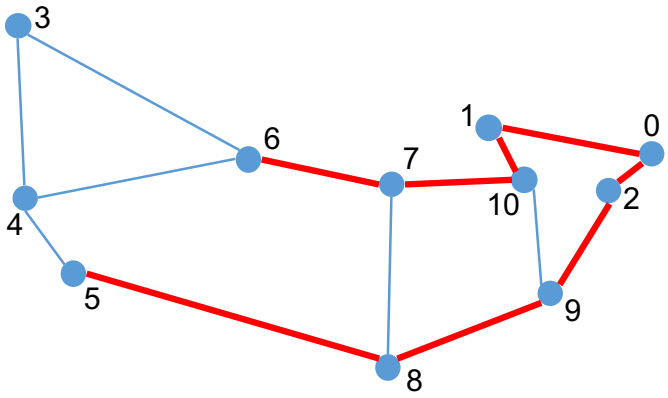


Fig. 2.7.: Abilene with links augmented shaded in red.

run until an integral solution is found, and the search procedure could be continued to determine multiple integral solutions. If the LP relaxation is tight, the search procedure solves at most as many LPs as the number of edges in the topology to find a failure scenario that results in the highest MLU, and our empirical experiments show it takes much fewer steps in practice.

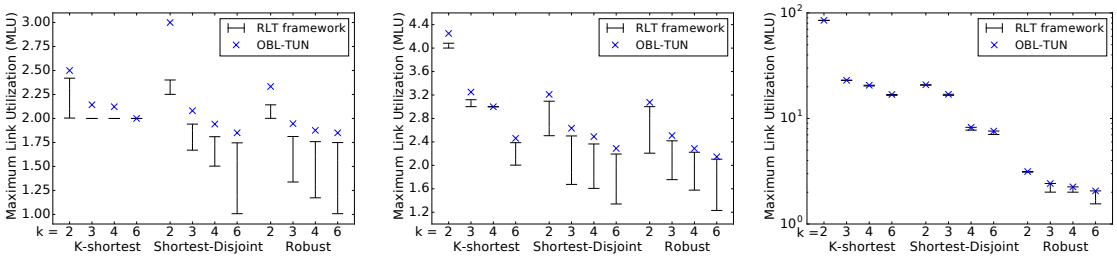
Emulation on an SDN testbed. We emulated the Abilene topology on Mininet [64]. Traffic was generated using the Ostinato traffic generator [65], and an actual Abilene traffic matrix snapshot. We used the procedure above together with our validation framework to identify multiple failure scenarios where MLU exceeded 1. Figure 2.6 presents measured Round Trip Time (RTT). Each curve corresponds to a failure scenario, and shows a CDF of the median RTT across all source-destination pairs for that scenario. The three curves to the right (black) represent failure scenarios identified by our framework with $MLU > 1$. To contrast, we show three other randomly generated 3-link failure scenarios with lower MLU (red, and to the left – note the curves overlap). The results illustrate that RTTs are significantly higher for the high MLU scenarios identified by our framework.

2.6.3 Deriving valid capacity augmentations

Our robust validation framework also guides operators in how best to augment link capacities to guarantee $MLU < 1$ across failure scenarios. As discussed in §2.5, our framework can be applied in an iterative approach that achieves optimal, or may be formulated as a single LP that does not guarantee optimality but solves efficiently.

Table 2.3 illustrates the iterative procedure for an Abilene traffic matrix under three simultaneous link failures. Recall that each iteration consists of (i) a validation step, which either certifies $MLU \leq 1$ for the topology (augmented by capacity increase suggested in prior iteration), or identifies a violating failure scenario; and (ii) an augmentation step, which identifies minimum capacity augmentation needed to handle all failure scenarios identified in prior iterations. The procedure terminates when the validation step certifies $MLU \leq 1$. The augmentation step is a small variant of (2.2). For a given scenario, the capacity augmentation problem is easy to model and solve as a linear program. Specifically, (2.2) is modified by setting the utilization bound $U = 1$, and replacing capacity c_{ij} with $c_{ij} + \delta_{ij}$, where δ_{ij} is the incremental capacity that must be added to link $\langle i, j \rangle$. The objective is $\sum_{ij} w_{ij}\delta_{ij}$, where w_{ij} is the cost associated with each unit of capacity added to link $\langle i, j \rangle$. The formulation is easily extended to multiple scenarios, by replicating the set of constraints (2.2) modified as above, for each scenario. Practical cabling constraints that constrain which links can have their capacity augmented and by how much are easily incorporated by adding bounds to δ_{ij}.

We have also formulated the problem as an LP with the stricter requirement that the RLT relaxation of the validation problem achieves $MLU \leq 1$ (§2.5). The LP achieves the same optimal augmentation as the iterative approach above, which is not surprising given that in all instances we have tried the integrality gap has been 1. More generally, the design LP yields an augmentation cost no worse than $\alpha\text{OPT} + (\alpha - 1)\text{BASE}$, where OPT is the optimal augmentation cost, BASE is the cost of the base network and α is the integrality gap of the RLT relaxation.



(a) Abilene (b) ANS (c) GEANT

Fig. 2.8.: MLU of RLT framework and Oblivious Tunneling (OBL-TUN) for different tunnel designs and topologies.

2.6.4 Validation across traffic demands

We next consider how our approach can validate that utilizations are acceptable across demands, focusing on the tunneling case study. For each topology, we consider tunnels pre-selected using the following strategies:

Non-robust strategies. These strategies pick tunnels without explicitly considering tolerance to a range of demands. Specifically, we consider: (i) *K-shortest:* Here, the K shortest paths between each source and destination pair are chosen. Prior works [4, 48] have used this approach to generate an initial candidate set of tunnels, and [4] ultimately picks a subset in a demand-sensitive manner; (ii) *Shortest-Disjoint:* Here, the shortest path is selected. Among other paths, one that overlaps the least with prior choices is selected in an iterative fashion. Combining path lengths and disjointness is a natural approach to tunnel selection [66].

Robust strategies. We also consider a heuristic called *Robust*, which derives tunnels by decomposing the optimal oblivious routing [67]. To generate a set of tunnels by decomposing the optimal oblivious routing, a derived graph is considered which has the same nodes and edges as the original topology, but with each edge having a weight equal to the flow from the oblivious routing. The widest path (the path with the highest bottleneck link capacity) is chosen as a tunnel. The bottleneck capacity of this path is now decremented from all other edges on this path in the de-

rived graph. This procedure is repeated until k tunnels are obtained. Since oblivious routing derives an MCF that performs well across all demands, tunnels derived from such a flow have the potential to perform well across demands.

For each tunnel selection approach, our goal is to determine MLU with an adaptive strategy, where traffic is split optimally across tunnels for each demand by solving (2.3). Since the associated validation problem is non-linear, we obtain bounds on MLU using (i) our RLT-based framework and (ii) an *Oblivious Tunneling* formulation (abbreviated as OBL-TUN) which minimizes MLU across all demands under the constraint that the fraction of traffic on each tunnel cannot vary with demand (§2.4.4). While both the RLT framework and OBL-TUN provide upper-bounds on the actual MLU, our framework can also be used to derive a lower bound. Specifically, we solve (V) after fixing the demand to the worst-performing demand for the RLT (or oblivious) relaxation. While this already provides a lower bound, we improve the initial lower bound using a local search procedure on (V), which involves alternating minimization on (v, λ) and x^d. These are tractable problems since (V) is linear if either (v, λ) or x^d are fixed.

Results. We evaluate a total of six schemes, combining our three tunnel selection heuristics with the two ways to obtaining bounds on MLU. For the set of demands, we consider all demands that can be routed with given capacities (§2.4.3). An MLU higher than 1 indicates the amount of over-provisioning required if tunneling were used to support all demands that the topology could handle with MCF routing.

Figures 2.8a, 2.8b and 2.8c present the MLU across all traffic demands for each of three strategies and three topologies, and different number of selected tunnels (K). Each cross shows the upper bound determined by OBL-TUN, while the vertical bar shows the upper and lower bounds obtained with our RLT-based framework. For GEANT, our current RLT implementation had a high memory requirement that can be addressed using standard decomposition techniques [68] in the future – hence we only report upper bounds achieved by OBL-TUN.

Table 2.3.: Iterative optimal capacity augmentation for Abilene (Figure 2.7). Each row shows MLU and counter example generated by the validation step, and the total capacity that must be added across all links as per the augmentation step to address all prior counter-examples. H (F) indicates one (both) sub-link(s), (each initially 5 Gbps) associated with the edge fails.

Step	Counter Examples	MLU	Total New Capacity
1	$(1, 10, H)$, $(2, 9, F)$	1.274	2.744 Gbps
2	$(2, 9, H)$, $(1, 10, F)$	1.274	5.488 Gbps
3	$(9, 8, H)$, $(10, 7, F)$	1.217	7.653 Gbps
4	$(10, 7, H)$, $(9, 8, F)$	1.217	9.818 Gbps
5	$(0, 2, H)$, $(1, 10, F)$	1.192	11.743 Gbps
6	$(1, 0, H)$, $(1, 10, F)$	1.071	12.452 Gbps
7	$(7, 6, H)$, $(8, 5, F)$	1.006	12.509 Gbps
8	$(8, 5, H)$, $(7, 6, F)$	1.006	12.566 Gbps
9	–	1.000	–

Several points can be made. First, our RLT framework often obtains tighter upper bounds than OBL-TUN strengthening Proposition 2.4.2. For example, for Abilene with $K = 2$, and Shortest-Disjoint tunnel selection, the upper bounds with OBL-TUN and the RLT framework are 3 and 2.4 respectively. Second, by providing lower bounds as well, our framework can exactly solve the non-linear problem (V) in quite a few cases. For instance for Abilene and the K-shortest heuristic, a single horizontal line is shown for $K = 3$ and higher, indicating that our framework can determine the optimal MLU.

Third, through a combination of lower and upper bounds, our framework can provide valuable insights on tunnel selection heuristics used by system practitioners. For example, for K-shortest $K = 6$, not only does our framework determine exact MLU, but also the MLU is the same as OBL-TUN. This indicates that when tun-

nels are selected using K-shortest, adapting how traffic is split across tunnels with demand performs no better than a non-adaptive approach. The trend is particularly pronounced for GEANT where our framework indicates the lower-bounds on MLU are higher than 16 even for $K = 6$, and very close to the oblivious solution. While recent work has suggested picking the K shortest tunnels and then picking a subset in a demand-sensitive manner [4], this result shows the possibility for this heuristic to perform poorly under certain demand patterns. The Shortest-Disjoint heuristic performs much better for Abilene and ANS, but performs poorly for GEANT – for $K = 6$, the lower bound is 7.05, close to the MLU of 7.59 with an oblivious approach.

While the non-robust design strategies perform poorly, *Robust* performs much better. The benefits are particularly stark for GEANT, e.g., for $K = 6$, the MLU ranges between 1.54 and 2.05. We have also experimented with robust tunnels and predicted demands obtained from real traffic matrices that are scaled so as to stress the Abilene network. The RLT framework achieves the optimal MLU (Proposition 2.4.3). OBL-TUN however results in MLU that is 6.84 times worse than optimal. Overall, these results show the value of our RLT-based framework.

2.7 Related work

Like work on network verification (e.g., [31, 32]), robust validation ensures that network designs meet operator intent. While verification efforts have focused on correctness of the network data-plane, and switch configurations, robust validation is an early attempt at verifying quantifiable network properties. Our framework complements topology synthesis tools [29] by allowing specification of robust design requirements, and providing the underlying optimization substrate.

Prior work on traffic engineering has focused on adaptive settings [69, 70] or has derived a robust routing that optimizes for multiple demands assuming that the routing does not change across demands [23, 39, 42, 71]. Robust routing schemes include oblivious schemes which do not use prior traffic data (e.g., [23,39]), that route

based on multiple historical traffic matrices (e.g., [71]), and those that combine these techniques [42]. Oblivious schemes arose from pioneering work in the theoretical computer science community [40, 41]. In contrast, we obtain worst-case utilization bounds for network designs, where topology and tunnels are invariant, but routing may adapt in practical yet richer ways. It has been shown that adaptive tunnels may, in the worst-case, not benefit much relative to oblivious routing [67]. Instead, we show that provable gains are achieved for specific topologies which have also been observed in practice [72].

Several works have looked at traffic engineering in the presence of failures [18, 22, 23, 28], and we have extensively compared our work with [28]. [22, 23] studied partial adaptation to failures as a way to balance flexible adaptation with the cost for adaptation. [18] optimizes bandwidth assignments to flows, guaranteeing that no congestion occurs with failures. While we do not elaborate, this model can be expressed using our framework. Prior work [25] developed ways to choose OSPF weights which are robust to single link failures. In contrast, we allow flexible adaptation, minimize MLU, and aid robust design of networks that cope well with failures.

Many recent works have looked at how traffic must be routed in the presence of middleboxes (e.g., [50, 51, 54]). There is a growing trend for virtualization of middleboxes, which may allow placements to change on the fly [51, 73]. Our framework can accommodate problems that adapt routing to handle uncertain demands/failures while satisfying middlebox constraints both for fixed placements, and when allowing placements to adapt along with routing.

Beyond networking, the complexity status of robust optimization formulations has been investigated and tractable formulations derived for various special cases [43, 44]. Recent literature has considered limited adaptability in robust binary programming applications including supply chain design and emergency route planning [74, 75]. Instead, our work considers more general forms of adaptivity, focuses on the networking domain, and brings relaxation hierarchies from non-convex optimization to bear on robust optimization problems.

2.8 Conclusions

In this chapter, we have made three contributions. First, we have presented a general framework that network architects can use to validate that their designs perform acceptably across a (possibly exponential and non-enumerable) set of failure and traffic scenarios. Second, by explicitly modeling richer ways in which networks may adapt to failures, and traffic patterns, we have obtained tighter bounds on MLU than current theoretical tools, which consider more limited forms of adaptation for tractability reasons. Third, we have demonstrated the practical applicability of our framework. While the first-level RLT can provably solve the validation problem for predicted demand, surprisingly, it also determines optimal MLU for for all our experiments with the failure case study. Empirical results confirm that our techniques consistently out-perform oblivious methods that can be unduly conservative. Finally, our framework can enable operators to understand performance under failures, guide incremental design refinements, and shed new light on commonly accepted design heuristics. Our initial results encourage us to explore larger networks, study the quality of bounds on other validation problems, and consider network design more extensively in the future.

3. GENERALIZED ROBUST NETWORK VALIDATION

In the previous chapter, we conducted two case studies in robust network validation: f simultaneous link failures and variable demands. In this chapter, we will explore a wider set of case studies, including shared-risk link group (SRLG) failure, heterogeneous link failure, partial link failure, route restriction, and multiple traffic classes. In the wide-area network context, the common sources of failures are IP links, IP routers, and optical related failures [8–10, 76]. The last two categories can simultaneously impact multiple links, and constitute an SRLG [8, 22]. Failures of different SRLGs are typically independent [8]. Partial link failures may impact part of the capacity of an IP link, and may occur because each IP link is usually provisioned as multiple sub-links with different failure modes (e.g., attached to different router line cards). Finally, failure probabilities may vary across devices and links [8–11, 76], owing to the underlying technology, whether a link is terrestrial or trans-oceanic, and possibly age of equipment. We will detail the modeling of different failure patterns in §3.1.

In practice, certain routing in a network may not be allowed. For instance, in a network comprising of core routers and edge routers, network architects may impose a policy that traffic sourced from edge router s and destined to edge router t must not traverse a third edge router, i.e., traffic entering into an edge router from core routers must not go back to core routers. We will illustrate how to integrate route restrictions into modeling network response in §3.1.

Additionally, previous network response model focuses on using the utilization of the most congested link (i.e., Maximum Link Utilization, or MLU) as the performance metric, and the traffic matrix belongs to a single traffic class, whereas in this chapter we discuss the network performance using a throughput metric that we define later. To study all these variants of network robust validation, we propose a generalized

network model (§3.1), which is capable of describing the aforementioned different failure patterns, route restriction, and multiple traffic classes, in a single model.

The previous chapter also leaves two questions open: (i) How effective is robust validation in providing tight bounds under different failure and traffic models? (ii) Deriving RLT is manual and error-prone. Can the process of conducting RLT be automated to enable experiments on richer case studies? This chapter resolves these two questions in the following sections.

3.1 Network model

Recall that in §2.3.1, we define a robust network validation problem as the following, where X denotes the uncertainty set, $Y(x)$ denotes the network adaptation. i.e., the routing permitted, and $F(x, y)$ stands for the performance metric. Note that we use *min max* here (compared to *max min* in the previous chapter), because we now optimize for throughput performance metric instead of MLU in the previous chapter.

$$F^* = \min_{x \in X} \max_{y \in Y(x)} F(x, y) \tag{3.1}$$

To study robust network validation problem under richer failure patterns, route restriction, and multiple traffic classes, we propose a generalized network model called *Generalized Maximum Concurrent Flow* (GenMaxCF). It is a generalization of the well-known Maximum Concurrent Flow (MaxCF) model [77]. The MaxCF problem maximizes throughput (the ratio of the flow between a pair of entities to the predefined demand for that pair, which must be the same for all pairs), subject to capacity constraints. In this chapter, we focus on the GenMaxCF model because it provides more flexibility in terms of how traffic can be routed, captures richer failure models of interest to practitioners, and is feasible in SDNs which allow for flexible response. We show the difference between GenMaxCF and the network models in the previous chapter in Table 3.1. Moreover, we will illustrate that GenMaxCF provides tighter bounds than the network model in the previous chapter in Section 3.1.1.

Table 3.1.: Differences between Generalized Maximum Concurrent Flow (GenMaxCF) model and model (F) (§2.3.3).

Network models	GenMaxCF	Model (F), §2.3.3
Uncertainty set	(Partial) link failure, SRLG, or heterogeneous link failure	Link failure
Adaptation	Edge-core route restriction	Multi-commodity flow
Metric	Throughput	Utilization of most congested link
Number of traffic classes	Multiple	Single

3.1.1 Generalized Maximum Concurrent Flow model

Given demands for a set of traffic classes, GenMaxCF models network response as seeking to determine how to route each flow, so all traffic of higher priority classes can be accommodated, while maximizing the fraction of demand for a lower priority class that can be met. This models SDN controllers [3, 4] that prioritize latency-sensitive traffic and throttle elastic traffic.

Consider a network with link set E, where each link $\langle i, j\rangle$ has n_{ij} units of bandwidth with each unit having capacity c_{ij}. Assume x^f_{ij} ($\leq n_{ij}$) units of that link have failed. Further, let there be M classes of traffic, where d^k_{st} denotes the total traffic from source s to destination t for class k (lower k denotes higher priority). The network determines r_{ijt} (the traffic destined to t over link $\langle i, j\rangle$ across all classes and sources) so that the highest fraction, Z, of class k traffic is handled besides all traffic

from higher priority classes. The network thus solves $\max_{y=(r,Z)\in Y(x)} Z$, where $Y(x)$ is as below:

$$\begin{aligned} & c_{ij}(n_{ij} - x^f_{ij}) \geq \sum_{t,\langle i,j,t\rangle \in S} r_{ijt} \quad \forall \langle i,j\rangle \in E \\ & \sum_{j,\langle i,j,t\rangle \in S} r_{ijt} - \sum_{j,\langle j,i,t\rangle \in S} r_{jit} \geq \sum_{k=1}^{M-1} d^k_{it} + Z d^M_{it} \quad \forall i,t, i \neq t \\ & r_{ijt} \geq 0 \quad \forall \langle i,j,t\rangle \in S \end{aligned} \tag{3.2}$$

The first constraint models that traffic on a link does not exceed capacity (accounting for failures). The second constraint models flow balance. The set S captures routing restrictions that limit links $\langle i,j\rangle$ which can carry traffic destined to t. For instance, consider backbone routers in a PoP, connected by a set of edge routers. Typically, traffic from a core router i is forwarded to an edge router j only if j is the destination. The model may return a negative Z if traffic of classes $M-1$ and lower cannot be handled. The formulation may be iteratively used with a binary search procedure to identify the lowest priority class that can be handled.

Reformulation. Applying the similar reformulation technique described in §2.4.2, we reformulate GenMaxCF introducing variables to model capacity reservation along alternate routes for links that may fail. Then we dualize the modified model resulting in the following non-linear minimization problem:

$$\begin{aligned} (R) \quad & \min_{v,x} \sum_{\langle i,j\rangle \in E} v_{ij} c_{ij}(n_{ij} - x^f_{ij}) - \sum_t \sum_{i,i\neq t} \sum_{k=1}^{M-1} d^k_{it} v_{it} \\ \text{s.t.} \quad & \sum_{t,i\neq t} d^M_{it} v_{it} = 1 \\ & v_{it} - v_{jt} \leq v_{ij} \quad \forall \langle i,j,t\rangle \in S \\ & v_{it} \geq 0 \quad \forall i,t, i \neq t \\ & x \in X \end{aligned} \tag{3.3}$$

Here, we use x to denote failure variables associated with links, and more generally SRLGs. Further, v are dual variables, and $x \in X$ represents a set of constraints that model the failure set X, which we detail in Table 3.2.

Table 3.2.: Constraints for various failure models.

Homogeneous link model; f links fail	$\sum_{\langle i,j\rangle \in E} x_{ij}^f = f$ $x_{ij}^f \in \{0,1\} \quad \langle i,j\rangle \in E$
Heterogeneous link model; L link categories; f_k links fail in category L_k	$\sum_{ij} x_{ij}^f \leq f_k \;\; \langle i,j\rangle \in L_k$, $1 \leq k \leq L$ $x_{ij}^f \in \{0,1\} \quad \langle i,j\rangle \in E$
SRLG model f SRLG failures	$\sum_{k\in G} x_k^g = f$ $x_k^g \leq x_{ij}^f \quad k \in G, \langle i,j\rangle \in g_k$ $\sum_{k\in G, \langle i,j\rangle \in g_k} x_k^g \geq x_{ij}^f$ $x_k^g, x_{ij}^f \in \{0,1\}\ g \in G, \langle i,j\rangle \in E$
Partial link failure model f units fail across links	$\sum_{\langle i,j\rangle \in E} x_{ij}^f = f$ $x_{ij}^f \in [0, n_{ij}], x_{ij}^f \in \mathbb{Z}\ \langle i,j\rangle \in E$

Table 3.3.: Edge-core route restriction

Name	Definition
E	Set of all links
V	Set of all nodes
I	Set of core (internal) nodes
S_1	$= \{(i,j,t) \vert (i,j) \in E, t \in V, i \neq t\}$
S_2	$= \{(i,j,t) \vert (i,j) \in E, i \in I, j \in V - I, t \in V, j \neq t\}$
S	$= S_1 - S_2$: The set of (i,j,t) where (i,j) is allowed to carry traffic to t

Route restriction. Generally, route restriction can be expressed by the set of all permissible $\langle i,j,t\rangle$ pairs (set S in model (R)). For example, edge-core network routing restriction described in the beginning of this chapter is illustrated in Table 3.3.

Intractability of network validation. We first show that the problem of determining whether all scenarios in an uncertainty set X achieve acceptable performance when a centralized scheme is used (i.e., model (F) in §2.3.3) is NP-complete, a previously open problem [28].

Proposition 3.1.1 *Consider that the network response for any failure scenario is an optimal multi-commodity flow that minimizes MLU. Then the problem of determining whether the worst-case MLU over all f link failure scenarios is less than U^t (the desired performance threshold) is NP-complete.*

Proof sketch. The proof involves a reduction from the Cardinality Maximum Flow Network Interdiction Problem (CMFNIP) [78], where an enemy moves as much of a single commodity as possible from a node s to t in a network where edge $\langle i, j \rangle$ has capacity c_{ij}. An interdictor must decide whether there is a collection of R arcs in the network such that breaking them will ensure that enemy cannot transmit more than M flow. The CMFNIP problem is known to be NP-hard by a reduction from the MAX_CLIQUES problem [78].

Proof First, the problem (that we denote as RV) is in NP. Given a particular f link failure scenario, MLU can be checked to not exceed U^t by solving an LP, in polynomial-time, on a reduced graph with corresponding links deleted.

To show that RV is NP-hard, we reduce, to an instance of RV, the Cardinality Maximum Flow Network Interdiction Problem (CMFNIP) [78], where an enemy moves as much of a single commodity as possible from a node s to t in a network where edge $\langle i, j \rangle$ has capacity c_{ij}. An interdictor must decide whether there is a collection of R arcs in the network such that breaking them will ensure that enemy cannot transmit more than M flow. The CMFNIP problem is known to be NP-hard by a reduction from the MAX_CLIQUES problem [78].

Our reduction relies on the simple fact that scaling all the demands in a network increases the utilization of links by the same factor. Given a CMFNIP instance, we construct an instance of RV on the same graph where each edge retains the same

capacity, but the demand between s and t is scaled down to 1 while demand between remaining nodes is set to zero. Then, based on the observation above, the CMFNIP instance admits a flow of M if and only if the MLU of the corresponding RV instance, with R failures, is at most $\frac{1}{M}$. Therefore, to solve the CMFNIP instance it suffices to check the latter condition. ■

We next present results that shed light on the tractability of (R). We denote by R_{MH} the special case of R with f homogeneous link failures and MaxCF, a special case of GenMaxCF as network response. We have the following result.

Lemma 3.1.1 *Determining whether the worst-case throughput of $R_{MH} < Z$ is NP-complete.*

We omit proofs, but note that this claim follows since the throughput metric is the reciprocal of the utilization metric. We next present a stronger result for *the SRLG model*, where a failure results in all associated links of a group failing. We denote R with f group failures and MaxCF recourse as R_{MS}.

Proposition 3.1.2 *Finding whether the optimal value of $R_{MS} < Z$ is NP-complete even for the special case of a two-node undirected graph with parallel links.*

Proof The proof follows by a reduction from the set-cover problem. First, the decision problem is in NP. Given a particular k-group failure scenario, we consider the graph obtained by deleting links in all the groups, determine the MCF using a polynomial time algorithm, and verify that $R_{MS} < Z$.

To show it is NP-hard, we do a reduction from the set cover problem. Given a set of elements $\{1, 2, \ldots n\}$ called the universe N, and a collection S of subsets of this set, the set cover problem seeks to find if there exists a cover C such that $C \subseteq S$, $|C| \leq k$, and the union of all sets within C is N.

Given an instance of the set-cover problem, we construct an instance of the failure group problem as follows. Consider a topology with two nodes A and B, and n edges between them, with each link corresponding to one element in N. Each edge has

capacity c, and there is unit demand between the nodes. Further, for each subset S_i in S, define a failure group G_i which consists of all the links corresponding to S_i. Then, we show that a cover C with $|C| \leq k$ exists iff the topology can be disconnected under k simultaneous group failures (i.e., its utilization exceeds $1/c$ under failure).

Let's assume a cover with $|C| \leq k$ exists. The failure of all corresponding groups would disconnect the above topology. To prove the reverse direction, let's say the topology gets disconnected under the simultaneous failure of k groups. Take the corresponding sets in S. Their union is N, and the sets would constitute a valid cover. ■

Since (R) is intractable, we obtain a conservative (lower) bound by relaxing it into a linear program (LP) obtained by multiplying pairs of constraints and replacing the non-linear terms with new variables (§2.4.1, the first level of the RLT relaxation [46]). Next, we discuss how to improve bound quality by augmenting (R) with bound constraints on v variables and their products with constraints involving x^f.

Tightening bound quality. The intractability of (R) guided us in deriving even better bounds. For instance, for special graph constructions inspired by the NP-hardness proof of deterministic interdiction [79], we observed larger than expected gaps between the relaxation for R_{MH} and its true optimal. We discuss the details in the case study later this section. To address this, we show the following.

Proposition 3.1.3 *Augmenting R_{MH} with the constraint $v_{it} \leq 1/d^1_{min}$ $\forall i, t, i \neq t$, where $d^1_{min} = \min_{i,t}\{d^1_{it} \mid \forall d^1_{it} > 0\}$ does not alter its optimal value.*

When $d^l_{it} > 0$, the bound on v_{it} follows from the first constraint in (R). When there is a single traffic class, we show that if, $d^l_{it} = 0$, and the optimal v_{it} violates the bound, there exists an alternate optimal solution that satisfies the constraint. Finally, we show that R_{MH} along with the constraints in Proposition 3.1.3, yields a relaxation that (i) provides tighter bounds relative to R_{MH} for the special graph constructions (see the case study later); and (ii) works well in practice (see §2.6).

With an augmentation, Proposition 3.1.3 can generalize to models with two traffic classes, which we omit the proof.

Case study: topologies where the augmenting constraint is necessary. As discussed in Proposition 3.1.3, augmenting R_{MH} with an extra constraint tightens the bound quality. In this section, we demonstrate the necessity of the augmenting constraint in a set of specially constructed topologies, which is first described in [79].

The series of topologies are generated with two parameters, κ and μ [79]. An example of the topology when $\kappa = 2$ and $\mu = 4$ is shown in Figure 3.1. We have experimented the model (R) both with and without the augmenting constraint on various settings of κ and μ and different number of link failures, using a traffic matrix with only s sending a unit flow to t. The result is shown in Table 3.4. We observe that for the topology with $\kappa = 1$ and $\mu = 1$, even the results obtained with model (R) without the extra bounding constraint match the optimal. For the topology with $\kappa = 2$ and $\mu = 4$, starting from 2 link failures, the results obtained with model (R) without the bounding constraint are zeros, i.e., the relative gap is in fact infinity compared to the optimal value. Instead, augmenting model (R) with the bounding constraint tightens the bound quality to matching the optimal in all the cases except $f = 5$, where the optimal throughput is 4, and (R) with the augmenting constraint achieves 3. This case study demonstrates the effectiveness of tightening model (R) with the bounding constraint described in Proposition 3.1.3.

3.2 Toolkit

We have shown that GenMaxCF is a general network response model which applies to various failure models and network adaptations. Since the relaxation process will differ for each combination of failure model and network adaptation policy, it then becomes critical to automate the relaxation process, i.e., to automate RLT. We build a toolkit which highlights the following features: (i) automating the RLT process for different failure models and route restriction policies; (ii) providing user-friendly

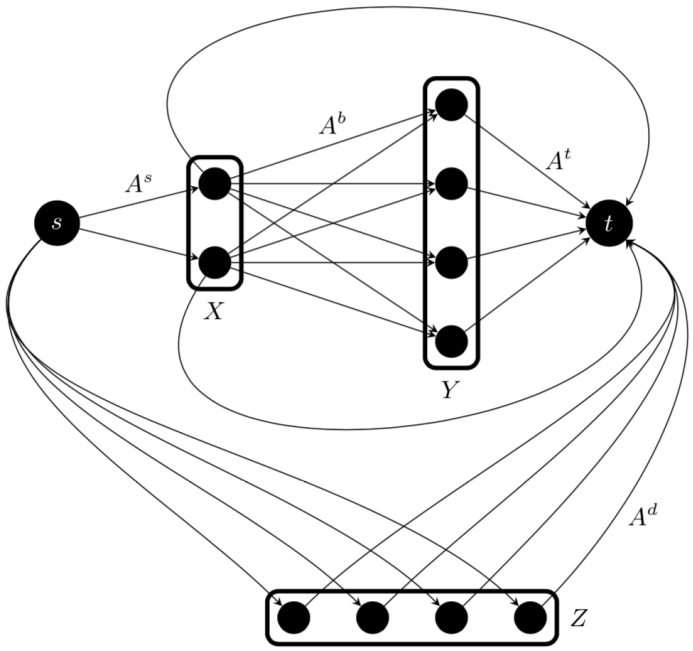


Fig. 3.1.: An example of the topology when $\kappa = 2$ and $\mu = 4$ (Figure 1 in [79]).

Table 3.4.: Throughput obtained with model (R), with and without the augmenting constraint in Proposition 3.1.3.

κ	μ	Number of failure	Optimal value	With bound	Without bound
1	1	0	2	2	2
1	1	1	1	1	1
2	4	0	70	70	70
2	4	1	54	54	54
2	4	2	38	38	0
2	4	3	22	22	0
2	4	4	6	6	0
2	4	5	4	3	0

APIs; and (iii) supports higher level of RLT. The toolkit is developed in Python

Table 3.5.: APIs provided by the toolkit

Definition	Effect
RLT_reformulate(model, constr_pairs, constraints)	Linearize and relax model
Linearize(expr)	Linearize the expression

leveraging the framework of Pyomo [80], but note that the idea of RLT automation is general and does not depend on specific programming languages or software.

3.2.1 APIs

We design user-friendly APIs, which take users' non-linear models as the inputs, and output the linearized and relaxed models using RLT (§2.4.1). The API definition is presented in Table 3.5. The users only need to call *RLT_reformulate* on their original non-linear models and will get the linearized and relaxed models processed by RLT. We rely on users to provide the original non-linear model complying with model (R), an array of constraint pairs which need to be taken products, and an array of all original constraints. The model provided by the user should comply to a canonical form: (i) all constraints must be written in the form of $left\_hand\_side \geq 0$; (ii) bounding constraints for all variables must be explicit. For instance, for a binary variable x, we must put both the bounding constraints $x \geq 0$ and $x \leq 1$ explicitly in the model, which can later be used in RLT. The APIs can also perform higher level of RLT in addition to the first level. For instance, the model could have constraints containing terms with degrees higher than two (e.g., $x^2y - x - y - 1 \geq 0$).

For concreteness, we present the following code snippet to illustrate how users can leverage these APIs to reformulate a simple non-linear model described in Equation 3.4.

$$\begin{aligned} \min_{x,y} \quad & xy - x + y \\ \text{s.t.} \quad & 0 \leq x \leq 3 \\ & 0 \leq y \leq 4 \end{aligned} \tag{3.4}$$

```
# Based on Python 2.7 and Pyomo 5.0
# Package importing omitted
# Original model definition is omitted. Assume we get an object called
# 'instance', which carries all the variables, constraints, and
# the objective of the formulation.

# Do RLT relaxation
all_constraints = [instance.eq1, instance.eq2,
                   instance.eq3, instance.eq4]
constr_pairs = [
    (instance.eq1, instance.eq3),
    (instance.eq1, instance.eq4),
    (instance.eq2, instance.eq3),
    (instance.eq2, instance.eq4)]
rlt.RLT_Reformulate(instance, constr_pairs, all_constraints)

# Create a solver
opt = SolverFactory(solver)
results = opt.solve(instance)
print results
```

In the code snippet, we can see that users only need to add a few extra lines to define the constraint pairs which they request to take products, and reformulation

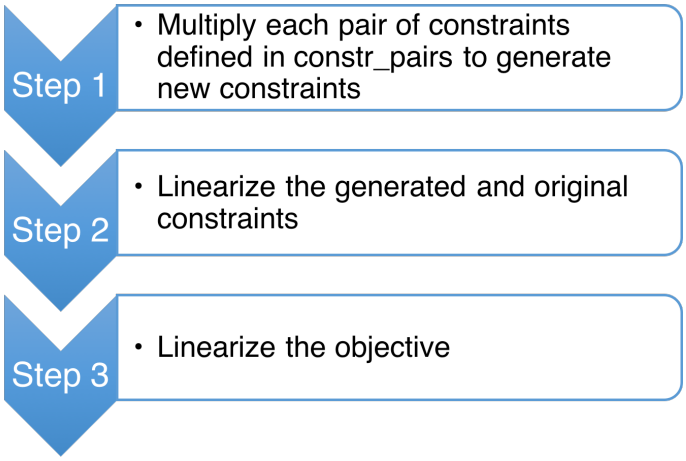


Fig. 3.2.: The process of RLT.

and linearization are done with the simple API call of $RLT\_Reformulate$. Under the hood, the `rlt` module conducts RLT process as depicted in Figure 3.2.

3.2.2 RLT automation implementation details

The key to RLT automation is how to efficiently multiply two constraints, because constraint multiplication may need to be performed millions of times during an RLT process. We refer to the vector formed by all the coefficients of a constraint left-hand-side expression as *coefficient vector*, and the vector formed by all the terms as *term vector*. Each constraint's left-hand-side can be viewed as the inner product of coefficient vector (v_c) and term vector (v_t). When we multiply two constraints, we actually perform vector multiplication. For instance, say we multiply $x + 5 \geq 0$ and $y - 3 \geq 0$. The coefficient vector of the first constraint is $v_c^1 = [1, 5]$, and the term vector is $v_t^1 = [x, 1]$. The coefficient vector of the second constraint is $v_c^2 = [1, -3]$, and the term vector is $v_t^2 = [y, 1]$. The product of these two constraints is $\sum_{i,j} M_{ij}^c M_{ij}^t \geq 0$, where $M^c = (v_c^1)^T v_c^2$, $M^t = (v_t^1)^T v_t^2$. This mathematical view is useful in improving the performance of RLT automation, because the essence of constraint multiplication is matrix multiplication, and all techniques that accelerates matrix multiplication can be applied to accelerate RLT automation.

We also need to maintain a symbol mapping from all the non-linear terms to linearized new terms. For instance, product of two terms v and x is mapped to a new linear term vx. When handling terms with degrees higher than 2, to avoid repetition, we need to first sort the symbol characters, and then look up the mapping. For instance, both the product of x and xy, and the product of x^2 and y should be mapped to a new linearized term xxy.

Limitations. From the example above we can see that the RLT automation toolkit is powerful: it takes the original optimization model defined by users, together with a few extra lines defining the constraint pairs to be multiplied, and will generate the model processed by RLT. Since this RLT automation implementation is mainly for a proof-of-concept, the major limitation is in scalability. The current implementation based on Python and Pyomo can scale up to a network with close to a hundred nodes and edges on a single 128GB memory machine. The networks beyond this size result in larger memory consumption which our servers cannot hold in a single machine. In the future, we can improve the scalability in the following potential ways: (i) Python and Pyomo by default does not optimize in memory usage. An alternative way of implementation may scale better, e.g., C and Gurobi C APIs; (ii) current implementation handles each constraint separately instead of grouping them when it is possible, e.g., $x_{ij} \geq 0, i, j \in \{0, 1, \ldots, 9\}$ are actually treated as 100 constraints, and the product of $x_{ij} \geq 0$ and $y_{ij} \geq 0, i, j \in \{0, 1, \ldots, 9\}$ needs 10,000 multiplications. Instead, we can leverage the constraint structure to do a symbolic constraint multiplication which will save large amount of computations and memory usage. For instance, in the example above, only one symbolic multiplication is needed to obtain $x_{ij}y_{i'j'} \geq 0, i, j, i', j' \in \{0, 1, \ldots, 9\}$; (iii) for even larger or more complicated networks, we can leverage distributed systems like MapReduce to perform constraint multiplication in a distributed fashion, because constraint multiplication has no correlation and can be done in parallel. However, note that all the relaxed constraints need to be reduced to one machine and solved in a single model, which may

sooner become a computation and memory bottleneck than performing constraint multiplication.

3.3 Evaluation

In this section, we evaluate the quality of bounds obtained with model (R) together with various failure models (Table 3.2), and the performance of RLT automation, using real topologies (Table 2.1) and synthetic traffic matrices generated from gravity model [37].

3.3.1 Quality of bounds for various failure models

Even though as we have shown in the case study (§3.1.1), model (R) plus the augmenting constraint may still have a gap from the optimal in some specially constructed topologies, the quality of bounds generated in practical topologies is surprisingly good. The results match exactly the optimal except SRLG failure model. We will next focus on how to certify SRLG failures, and potential approaches to bridge the gap.

3.3.2 Certifying SRLG failures

SRLGs typically occur owing to backbone router and/or optical equipment failures. Since we do not have access to failure data on how IP links map to the optical segments, we focus on backbone router failures. Mimicking realistic backbone network settings [5], we create a richer topology by (i) replicating the core network constituting two parallel networks; and (ii) connecting each edge router to two core routers in each replica.

The relaxations exhibited a significant gap when determining worst-case performance across all possible f SRLG failures for some of our topologies. The gap manifested even with a simple topology comprising a replicated hexagon core network,

with three edge routers. Investigating further, we were able to devise a way to reduce this gap. Specifically, we (i) limited the total number of SRLG failures in each topology replica to f_1 and f_2, where $f_1 + f_2 = f$; (ii) introduced constraints of the form $\sum_{k \in G_1} x_k^g = f_1$, and $\sum_{k \in G_2} x_k^g = f_2$, where G_1 and G_2 respectively denote the SRLGs corresponding to router failures in each of the topology replicas; and (iii) took products of these constraints with constraints involving dual variable v in (R). This approach completely eliminated the gaps for the hexagon construct, and significantly reduced them for our real topologies. For example, for the extended Abilene topology with 1 or 2 SRLG failures, there was no gap. For 3 simultaneous SRLG failures, the optimal Z was 0.9924, while our relaxation gave a conservative estimate of 0.9887. While this requires solving a series of LPs, for practical f, the number of possible splits (f_1, f_2) after eliminating symmetries is small. We note that this bound can be achieved using an IP with a few binary variables modeling the binary expansion of f_1, f_2. For example, the binary expansion of f_1 can be described as $f_1 = 1 + 2^1 b_1 + 2^2 b_2 + \ldots$ with binary variables b, and similarly for f_2.

3.3.3 RLT automation performance

Here we compare the performance of certifying GEANT networks with automated and manual RLT processes under different number of link failures. The CPU time is measured on a single-threaded 3.00GHz CPU. The result is shown in Figure 3.3, and we have the following observation: the performance of automated RLT is close to that of manual RLT, though automated RLT is about 23% to 40% slower. The performance hit of automated RLT is due to the following reasons: (i) automated RLT takes extra time to generate the relaxed constraints through constraint multiplication compared to manual RLT, which already contains those relaxed constraints input manually. The number of newly generated relaxed constraints is $O(|E|^2|V|)$, i.e., the larger the topology, the longer it takes in generating these constraints automatically. (ii) Our RLT automation implementation is based on Pyomo, which is an extra layer

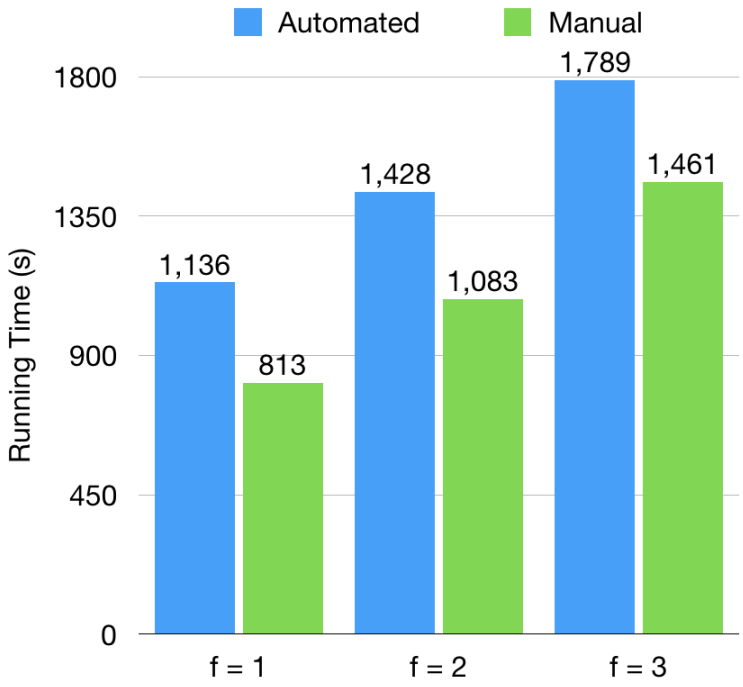


Fig. 3.3.: CPU Time comparison between automated and manual RLT under 1 to 3 link failures for GEANT network.

of indirection before solving the model in optimization solvers (e.g., Gurobi). On the contrary, the manual RLT model is directly implemented with Gurobi APIs. Therefore, the efficiency of algorithms and data structures in Pyomo will impact the performance of the automated RLT. To sum up, we have achieved reasonable performance for a proof-of-concept automated RLT implementation, and there are opportunities to further improve the performance leveraging the approaches described in §3.2.2.

4. SLICE: ANALYZING AND PROTECTING NETWORK PERFORMANCE UNDER FAILURES

4.1 Introduction

With the increasing adoption of online and cloud-based services, there is an ever growing requirement on the underlying network infrastructure to ensure that business-critical applications continually operate with acceptable performance [26, 27]. Typically, these requirements are expressed as service level objectives (SLO) [81]. SLOs not only require network connectivity, but also require that the promised bandwidth between data-center sites is available [2]. Networks must meet their performance objectives while coping with significant *uncertainty* in their operations. The global scale and rapid evolution of networks imply that failure is the norm in both cloud provider [2, 10–12] and ISP [8, 9] settings, and the complexity of failures is on the rise [12]. Failures occur due to both natural factors (hardware/software failures, or natural disasters [82]) and maintenance operations, and often involve multiple concurrent events [12, 83].

Recent trends exacerbate the challenges in planning networks for uncertainty. First, the increasing pressure to reduce costs, and the dramatic growth in traffic has motivated both online service companies [3–5], and more recently, large ISPs [6, 7] to operate networks closer to capacity, in contrast to the highly over-provisioned nature of networks in the past. Second, performance requirements are getting increasingly stringent. For instance, a recent paper from Google [2] indicates that not only has traffic increased by 100x over the last five years, but also bandwidth requirements must be met 99.99% of the time compared to 99% in the past.

The process of designing networks [29, 30] for such uncertainty, and certifying a network can tolerate a desired set of failures often requires extensive simulations

[26, 27]. Unfortunately, the space of possible designs is large, and exhaustive testing requires considering combinatorially many failure scenarios [18, 28, 84]. Consequently, architects today use ad-hoc techniques to verify compliance and to design networks, which may fall short of meeting performance requirements.

Motivated by these challenges, recent works have explored the use of robust approaches where the primary goal is to optimize the *worst-case performance* of a network across a range of possible failure scenarios [18, 22, 28, 84]. Unfortunately, while an important first step, worst-case design may be unduly conservative since a small number of bad failure scenarios may be expensive or even infeasible to design for. Further, worst-case approaches do not provide the architect with an understanding of the distribution of performance across failure scenarios, and an understanding of *which* scenarios and *what fraction* lead to unacceptable performance.

In this chapter, we present **Slice**, a formal framework for analyzing network performance (whether bandwidth requirements are met), and designing networks, for more general objectives than worst-case performance under failures. Specifically, we make the following **contributions**:

Failure classification. Slice *classifies* failure scenarios based on whether performance is acceptable or not. Slice efficiently handles the multitude of possible scenarios through a novel divide-and-conquer algorithm that partitions scenarios into sets, and analytically classifies entire sets as a whole. A highlight is that the algorithm compactly represents scenarios with acceptable performance as the union of a relatively small number of sets when possible, which can aid design.

Efficiently analyzing diverse network response. We have designed Slice for networks with (i) protection routing schemes, where local mechanisms are used to quickly reroute traffic on link failures [22, 28]; and (ii) an ideal centralized scheme [3, 4, 7] that optimizes traffic for each scenario. While protection routing schemes are reflective of performance achievable by a network in practice, the centralized mechanism establishes the best achievable performance. We develop an combinatorial polynomial-time algorithm to classify scenarios for protection routing schemes. For

centralized schemes, we show that the problem is intractable resolving an open question [28, 84]. Yet, we improve existing techniques to achieve better efficiency.

Generalized performance analysis under failure. Slice can certify that performance is acceptable for a desired percentage of scenarios, and generate *performance profiles*, i.e., determine the percentage of scenarios for which network performance (e.g., the fraction of lower priority traffic that can be sustained) reaches various thresholds. Further, Slice's classification algorithm may be combined with sampling techniques to scale to larger networks. The hybrid approach gains efficiency by limiting sampling to specific failure sets, and providing a better classification of failure scenarios.

Designing for more general objectives than worst-case. Existing works [18, 28, 84] design networks to perform acceptably over all scenarios involving f or fewer simultaneous failures. Slice supports design for *most* rather than *all* such scenarios. Doing so may be necessary since design for the worst case can lead to unacceptable, or overly conservative performance for the vast majority of scenarios. Yet, deciding which subset of combinatorially many failure scenarios to design for is challenging. We demonstrate Slice's ability to aid with such design problems in the context of protection routing. Slice's approach involves classifying failure scenarios that can be handled by a centralized mechanism, and restricting protection routing design to these scenarios. We develop generalizations of the protection routing model that supports design for arbitrary sets of scenarios while providing theoretical guarantees. Finally, we show how the compact representation of promising scenarios found by Slice's divide-and-conquer approach makes design tractable.

We evaluate Slice using multiple real network topologies with failure models inspired by studies on ISP and cloud provider networks [8–10, 12]. Our results show that (i) Slice is effective in classifying scenarios that can be supported by both protection routing, and centralized mechanisms, revealing large gaps in the set of scenarios that can be handled by the mechanisms; (ii) Slice effectively aids design for more generic objectives than the worst-case, and doing so offers substantial benefits over worst-

case design. Specifically, Slice leads to the design of new protection routing schemes that outperform existing ones, while achieving performance comparable to centralized mechanisms; and (iii) Slice performs well with reasonable certification time. Validations over an SDN testbed demonstrate the benefits of Slice's approach. Overall the results show the promise of Slice.

4.2 Slice network model

In this section, we discuss the performance metrics Slice analyzes, and how Slice models failures and network behavior.

Performance model. The primary performance metric that we consider in this chapter is ensuring that the promised bandwidth requirements between every pair of sites can be sustained. We focus on this metric given its common use in network SLOs [2]. Much of our analysis is based on the utilization of the most congested link (henceforth referred to as Maximum Link Utilization or MLU). The network meets its requirements if MLU < 1. We also consider contexts with multiple traffic classes where the goal is to meet all traffic of higher priority classes (e.g., latency sensitive traffic), and as much of the lower priority (e.g., bulk transfer) traffic as possible. Slice focuses on whether desired performance can be achieved in the steady state in any scenario. While Slice does not model transient network performance, Slice can analyze routing schemes [18, 28] that respond fast to failures.

Modeling failures and uncertainty. In Slice, each scenario represents the network experiencing a given demand, and zero or more simultaneous failures. In the wide-area context, the common sources of failures are IP links, IP routers, and optical related failures [8–10, 76]. The last two categories can simultaneously impact multiple links, and constitute a shared-risk link group (SRLG) [8, 22]. Partial link failures may impact part of the capacity of an IP link, and may occur because each IP link is usually provisioned as multiple sub-links with different failure modes (e.g., attached to different router line cards). Each failure itself could correspond to an

SRLG, IP link or a partial link failure (§4.3.5). Beyond failures, Slice is designed to work with other forms of uncertainty regarding network state such as traffic demand that can be modeled, or conservatively estimated using a discrete distribution (§4.3.5).

Modeling diverse network response schemes. Slice models diverse ways in which networks may respond to failures while taking performance into account. For concreteness, we focus on (i) an ideal centralized approach that achieves the best possible performance under failures by optimally re-routing traffic; and (ii) a *link-based protection* scheme that ensures the network is congestion-free on failures (e.g., [28]). Though we do not consider in this chapter, Slice can also analyze path-based protection schemes (e.g., [18]) as we discuss in §4.6. By modeling ideal centralized response, Slice allows architects to compare the performance achieved by a specific routing algorithm with the performance that the network is intrinsically capable of achieving.

4.2.1 Generalized protection routing model

We next discuss Slice's model for link-based protection, which generalizes the state-of-the-art scheme [28].

With link-based protection, traffic on a link l is re-routed upon its failure, along pre-computed detour paths. To achieve this, an offline procedure is used, which for each link $l = \langle i, j \rangle$, reserves bypass capacity, not used until l fails, along paths that are disjoint from l and can carry flow from i to j. When l fails, the network uses this reserved capacity to re-route the traffic on l and executes an efficient online procedure to compute the changes required should another failure occur.

Consider a network with nodes V and edges E for a traffic matrix d. We present below an offline linear program (LP) that determines an optimal protection routing when the network must be protected against all possible scenarios in a set (henceforth

referred to as an SSet) X. The relevant parameters and variables are defined in Table 4.1, The LP minimizes the MLU U across all the scenarios in X.

$$
\begin{aligned}
&\text{(H)} \min_{r,p,a,U} \quad U \\
&\text{s.t.} \quad r_{st} \text{ is a unit flow from } s \text{ to } t. \quad \forall s,t \in V \\
&\qquad p_l \text{ is a flow of } a_l \text{ from } i \text{ to } j. \quad \forall l \in E, l = \langle i,j \rangle \\
&\qquad \forall x \in X, e \in E, \\
&\qquad\qquad \sum_{s,t} d_{st} r_{st}(e) + \sum_{l \in E} x_l p_l(e) \leq U c_e (1 - x_e) + a_e x_e \qquad (4.1) \\
&\qquad a_e \geq 0 \quad \forall e \in E; \quad U \geq 0
\end{aligned}
$$

The LP solves the variables r, p, and a, where r represents routing under normal condition, p represents protection routing, and a represents reserved capacity. Note that the routes protecting against the failure of e should not traverse itself, i.e., $p_e(e) = 0$. The first two constraints which capture standard flow balance constraints for r_{st} and p_l are detailed as the following. r_{st} is a unit flow from s to t. Note that we define $e = \langle i', j' \rangle$, and use notation $r_{st}(e)$ and $r_{st}(i', j')$ interchangeably.

$$
\begin{aligned}
&\forall i' \in V, \\
&\sum_{j';\langle i',j' \rangle \in E} r_{st}(i',j') - \sum_{j';\langle j',i' \rangle \in E} r_{st}(j',i') = \begin{cases} 1 & i' = s \\ 0 & i' \neq s, i' \neq t \\ -1 & i' = t \end{cases} \qquad (4.2) \\
&r_{st}(i',j') \geq 0 \quad \forall i', j'; \langle i', j' \rangle \in E
\end{aligned}
$$

Similarly, we can define p_l by a simple rewriting of (4.2): replacing $r_{st}(i', j')$ with $p_l(i', j')$, replacing the right-hand-side of the flow balance equation with a_l, 0, and $-a_l$, and replacing (s, t) with $l = \langle i, j \rangle$. (4.1) captures the capacity and reservation constraints of all links must be met under all possible failures $x \in X$. The two terms on the left-hand-side indicate the total traffic that link e must carry, which includes (i) the traffic under normal conditions; and (ii) the excess traffic owing to

other link failures. The right-hand-side captures (i) link e carries at most Uc_e traffic when e is operational; and (ii) when e fails ($x_e = 1$), a reserved capacity of at most a_e is available along bypass paths. As presented, (4.1) is enumerated over all possible scenarios, leading to exponential possible constraints. Instead, similar to [18,28], (4.1) is reformulated by relaxing the integrality requirements on x variables, and leveraging LP duality.

Converting (H) to an LP with duality. A key difficulty in translating (H) into an LP is that the obvious strategy would create one constraint per failure scenario. This is not scalable since X may have exponentially many failure scenarios. Instead, this is tackled by equivalently rewriting the capacity constraints as

$$\max_{x \in X} \quad \sum_{l \in E} x_l q_l(e) \leq U c_e - \sum_{s,t} d_{st} r_{st}(e) \quad \forall e \in E \tag{4.3}$$

where $q_e(e) = Uc_e - a_e$, and $q_l(e) = p_l(e), l \neq e$. Each of these constraints is reformulated by (i) relaxing the integrality requirements on x variables, resulting in an LP; (ii) expressing the left-hand-side as a minimization problem leveraging LP duality. When X has integral corner points, we can relax the integrality of x variables without affecting the maximum. Then, we take the dual of the resulting maximization problem. If X is modeled using m constraints, this adds only m variables and E constraints to the problem for each $e \in E$, in contrast to the obvious strategy mentioned above. For the procedure $DoAllCertify(A)$, discussed in §4.3, the above property is true on every set A that this procedure is called on.

Generalizations over state-of-the-art. Instead of a_e, R3 [28] reserves a capacity of c_e and expresses the right-hand-side of (4.1) as Uc_e by assuming that $Uc_e x_e \leq c_e x_e$, which holds only if $U \leq 1$. As a result, R3's formulation is only valid if $U \leq 1$, an issue that does not arise with (H), and is conservative relative to (H) since it fixes the reserved capacity at c_e instead of choosing it optimally. For instance, for the Abilene network [85], and an example traffic matrix [36], the MLU with R3 when protecting against all two failure scenarios (each failure impacting 50% of link capacity) is 0.56. In contrast, the MLU with (H) is 0.12 (the optimal achievable if the network could respond ideally). (H) allows us to determine the fraction

Table 4.1.: Symbol table.

Symbol	Definition
c_e	Capacity of link e
d_{st}	Traffic from s to t
a_l	Reserved bypass capacity for link l
x_e	Failure status of link e
$r_{st}(e)$	Fraction of s to t traffic on link e (no failure)
$p_l(e)$	Reservation on link e to handle link l failure

of traffic that may be carried across all failure scenarios, since this shares an inverse relation with U. We have also generalized R3 to design for more general failure sets and not just scenarios involving f simultaneous failures (§4.3.4), and have developed a variant of (H) which allows reasoning about multiple classes of traffic (§4.4.4).

4.3 Slice design

In this section, we present a design of Slice. We begin with an algorithm that allows Slice to efficiently classify scenarios into those that perform acceptably and those that do not (§4.3.1). The algorithm is generic to any form of network response but relies on oracle procedures. We discuss the design of these oracle procedures for protection routing and centralized mechanisms in §4.3.2 and §4.3.3, respectively. We then discuss how Slice enables design for more general objectives than the worst-case (§4.3.4), and other generalizations (§4.3.5).

4.3.1 Classifying scenarios compactly

We discuss Slice's classification algorithm which generates a compact representation of scenarios with acceptable performance as the union of a relatively small number of sets when possible. A naive approach is to enumerate all scenarios. This is compu-

Algorithm 1: Slice classification.

```
Function Slice(target)
    SSets ⟵ [F_f, F_{f−1}, ..., F_1, F_0]
    pass_set, fail_set ⟵ ∅, ∅
    while SSets do
        A ⟵ SSets.pop()
        Classify(A, pass_set, fail_set, SSets)

Function Classify(A, pass_set, fail_set, SSets)
    if DoAllCerify(A) then
        pass_set.add(A)
    else if DoAllViolate(A) then
        fail_set.add(A)
    else
        A_1, A_2, ..., A_n ⟵ Partition(A)
        SSets.extend([A_1, A_2, ..., A_n])
```

tationally prohibitive given the large number of scenarios, and results in acceptable scenarios being represented as the union of a large number of sets which impacts the tractability of design (§4.3.4). Instead, Slice employs a divide-and-conquer algorithm (Algorithm 1), that classifies an entire set of scenarios to the extent possible, and when necessary, partitions the set further.

Let A represent an SSet. Slice uses the oracle procedures *DoAllCertify(A)* and *DoAllViolate(A)* which respectively determine whether *all scenarios* in A meet or violate the performance requirements. Ideally the procedures are exact (i.e., they return True if and only if all scenarios meet or violate requirements). For tractability, Slice only requires exactness when the SSet consists of a single scenario, and permits false negatives otherwise. Specifically, if A consists of multiple scenarios. *DoAllCertify(A)* may return False even if all scenarios certify, but must return True only if the condition is met. Likewise, *DoAllViolate(A)* may return False even when all scenarios violate the requirements.

If either procedure returns True, the status of SSet A is resolved. In general, neither of these procedures may return True, since performance may be acceptable for some scenarios in A, but unacceptable for others, and since the procedures may have false negatives. In such a case, the algorithm partitions A further into two or more disjoint and complementary subsets using the procedure *Partition(A)*, and the process is repeated on each smaller SSet. The algorithm converges since the oracle procedures must be exact for SSets containing single scenarios.

For concreteness, considering the link failure model (more general failure models are discussed in §4.3.5), Slice begins by creating SSets of the form $F_0, F_1, \ldots, F_f$, where F_m comprises all failure scenarios with exactly m links failed. At any stage, Slice picks an SSet whose status is not yet resolved. For $i < j$, Slice explores F_i before F_j, because scenarios in F_i are more likely to occur and have acceptable performance. If performance is neither certifiable nor violating for the chosen F_m, the *Partition* oracle procedure chooses a link l, and partitions F_m into two subsets: (i) scenarios with exactly m links failed including l and (ii) remaining scenarios. We discuss how

to find a good choice of l later. If necessary, each subset is partitioned further by fixing the status of additional links. The process continues until all scenarios in F_m are classified, after which Slice similarly handles F_{m+1}. The algorithm terminates when all scenarios with f or fewer failures have been classified, or earlier if the goal is to check that a desired percentage of scenarios certify (§4.3.5). Figure 4.1 in §4.4.1 illustrates an example search tree generated by Slice classifying a 2-failure SSet.

We present a theoretical bound on the number of SSets inspected by the algorithm, which impacts its performance.

Proposition 4.3.1 *Assume that there are s violating scenarios within F_f, where F_f are scenarios that have f failures, and that DoAllCertify(A) is exact (i.e., no false negatives). Then, Slice solves at most $2sE + 1$ nodes while exploring F_f, where E is the number of branching variables, which is the number of links if branching is on link state.*

Proof sketch. Assume a set of 0-1 points (failure scenarios with f failures), C, form the corner points of a set defined by linear constraints and the set V of 0-1 points (violating scenarios) has cardinality s. Consider the divide-and-conquer approach used by Slice. Since an LP could be solved at the parent node, there are no siblings that only contain scenarios outside of V. Hence, at any level there are at most s nodes that contain scenarios in V, and s siblings that do not contain such scenarios. Including the root node, there are a total of $2sE + 1$ nodes. The argument can also be extended to address partial failures, where each link $\langle i, j\rangle$ may be viewed as consisting of n_{ij} sub-links, any of which could fail leading to partial capacity loss. The only difference is that there may be more siblings that do not contain scenarios in V. In this case, the tree may contain at most $1 + s\sum_{\langle i,j\rangle}(1 + n_{ij})$ nodes.

4.3.2 Tailoring Slice for protection schemes

We next discuss how the oracle procedures described above are realized for link protection schemes. Consider a protection routing with parameters r^*, p^*, and a^* that

achieves utilization U^* derived using (H) for an SSet X (§4.2.1). We use the $*$ notation to refer to a specific protection routing rather than variables to be determined. Slice may be used to determine scenarios for which the MLU is under a desired threshold U^t for this routing. For example, U^* maybe unacceptably high, and an architect may wish to determine a subset of X for which a tighter U^t threshold can be met.

DoAllCertify(A). The procedure determines whether the performance for protection routing (r^*, p^*, a^*) is acceptable for all scenarios in an SSet A. This requires verifying the capacity constraint (4.1) holds for (r^*, p^*, a^*) for every link e and every failure scenario $x \in A$, and for threshold U^t.

Performing this check requires iterating over every $x \in A$, possibly exponentially many scenarios. A potential approach to tackle this certification is to solve (H) but with p, r, and a fixed at p^*, r^*, and a^* and the failure set limited to only include scenarios in A. However, doing so would require solving an LP for each SSet inspected by Slice during the divide-and-conquer procedure.

Instead, to understand Slice's approach, note that the capacity constraint (4.1) can be equivalently rewritten as:

$$\max_{x \in A} \sum_{l \in E} x_l q_l^*(e) \leq U^t c_e - \sum_{s,t} d_{st} r_{st}^*(e) \quad \forall e \in E \tag{4.4}$$

where $q_e^*(e) = U^* c_e - a_e^*$, and $q_l^*(e) = p_l^*(e)$, $l \neq e$. Slice certifies the above for each edge individually by exploiting two observations: (i) x is the only variable in (4.4); and (ii) the maximum is obtained by fixing those x_l as 1, whose associated coefficients $q_l(e)$ are the highest, and permissible for A. Doing so also determines the worst scenario x that results in the highest utilization for e.

Motivated by these observations, Slice simply sorts $q_l(e)$ for each e and finds the appropriate worst failure scenario. Further, for efficiency, sorting is only performed once for each edge across all SSets in the divide-and-conquer procedure rather than for each SSet. With this, the worst-case time complexity of the algorithm is $O(|E|^2 log(|E|) + K|E|^2)$, where $|E|$ is the number of edges, and K is the number of nodes visited by Slice. For protection routing, the oracle procedure is tight, and

hence the bound on K shown in Proposition 4.3.1 is guaranteed to hold. While already promising, several optimizations can further reduce running time. For instance, for each SSet, only those edges must be certified that have not already been certified at an ancestor node.

DoAllViolate(A). Slice determines whether all scenarios violate by checking the contrary, i.e., whether there exists $x \in A$ for which performance is acceptable. To check this, Slice solves the following feasibility LP derived from (H):

$$\begin{aligned} & \min_{x} \quad 1 \\ \text{s.t.} \quad & \sum_{s,t} d_{st} r^*_{st}(e) + \sum_{l \in E} x_l p^*_l(e) \leq c_e(1 - x_e) + a^*_e x_e \quad \forall e \in E \\ & x \in A \end{aligned}$$

All scenarios in an SSet violate iff no feasible solution exists, in which case, Slice need not further explore that SSet. In contrast to the LP for worst-case performance, this LP is small and accounts for less than 1% of Slice's running time.

Finding link to fix at each step. If the network does not perform acceptably in all scenarios within SSet A, Slice partitions A into scenarios where link l fails (denoted by A_1), and where it does not (A_0). The choice of l only impacts the performance of the classification algorithm and not its correctness. Ideally, Slice may pick l such that one of the nodes A_0 or A_1 is found acceptable or violating so that the remaining node is the only one needing further exploration. Therefore, as a heuristic, Slice computes $\sum_{e \in E'} q_l(e)$ for each link l, where E' is the set of edges for which (4.4) is violated, and picks l for which the sum is the largest, and whose failure status is not already fixed in A. The intuition is to pick l that is part of the bad scenarios for many highly utilized edges e.

4.3.3 Tailoring Slice for centralized response

We next consider how Slice models and analyzes networks when they use a centralized scheme to respond to failures. While Slice can work with any centralized

scheme that responds optimally to failures, for concreteness, we focus on the case where a network may optimally re-route traffic to minimize the MLU by solving a multi-commodity flow (MCF) problem. While a centralized scheme may not necessarily be able to respond quickly since it takes time to compute and reroute traffic optimally, and to update router configurations, it provides a benchmark for comparison with other schemes. Slice only requires that (i) the network re-optimizes when performance may otherwise be violated, and not on each failure event or demand shift; and (ii) the network makes sufficient changes to achieve acceptable performance (rather than all changes to meet the optimal). We discuss how Slice realizes the oracle procedures (§4.3.1) for centralized response:

DoAllCertify(A). We have shown that, in §3.1, the problem of determining whether the worst-case MLU over all f link failure scenarios is less than U^t (the desired performance threshold) when a centralized scheme is used is NP-complete.

Given the intractability, Slice obtains a conservative upper bound U^b on the worst-case MLU across all scenarios in A, and verifies if $U^b < U^t$. This guarantees that *DoAllCertify(A)* only returns True when all scenarios have an MLU lower than U^t, although there may be false negatives which is acceptable (§4.3.1). Slice obtains U^b by designing the optimal protection routing for A using the LP (H) and using the resulting MLU. This is acceptable since for every scenario, a protection routing achieves the same or worse MLU than a centralized mechanism.[1] Finally, if A is a singleton set, Slice runs an MCF which is exact. This ensures the algorithm converges and the final classification is correct (§4.3.1).

An alternate approach to bounding worst-case MLU involves an LP relaxation strategy suggested in [84]. While [84] produces provably tighter bounds than (H), the LP is larger. We use (H) for obtaining bounds since it is computationally faster. Further, our empirical experiments have shown (H) matches the optimal in all prac-

[1]In §4.3.2, a given protection routing (not necessarily designed for A) is certified for A. In contrast, here the worst-case MLU of an optimal protection routing designed for A is used as a conservative estimate of performance with a centralized approach.

tical instances in our context. The choice is not integral to Slice, which is a general framework that allows for different bounding strategies.

DoAllViolate(A). Like in §4.3.2, Slice determines whether all scenarios violate by solving an LP (4.5) to determine if there exists any scenario $x \in A$ for which a feasible solution can be found such that the capacity constraints of all non-failed edges are met, and no failed link carries traffic. Slice checks the contrary to determine if any scenario $x \in A$ can be certified by solving the following LP.

$$\begin{aligned} & \min_{r,x} \quad 1 \\ \text{s.t.} \quad & r_{st} \text{ is a unit flow from } s \text{ to } t. \quad \forall s,t \in V \\ & \sum_{s,t} d_{st} r_{st}(e) \leq c_e(1 - x_e) \quad \forall e \in E \\ & x \in A \quad \forall e \in E \end{aligned} \tag{4.5}$$

The LP checks if there is a feasible multi-commodity flow r from s to t such that the capacity constraints of all non-failed edges are met, and no failed link carries traffic, with the integrality requirements on x relaxed.

Partitioning strategy. If Slice is unable to certify A, it partitions A further. To decide which link to fix to obtain this partition, we compute $\sum_e m_e x_l(e)$, where $x_l(e)$ indicates if l is present in the worst-case failure scenario for edge e, and m_e is a measure that relates to how utilized edge e is in the worst-case scenario. Both $x_l(e)$ and m_e are obtained as dual solutions from the LP reformulation of (H). The intuition behind the heuristic is to fix a link l that is part of the bad scenarios for many highly utilized edges e.

4.3.4 Design with excluded scenarios

Existing worst-case design approaches [18, 28, 84] only consider SSets that include all f or fewer simultaneous failures. In the context of protection routing, consider that the MLU with (H) is greater than 1 for such an SSet, indicating that it is infeasible to protect against all such scenarios. An architect today has no recourse but to restrict

design to scenarios with $f-1$ or fewer simultaneous failures. We discuss how Slice can design for general SSets (e.g., a set with most f failure scenarios handled and only some excluded). The key questions are (i) determining *which* scenarios to design for and exclude; and (ii) how to design a protection routing with a set of scenarios excluded. We discuss each in turn.

Determining which scenarios to design for. To appreciate the complexity of the problem, for a network with E links, there are $\binom{E}{f}$ scenarios involving f link failures, and $2^{\binom{E}{f}}$ possible failure scenario sets to consider (e.g., for a network with 300 links, there are 44850 2-failure scenarios, and 2^{44850} possible sets of 2-failure scenarios). The design of an implementable protection routing involves finding a set among these possible sets (ideally, with cardinality as large as possible) for which (H) can find a congestion-free routing.

Observe that any set for which a protection routing design is feasible must necessarily exclude *all* scenarios that have unacceptable performance with Centralized (since a protection routing approach can perform no better than Centralized, and since including a single bad scenario can make protection routing design infeasible). Unfortunately, by Proposition 3.1.1, discovering even one scenario that has unacceptable performance with Centralized is NP-complete, let alone discovering all such scenarios.

Slice's classification algorithm with Centralized when run to completion determines the set (say L) that excludes *only* those failures that perform unacceptably with Centralized. When design is viable for L (which interestingly has always been the case in our evaluations), it is guaranteed to be the set with maximum cardinality for which protection routing is viable. More generally, Slice classification may be continued with an updated lower threshold (say $0.9U^*$). The insight is that considering scenarios which are comfortably handled by Centralized could identify a failure set that is amenable to a protection routing. The threshold can be iteratively reduced if needed. For larger topologies, the classification algorithm may be terminated earlier when it identifies a sufficiently large subset of L that meets the architect goals.

Protection routing design with excluded scenarios. Even if a heuristic were to identify which scenarios to exclude, it is unclear how to use LP (H) to design for an SSet X that excludes some scenarios with f failures. One approach is to add constraints which indicate that certain failure scenarios cannot be part of the optimal solution. For instance, to eliminate the failure scenario $x = (1, 0, 1, 1)$ (which indicates the failure of the first, third and fourth links), a constraint could be added to X of the form $x_1 + (1 - x_2) + x_3 + x_4 \leq 3$. Unfortunately, the resulting formulation is conservative. because in this representation, all the corner points of X are not failure scenarios. Instead, we represent X as the *union* of multiple SSets A_i (i.e., $X = \cup_m^{i=1} A_i$), with each A_i itself expressible as an intersection of multiple SSets expressible using linear constraints. LP (H) is modified by replicating (4.1) m times, once for each A_i. Recall that Slice's classification produces the set of certifiable scenarios in a union representation. It is feasible to show that when a protection routing is designed for an X represented as a union of SSets produced by Slice, LP (H) is a tighter formulation (i.e., it will find a protection routing with MLU under 1 when one exists for that set of scenarios provided reservations are made a priori), since the corner points in X represent failure scenarios.

Importance of Slice's compact representation. When designing for $X = \cup_m^{i=1} A_i$, since Equation (4.1) of LP (H) is replicated m times for each of these sets, and since the reformulation after dualization (Equation (4.3)) has $|E|$ constraints for each edge e, the resulting formulation has $O(m|E|^2)$ constraints in the modified (H). An attractive aspect of Slice is that it naturally aggregates failure scenarios into sets, each of which is guaranteed to admit a protection routing, while keeping m small. A naive alternative which exhaustively enumerates all scenarios using Centralized to determine which ones certify, would lead to a much larger m with one failure set for each certifiable scenario.

Table 4.2.: Example SSets for various failure models.

Homogeneous link model; f links fail	$\sum_{\langle i,j\rangle \in E} x_{ij}^f = f$ $x_{ij}^f \in \{0,1\} \quad \langle i,j\rangle \in E$
Heterogeneous link model; L link categories; f_k links fail in category L_k	$\sum_{ij} x_{ij}^f \leq f_k \;\; \langle i,j\rangle \in L_k$, $1 \leq k \leq L$ $x_{ij}^f \in \{0,1\} \quad \langle i,j\rangle \in E$
SRLG model f SRLG failures	$\sum_{k \in G} x_k^g = f$ $x_k^g \leq x_{ij}^f \quad k \in G, \langle i,j\rangle \in g_k$ $\sum_{k \in G, \langle i,j\rangle \in g_k} x_k^g \geq x_{ij}^f$ $x_k^g, x_{ij}^f \in \{0,1\}\ g \in G, \langle i,j\rangle \in E$
Partial link failure model f units fail across links	$\sum_{\langle i,j\rangle \in E} x_{ij}^f = f$ $x_{ij}^f \in [0, n_{ij}], x_{ij}^f \in \mathbb{Z}\ \langle i,j\rangle \in E$

4.3.5 Generalizations and extensions

Checking if sufficient scenarios certify. The algorithm in §4.3.1 is easily adapted to certify that a fraction p of scenarios (say all simultaneous f link failures) certify. The algorithm starts with an SSet that represents all such scenarios, and terminates when the count of certifiable scenarios exceeds a fraction p of the total, or violating scenarios exceeds $1 - p$.

Slice summarizes network performance across failures that occur with different probability using a metric that we refer to as *ProbCS* which captures the probability of scenarios for which the steady state performance of the network is acceptable. Slice is general and works with any oracle that provides the probability of each SSet.

Slice probability model. We present a formal argument to show Slice can work with any general oracle that provides the probability of each SSet. Let $\mathcal{F}(x)$ denote a metric of congestion the network faces (e.g., the minimum MLU) under failure scenario x. Now, let χ be a random failure scenario drawn from X, the set

of all failure scenarios, with a given probability distribution. Then, for the random variable $\mathcal{F}(\chi)$, which measures network congestion for randomly drawn scenario χ, we are interested in the probability of the level-set, $\mathcal{F}(\chi) \leq U^*$, where U^* is a given threshold congestion, typically chosen as 1 or below when MLU is the metric chosen to capture congestion. We call the set of failure scenarios in which congestion does not exceed the given threshold as the set of certifiable scenarios and denote the occurrence of such a scenario as the event C. The complement of C, the set of violating scenarios, is denoted as B.

Let the current SSet A be partitioned into $\{A_1 \mid \ldots \mid A_n\}$ so that $\sum_{i=1}^{n} P(A_i) = P(A)$. We are interested in estimating $P(C \cap A)$ and $P(B \cap A)$. Since $A_1, \ldots, A_n$ form a partition of A, for any event Y, and in particular for C and B, we have $P(Y \cap A) = P(Y \mid A)P(A) = \sum_{i=1}^{n} P(Y \mid A_i)P(A_i)$. To bound $P(Y \cap A)$, we approximate the right-hand-side of the above equivalence. To do so, assume that we have available two index sets $I, J \subseteq \{1, \ldots, n\}$ so that, for all $i \in I$, $P(Y \mid A_i) = 1$, and, for $i \in J$, $P(Y \mid A_i) = 0$. Then, $\sum_{i \notin J} P(A_i) \geq P(Y \cap A) = \sum_{i \in I} P(A_i) + \sum_{i \notin I \cup J} P(Y \mid A_i)P(A_i) \geq \sum_{i \in I} P(A_i)$. With a sufficiently fine partition, $\overline{I \cup J} = \emptyset$, since individual failure scenarios are either certifiable or violating. For such a partition, the upper bound matches $P(Y \cap A)$ exactly.

Since the above treatment applies when $Y = C$ or $Y = B$, it yields lower as well as upper bounds on the probability of certifiable and violating scenarios. Applying the bounding argument with $Y = C$, an element $i \in I$ if $P(C \mid A_i) = 1$, which occurs if and only if the performance of all scenarios in A_i is acceptable (i.e., $\mathcal{F}(x) \leq U^*, \ \forall x \in A_i$). Likewise, $i \in J$ if $P(C \mid A_i) = 0$, which occurs if and only if no scenario has acceptable performance (i.e., $\nexists x \in A_i, \mathcal{F}(x) \leq U^*$). Note that the above treatment is general, and works for any general oracle that can provide the necessary $P(A_i)$.

Slice can handle correlated failures if the joint failure probability is provided, along with the probability of each failure occurring by itself. Pragmatically, an architect typically models failures as independent events, where an event may correspond to the failure of an SRLG, an IP link, or a sub-link. If failure probabilities are homogeneous,

the probability of each FSSet is computed using a binomial distribution. When probabilities vary, Slice categorizes failure events into classes, where probabilities of events in the same class are the same, and appropriately adapts the SSet probability computations.

In performing such certification, Slice may terminate the classification algorithm at an intermediate state, and use a sampling algorithm to test unclassified scenarios. Consider the null hypothesis (H_0) that the network does not perform acceptably for a fraction p of scenarios. The alternative hypothesis (H_a) is that the network performs acceptably for a fraction p_a of scenarios ($p_a > p_0$). Slice tests n samples and uses a one-sided one-proportion test [86] to determine whether the null hypothesis can be rejected. n is chosen such that (i) the probability of Type I error (i.e., rejecting H_0 when it is true) is less than α; and (ii) the probability of Type II error (i.e., failing to reject H_0 when H_a is true) is less than β, where α and β are parameters. The sample size depends on p_0, p_a, α and β (detailed later). If Slice is able to identify a fraction z of scenarios as having acceptable performance, then, p_0 and p_a in the sample size calculations are respectively replaced by $\frac{p_0 - z}{1-z}$, and $\frac{p_a - z}{1-z}$, which can significant reduce the number of samples required.

Sample size determination. In the sampling algorithm used by Slice, computing the exact sample size requires an iterative procedure using a binomial distribution. However, under typical conditions, the distribution of possible values of $\hat{p}$ (the fraction of cases the network performs acceptably) is approximately a normal curve (by central limit theorem), with mean the population proportion (p) of interest, and standard deviation $\sqrt{\frac{p(1-p)}{n}}$, where n is the number of tested samples. The sample size is estimated using this approximation. Specifically, we find $\hat{p}$ and sample size n such that $\frac{(\hat{p}-p_0)\sqrt{n}}{\sqrt{p_0(1-p_0)}} = z_\alpha$ and $\frac{(\hat{p}-p_a)\sqrt{n}}{\sqrt{p_a(1-p_a)}} = -z_\beta$, where z_α and z_β are the critical z-scores to ensure that probability of Type I error $< \alpha$ and that of Type II error $\leq \beta$, and p_0 and p_a are as defined in §4.3.5.

Note that a normal approximation can slightly underestimate the actual sample size obtained by exact calculation, but since we do not deal with large deviations,

this difference turns out to be usually small. If this approximation error is to be eliminated, exact calculation is preferable because the upper bound obtained via Chernoff's Theorem tends to be conservative.

Handling diverse failure models. Slice has been integrated with diverse failure models including (i) *Homogeneous link model*, where all links have the same downtime probabilities; (ii) *Heterogeneous link model*, where links belong to L classes, with links in the same class having the same downtime probabilities; (ii) *SRLG model*, where a network comprises G groups, with links belonging to one or more groups. The failure of a group implies all links belonging to that group fail, and if a link fails, at least one group to which the link belongs fails; and (iii) *Partial link failure model*, where units of a link can fail independent of one another.

A key difference across the models is how the initial SSets are generated. Table 4.2 shows initial SSets and associated constraints for each model. Note the partial link failure model uses integer variables x_{ij}^f instead of binary variables, while the SRLG model uses new binary variables x_k^g indicating if the SRLG corresponding to group k has failed. The algorithms also involve other minor changes - e.g., in the partial model, if SSet A_i must be further partitioned, we fix a link $\langle i, j \rangle$ as before, but create $n_{ij} + 1$ partitions, each corresponding to a different number of units of link $\langle i, j \rangle$ that fail.

Multiple traffic demands. While we have focused on failure uncertainty, we next discuss how Slice tackles the case, where demand is also uncertain and can take one of h values, say d_i for $i \in \{1, \ldots, h\}$. Here, each d_i is a traffic matrix consisting of traffic between all node-pairs. The uncertainty set X now consists of scenarios specified jointly by the current network failure state and traffic. For continuous demands (such as the convex hull of a set of discrete demands), we partition possible demands into regions, and, associate each region with a demand that dominates all the demands in that region. For instance, if, within a region, the demand for a node-pair (s, t) varies between d_{st}^l and d_{st}^h, with $d_{st}^l < d_{st}^h$, then, in the dominating demand, we assign the (s, t) demand at its highest possible value d_{st}^h. When the resulting

Table 4.3.: Topologies used in evaluations.

Network	Number of Nodes	Number of Edges
Abilene	11	14
GEANT	32	50
Deltacom	103	151
ION	114	135

discrete approximation is used, the performance profiles generated with Slice are a conservative estimate of true network capability. This estimate can be improved by approximating the demand distribution more closely using a finer partition of the demand set.

4.4 Evaluations

Our evaluations shows the effectiveness of Slice in (i) classifying failure scenarios (§4.4.1); (ii) analyzing metrics beyond worst-case performance for protection routing and centralized schemes (§4.4.2); (iii) enabling better designs by judiciously excluding failure scenarios (§4.4.3); (iv) generalizing to multiple traffic classes (§4.4.4); and (v) ensuring acceptable computation time (§4.4.5).

Topologies evaluated. Table 4.3 summarizes the topologies used in our evaluations. We select one moderate-size network, GEANT, and two large networks, Deltacom and ION, from the Internet Topology Zoo [85]. One-degree nodes in the topologies are recursively removed so that the networks are not disconnected with a single link failure. We use the gravity model [37] to generate traffic matrices with MLU in the range [0.6, 0.65] across the topologies. GEANT has links ranging from 1 to 100 Gbps. Link capacities for Deltacom and ION are unavailable, and we use uniform capacities.

Failure models. We model each link with its capacity split evenly across k sub-links that fail independently. $k = 1$ denotes full link failures, and $k > 1$ denotes partial link failures (§4.3.5). We summarize network performance across failure classes using the *ProbCS* metric (§4.3.5) which is sensitive to the probability of different failure scenarios. Prior network failure studies [8–10, 12] indicate that the vast majority of links have 3 9s or 4 9s of uptime. We primarily conduct our evaluations with a conservative model which we refer to as *HomogFM3*, where all sub-links have 3 9s of reliability, and the failure of sub-links is treated as independent events. We have also considered multiple models based on [9, 10] that capture heterogeneity in link failure probabilities, and differ in the fraction of links with 3 9s and 4 9s of reliability.

Slice implementation. We implement Slice in Python, and use Gurobi 8.0 [87] to solve LPs. We implement the classification algorithm described in §4.3.1, and oracle procedures described in §4.3.2 and §4.3.3. Starting with the base case where the design is tested against fixed numbers of failures, we dynamically derive and add constraints to fix failure states of links. When analyzing large networks such as Deltacom and ION with Centralized, we use the hybrid approach (§4.3.5) that combines classification with Slice and sampling. By default, we switch to sampling after the classification step certifies that the network achieves at least a ProbCS of 2 9s (0.99). The unclassified SSets are then sampled in the same proportion as their probabilities, with both Type I error (α) and Type II error (β) under 1%. We use $p_a = 0.9995$ for a ProbCS of 3 9s, and $p_a = 0.99995$ for 4 9s.

4.4.1 Illustrating classification with Slice

Figure 4.1 illustrates the results obtained by running Slice's classification algorithm (§4.3.1) with Centralized for GEANT, $k = 2$. Each tree node represents an SSet, with the root node representing all scenarios involving the simultaneous failure of 2 sub-links. Each node is annotated with the MLU of the worst-case scenario (a value under 1 indicates all scenarios corresponding to that node are certifiable), and

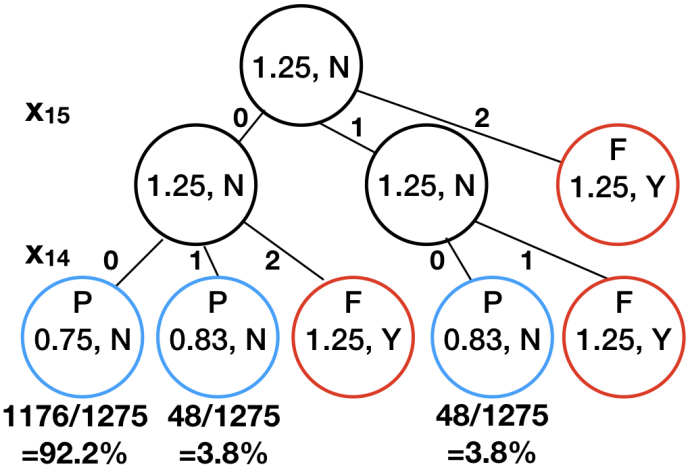


Fig. 4.1.: Slice's failure classification.

the return value (Yes or No) of $DoAllViolate()$. For example, Slice is neither able to certify all scenarios corresponding to the root node (MLU = 1.25), nor determine they all violate ('N'). Slice partitions the corresponding SSet into complementary and disjoint subsets that differ based on the status of link 15 (the link is chosen as described in §4.3.3). Notice that there are 3 possible states, corresponding to the number of sub-links of link 15 that fail (0, 1, or 2). The algorithm terminates by classifying all scenarios (1,275 in total) into six SSets (leaves of the tree). Blue nodes marked with "P" represent the certified SSets. Red nodes marked with "F" represent the violating SSets. Each certified node is also annotated with the number and percentage of certified scenarios. For instance, the leftmost leaf node corresponds to a single certifiable SSet that captures 92.2% of the scenarios. This ability to compact certifiable scenarios into a small number of SSets aids design with Slice as we discuss later.

4.4.2 Insights from Slice's failure analysis

Does design for f failures suffice? The existing approach to analyzing and designing resilient networks is to consider worst-case performance over all scenarios

involving up to f simultaneous failures [18, 28, 84], for a specific routing mechanism. In the context of protection routing, when LP (H) is used for GEANT, the MLU is under 1 for $f = 1$, indicating the resulting routing (which we term PR-Worst(1)) meets the promised bandwidth for single link failures. However, the MLU exceeds 1 for $f = 2$. While this indicates that it is infeasible to handle all two failure scenarios, the generated routing (which we term PR-Worst(2)) maximizes the fraction of traffic that can be delivered across the scenarios.

Figure 4.2 presents insights gleaned from Slice's general analysis framework that considers performance across scenarios, and provides comparisons with the ideal approach, Centralized. The figure shows the percentage of 1-, 2-, and 3-link failure scenarios for which the promised bandwidth requirements is met (MLU < 1) for the three schemes as determined by Slice. All schemes can handle all single failures. While Centralized can handle practically all 2- and 3-link failure scenarios, both PR-Worst schemes can only handle a much smaller subset of these scenarios. PR-Worst(2) performs poorly because it optimizes for the worst-case under 2-link failures, and cannot control the performance of the remaining scenarios. PR-Worst(1) performs better, but still incurs a large gap relative to Centralized because it does not explicitly design for any 2-link scenario. The results highlight (i) the importance of Slice's generalized analysis, and (ii) that when design is restricted to worst-case over f failures, there is no choice of f leading to good performance.

Analysis across topologies. Since failures may occur with different frequency (e.g., single link failures occur more frequently than two link failures), we summarize results with Slice across multiple topologies using the ProbCS metric and the *HomogFM3* failure model (§2.6).

We set ProbCS targets of 90%, 99%, 99.9%, and 99.99% (1 9 to 4 9s), and use Slice to check if the targets can be met. For PR-Worst, we consider the largest f for which LP (H) is able to find a protection routing, but always use an f of at least 1. Figure 4.3 shows the maximum number of 9s that can be certified with PR-Worst and Centralized for each topology and different k. The figure reveals a big gap in the

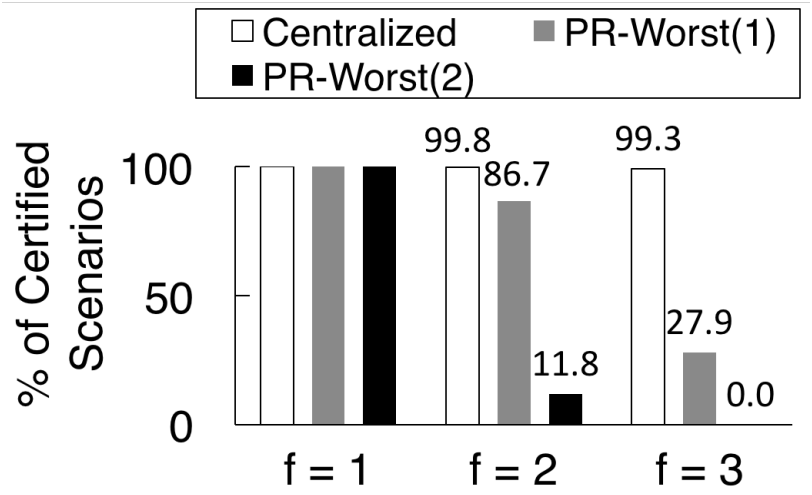


Fig. 4.2.: Worst-case design effectiveness.

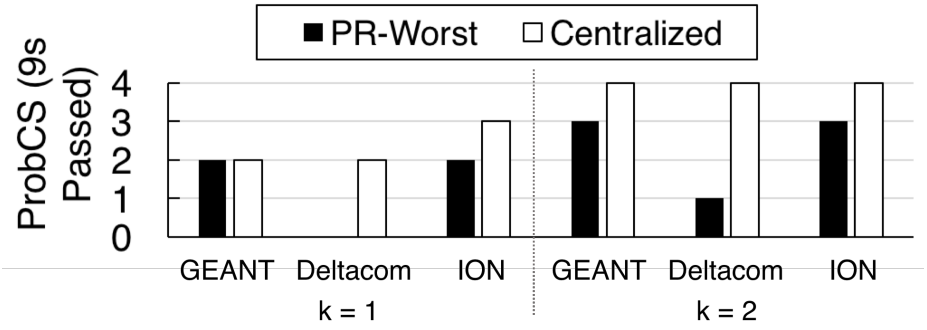


Fig. 4.3.: Probability of scenarios for which network performance is acceptable under various settings.

ProbCS achieved by PR-Worst and Centralized for all topologies and settings except GEANT $k = 1$. For Deltacom $k = 1$, PR-Worst is unable to protect against any single failure and cannot achieve even a single 9, while Centralized certifies 2 9s. For $k = 2$, both PR-Worst and Centralized achieve higher 9s than $k = 1$ indicating that the benefits of reducing the impact of failures with larger k outweighs the concern that the network is more likely to be in a failure state. However, the performance gap between PR-Worst and Centralized persists.

We have also considered models that capture heterogeneity in link failure probabilities, and differ in the fraction of links with 3 9s and 4 9s of reliability. Our results

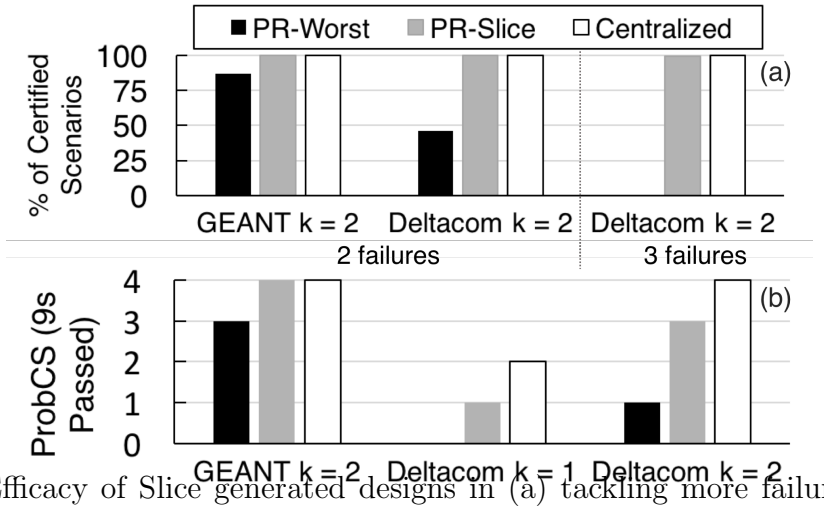


Fig. 4.4.: Efficacy of Slice generated designs in (a) tackling more failure scenarios; and (b) improving ProbCS.

indicate that a gap persists between PR-Worst and Centralized across all models, though both schemes achieve a higher ProbCS as more links are upgraded to 4 9s.

4.4.3 Design with excluded scenarios

We now evaluate the potential for bridging the large gap between PR-Worst and Centralized shown in §4.4.2 with Slice. The approach (named PR-Slice) is to design for a selected set of scenarios rather than for all f simultaneous failures.

Recall that PR-Slice classifies failure scenarios that certify with Centralized, and designs a protection routing considering only these scenarios (§4.3.4). To illustrate for GEANT $k = 2$, consider Figure 4.1 where Slice has classified 2-failure scenarios into three SSets (comprising 1272 scenarios) that certify, and other failed scenarios to be excluded. A new protection routing is designed for a union of these three SSets (besides SSets corresponding to no failure and all single failures) using our modifications to LP (H) (§4.3.4). Figure 4.4a shows the benefits of PR-Slice for GEANT $k = 2$. While PR-Worst can only certify 86.7% of 2-failure scenarios, PR-Slice is explicitly designed for all two or fewer failure scenarios except for 3 violating ones, and can handle 99.8% of 2-failure scenarios.

Slice vs. naive alternatives. Recall that LP (H) augmented for the union of m SSets has $O(m|E|^2)$ constraints (§4.3.4). Owing to Slice's compact representation of scenarios that certify, $m = 3$ suffices for 2-failure scenarios. In comparison, a naive approach of exhaustively enumerating and checking all scenarios with Centralized is not only time-consuming but would have $m = 1272$, resulting in a much harder to solve design LP with orders of magnitude more constraints. Another naive alternative is to randomly exclude, say $q\%$ of the scenarios. First, it is unclear how to compactly represent such a set to ensure a tractable design LP. Second, it is not clear what q to choose. Third, even if q were heuristically chosen, the probability of picking an acceptable set that excludes all three scenarios that perform poorly with Centralized is low. For example, if $q = 1\%$, there are 3.5×10^{30} possible sets of candidate 2-failure scenarios for GEANT, and only a fraction of 8.29×10^{-7} of these sets is acceptable. In contrast, Slice not only finds such a set, but also the one with maximum cardinality that only excludes the violating scenarios (§4.3.4).

Larger design study. We consider Deltacom $k = 2$ for which PR-Worst achieves a ProbCS of only a single 9, far short of the 4 9s achieved by Centralized (Figure 4.3). We ran Slice classification on 2-failure scenarios to identify promising scenarios for the design. Slice identified 6 SSets that compactly captured 11,465 2-failure scenarios, and determined 11 violating 2-failure scenarios. We found that design for these scenarios results in a protection routing that achieves a 2 9s target (an already significant improvement over PR-Worst). We explored the feasibility of doing even better by running the classification algorithm on 3-failure scenarios terminating when sufficient scenarios to meet a 3 9s target are identified. Slice identified 5 SSets representing 466,899 3-failure scenarios. We next design a protection routing for the union of the 5 promising 3-failure sets, 6 promising 2-failure sets, and 2 SSets each representing single and no failure scenarios. Figure 4.4a shows that the generated PR-Slice certifies 99.9% of 2-failure scenarios (matching Centralized) and 99.6% of 3-failure scenarios. In contrast, PR-Worst certifies only 46.1% of 2-failure cases, and no 3-failure case.

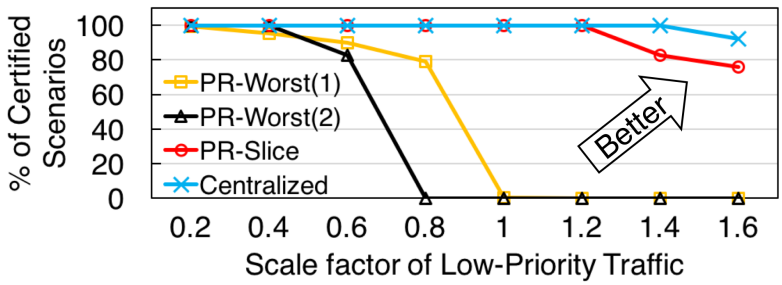


Fig. 4.5.: Percentage of certified 2-failure scenarios, obtained with different schemes for a range of scale factors of lower priority traffic in GEANT $k = 2$.

Impact on ProbCS metric. Figure 4.4b shows that PR-Slice achieves significantly higher ProbCS than PR-Worst in all cases. The gap with Centralized indicates even further room for improvement, which can potentially be bridged by running Slice's classification of scenarios with Centralized even further. For instance, for the Deltacom k=2 example described above, running Centralized until more 3-failure and possibly some 4-failure scenarios are classified may help to identify more promising scenarios that can be considered when designing a protection routing. Overall, the results show Slice's ability to improve design quality by judiciously excluding violating scenarios rather than considering the worst-case.

4.4.4 Design with multiple traffic classes

So far, we have focused on a single traffic class where the goal is to meet the entire bandwidth requirement. We show Slice's generality by considering a context with multiple traffic classes where the architect wishes to meet all high priority, and as much of the low priority traffic as possible.

Designing a protection routing for multiple traffic classes involves minor changes to LP (H) and is presented below. Let d^h and d^l represent high and low priority traffic matrices, with d^h_{st} and d^l_{st} representing the relevant traffic from s to t. The formulation determines the largest Z such that the network can handle a traffic matrix D where D_{st} is $d^h_{st} + Zd^l_{st}$ (i.e., the network can carry all high priority traffic, and a fraction Z

of low priority traffic). $Z \geq 1$ indicates all low priority traffic can be carried. Setting d^h to zero produces the special case where there is a single class of traffic, when Z would share an inverse relationship with the MLU metric. The protection routing has parameters (r^h, r^l, p, a), where r^h and r^l represent flows corresponding to high and low priority traffic from s to t. The formulation is similar to (H) except that r^l_{st} only need carry a fraction Z of the d^l_{st} traffic, however link capacity constraints must be strictly met.

$$
\begin{aligned}
\text{(G)} \quad & \max_{r^h, r^l, p, a, Z} \quad Z \\
\text{s.t.} \quad & r^h_{st} \text{ is a flow of } d^h_{st} \text{ from } s \text{ to } t. \quad \forall s, t \in V \\
& r^l_{st} \text{ is a flow of } Z d^l_{st} \text{ from } s \text{ to } t. \quad \forall s, t \in V \\
& p_l \text{ is a flow of } a_l \text{ from } i \text{ to } j. \quad \forall l \in E, l = \langle i, j \rangle \\
& \sum_{s,t} r^h_{st}(e) + \sum_{s,t} r^l_{st}(e) + \sum_{l \in E} x_l p_l(e) \\
& \qquad \leq c_e(1 - x_e) + a_e x_e \quad \forall x \in X, e \in E \\
& a_e \geq 0 \quad \forall e \in E \\
& Z \geq 0
\end{aligned}
$$

Certifying a given protection routing has a minor change from §4.3.2. Consider that the routing $(r^{h*}, r^{l*}, p^*, a^*)$ achieves Z^* with (G) when designed for a given set of failures X. To verify the fraction of scenarios for which a target Z^t can be achieved, we adopt a similar procedure to §4.3.2 except that instead of (4.4), we verify the slightly modified capacity constraint below:

$$
\max_{x \in A} \sum_{l \in E} x_l q^*_l(e) \leq c_e - \sum_{s,t} (r^{h*}_{st}(e) + r^{l*}_{st}(e) \frac{Z^t}{Z^*}) \quad \forall e \tag{4.6}
$$

where $q^*_e(e) = c_e - a^*_e$, and $q^*_l(e) = p^*_l(e)$, $l \neq e$. This simply indicates that the capacity constraint must be met, but with r^{l*} rescaled for Z^t rather than the originally designed Z^*.

The two-class LP determines a protection routing that handles all high-priority traffic and the low-priority traffic scaled by a factor Z, while maximizing Z. We

refer to Z as the scale factor, with $Z \geq 1$ indicating the entire low priority traffic is handled. We refer to PR-Worst(f) as a protection routing derived from this two-class LP when protecting against all simultaneous failures involving f or fewer links. As before PR-Slice is obtained by (i) using Slice to classify scenarios that obtain a $Z \geq 1$ with Centralized; and (ii) using the two-class LP to design with the set so obtained.

We modified Slice's classification algorithm to generate *performance profiles*, i.e., determine the fraction of scenarios that achieve different Z thresholds for the schemes above. Given multiple scale factors, $Z^1 < Z^2 < \cdots < Z^m$, the algorithm classifies scenarios to determine which of them meet the most relaxed Z^1 threshold. Violating SSets are not further inspected, but certifying SSets are explored further to verify if the more strict Z^2 threshold is met. The process continues until eventually the Z^m threshold is considered.

Figure 4.5 shows the results obtained by analyzing various schemes with Slice for GEANT $k = 2$ for 2-failure scenarios. We split the original traffic matrix into two classes with high and low priority. For each cell we assign a random fraction of it to the high-priority traffic class, and the rest to the low-priority one. Each curve corresponds to one scheme, and shows the fraction of 2-failure scenarios that can attain a particular Z. The top-most curve shows the ideal Centralized scheme, which can attain Z of 1 for over 99% of the scenarios. PR-Slice achieves a performance profile nearly as good as Centralized. While it degrades moderately for the most stringent performance thresholds ($Z = 1.4$ and 1.6), we note that an architect could use Slice to generate new protection routings optimized for these thresholds if this is desirable. The two PR-Worst schemes perform poorly, with no scenario achieving a Z of 1 where all low priority traffic could be carried. Note that PR-Worst(2) matches Centralized and performs slightly better than PR-Slice for the worst-case (achieving a Z of 0.42) but this comes at the expense of performance for the vast majority of scenarios.

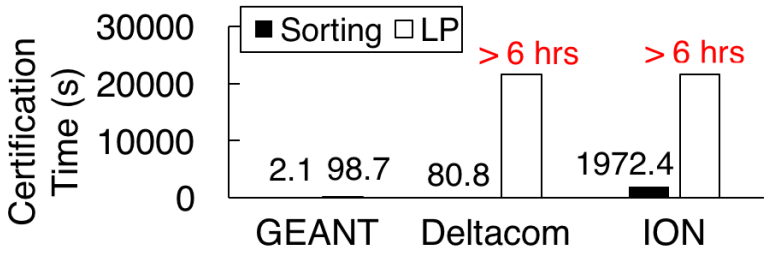


Fig. 4.6.: Certification time of Slice with PR-Worst.

4.4.5 Certification time with Slice

In this section, we discuss the running time with Slice to certify both the protection routing and centralized schemes. We report the total CPU time on a 3.00GHz Intel Xeon E5-2623 machine using a single-threaded implementation.

Protection routing. Figure 4.6 reports certification time for both Slice's sorting approach and an LP approach (§4.3.2) that we refer to as Sorting and LP. We focus on $k = 2$ settings since they involve more scenarios and take longer than $k = 1$. For each topology and scheme, we measure the certification time for multiple ProbCS targets ranging from 1 9 to 4 9s, and present the largest time (e.g., it takes longer to certify 4 9s than 1 9, but detecting 4 9s violates may be faster than certifying 1 9). We make the following observations. First, Sorting outperforms LP in all settings. The benefits are significant for larger networks, where we needed to terminate the experiments with LP after 6 hours. For Deltacom, Sorting is two orders of magnitude faster. For ION, while Sorting certifies 4 9s within 33 minutes, 3 9s is less than a minute. LP could not even certify 3 9s after 6 hours. ION takes much longer to certify than Deltacom despite similar network sizes. This is because ION certifies 4 9s, and a larger number of scenarios need to be considered, whereas Deltacom certifies 1 9 and violates 2 9s, requiring fewer scenarios to be checked.

Centralized schemes. Certifying Centralized takes longer because even the task of certifying a single SSet is NP-complete (Proposition 3.1.1). Despite this, Slice only takes 87.2 seconds for the moderately sized GEANT network running the classification

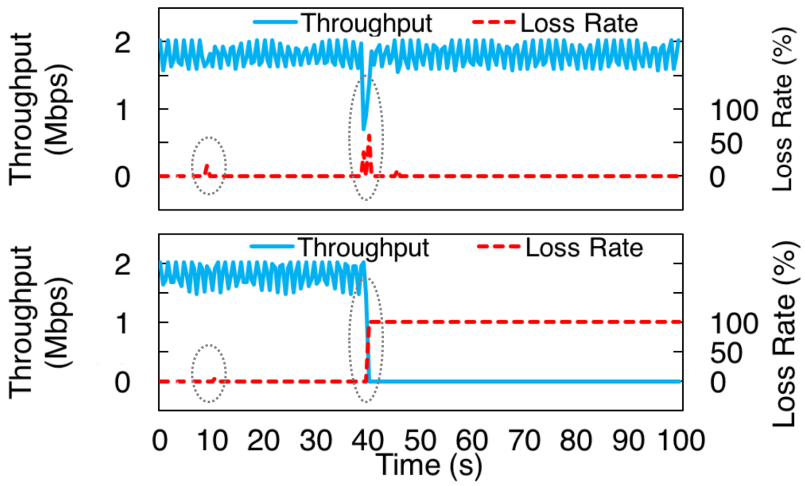


Fig. 4.7.: Throughput and loss rate across UDP flows with PR-Slice (top) PR-Worst (bottom) on testbed.

algorithm to completion. For larger topologies, recall that we run the classification algorithm until 2 9s are certified, and then switch to sampling (§2.6). Using this hybrid approach, Slice performs acceptably, taking 2.1 hours and 0.9 hours to certify 4 9s for Deltacom and ION, respectively. We have also run experiments comparing the hybrid approach to sampling alone. While the hybrid approach classifies all two sub-link failure scenarios, sampling only classifies about 83% for both topologies. Further, unlike sampling, the hybrid approach can group scenarios into compact SSets, necessary for tractable protection routing design. The hybrid approach achieves a speed-up of 2.8x for ION, and 1.3x for Deltacom. Finally, we note that Slice is relatively unoptimized. There are multiple ways to reduce certification time with Slice in the future, such as running the bounding LP (H) only at some nodes, and reusing the parameters computed at an ancestor node at some descendant nodes.

4.5 Validations on SDN testbed

In this section, we validate the effectiveness of PR-Slice in protecting against a wider range of failure states than PR-Worst using SDN testbed experiments. The

experiments also show that our generalizations to existing protection routing models (§4.2.1) is practically realizable.

Emulation setup and protection routing implementation. We conduct emulations using Mininet 2.2 and OpenVSwitch 2.10. Both normal (r) and protection routing (p) variables require each router to forward traffic to a destination with multiple next hops. We achieve this leveraging the select group table feature in OpenFlow 1.5 using multiple buckets with different weights in a group table entry. OpenVSwitch's internal hashing method guarantees that packets belong to a single flow map to the same forwarding path.

Initial flow rules for normal and protection routing are installed by a central controller. When a link $\langle i, j\rangle$ fails, the head (i) and tail (j) switches are responsible for pushing and popping a unique MPLS label (say l_{ij}) encoding the failure. All the switches forward the labeled packets based on flow rules pre-installed by the controller that translate the appropriate protection routing variables p. Packets may carry stacked labels if they encounter multiple failed links along the path. Failure information of $\langle i, j\rangle$ is also propagated to the controller which adjusts the protection routing parameters p and a to p' and a' to ensure that the protection routing for any subsequent failure $\langle u, v\rangle$ does not use either $\langle i, j\rangle$ or $\langle u, v\rangle$. Since this update is only to protect against future failures, it is acceptable to involve the controller in this operation. We omit the details regarding the adjustments, but we remark that the new variables are efficiently computable.

Experimental results. We compare the performance of PR-Slice and PR-Worst by emulating Abilene $k = 1$ on the testbed for a matrix where the traffic is dominated by a source destination pair. It is infeasible to design a protection routing with PR-Worst to protect against all 2-failure scenarios, since the graph may get disconnected on a few of these failure scenarios (and LP (H) is infeasible to solve). Hence, we design PR-Worst for all single failures. For PR-Slice, we design the protection routing for the union of the SSets corresponding to 0-failure, 1-failure and all 2-failure SSets which can achieve an MLU under 1 with Centralized (20 SSets representing 93% of 2-failure

scenarios in all) as described in §4.3.4. The MLU with LP (H) for the union of sets is 0.9, indicating all these scenarios can be tackled but the network may operate close to saturation on some of them.

We create 30 UDP flows between the source and the destination, and split the demand uniformly across the flows. Note that any flow is hashed to one path constantly, but multiple flows may be hashed to different paths following the routing parameters. The rate is set to the maximum possible given the system resources of the host machine, and link capacities are proportionally scaled down. Figure 4.7 presents the throughput and packet loss rate measured at the destination on the failure of two links e_1 and e_2 (highlighted by the circles), occurring 30 seconds apart. Figure 4.7 (top) shows the result for PR-Slice. Notice that each failure event leads to a transient impact on throughput and loss rate, though performance quickly recovers after protection routing is activated.

Figure 4.7 (bottom) shows the result for PR-Worst. While it can handle the first failure, there is complete packet loss after the second failure, because PR-Worst is restricted to design for single failure scenarios. In this example, PR-Worst resulted in e_1 and e_2 mutually using each other to protect against their respective failures, which is sufficient to guard against a single but not two failures. PR-Slice prevents this by using more diverse paths to protect e_1 and e_2.

Like [28], it is possible for PR-Slice to suffer transient loops when links fail near simultaneously. For instance, in the example above, each of e_1 and e_2 may partially use each other to protect against their respective failures. However, the final protection variables are correctly restored based on the sequence of failures observed by the controller using the adjustments described above. In contrast, PR-Worst is unable to eventually recover. In practice, other mechanisms may be needed for recovery such as a re-execution of LP (H), which may take several minutes, and necessitate significant traffic re-routing beyond diverting traffic on e_1 and e_2.

4.6 Related work

The closest related works to Slice are R3 [28], FFC [18], and robust validation [84]. FFC [18] is a path-based protection mechanism. When a particular path fails, traffic is redistributed by the source of the failed path to surviving paths. FFC assigns bandwidth to flows so the assignment can be handled under any scenario involving up to f link and f_n node failures. Robust validation [84] determines the worst-case performance for a given set of demands or failures if the network responded optimally for each scenario (i.e., if the Centralized scheme is used). Slice is a general framework that applies to all of these schemes by supporting failure classification, certification, performance profiles, and design for more general objectives than the worst case. While we have already discussed how Slice applies to R3 and Centralized, Slice also applies to FFC. For instance, considering all f failures with FFC may be conservative, and Slice can aid FFC to exclude failure scenarios, which may lead to a design with higher bandwidth assignment. The ideas in §4.3.4 can also be extended to allow FFC to design for arbitrary SSets.

Much earlier work on resilient network design has either (i) only focused on availability [13–17] resulting in poor performance on failures [12]; or (ii) only considered robust design for a small number of failure states (e.g., single-link or node failures) [19–25]. Our research considers performance (not just availability), and scales to consider the combinatorially many failure states arising from multiple concurrent failures.

Determining optimal ways to route traffic in a demand-invariant manner while minimizing MLU has been well studied [23,39,42,71]. These works could be viewed as providing guarantees on worst-case network performance, assuming adaptation is not permissible. Semi-oblivious traffic engineering [88] picks paths in a demand-invariant manner, but allows flexibility in how traffic is routed across tunnels.

Slice can complement and aid topology synthesis tools [29]. Much recent progress has been made on ensuring the correctness of network configurations, and the control

and data plane [31, 32, 89–91]. While we share similar inspiration, our focus is on performance properties, an area that has only started receiving attention (e.g., [92]). Slice simultaneously handles discovery and exclusion of scenarios unlike prior work in the optimization community that focuses solely on excluding a pre-specified list of scenarios [93].

4.7 Conclusions

We have argued that when analyzing and designing networks for failures, considering worst-case performance across scenarios is insufficient. We have presented Slice, a formal framework that supports failure classification, and generalized analysis and design, which can work with both centralized schemes and protection routing. For Deltacom $k = 2$, Slice's analysis reveals that PR-Worst only handles 46.1% of 2-failure scenarios and no 3-failure scenario. In contrast, PR-Slice (synthesized using Slice) can support nearly all 2- and 3-failure scenarios (closely matching Centralized). For the two traffic class model, Slice's analysis shows that PR-Worst carries less than 80% of low priority traffic in all 2-failure scenarios. In contrast, PR-Slice can handle all low priority traffic (besides high priority traffic) in all but three 2-failure scenarios, while modestly impacting performance for one case. Slice has reasonable computation time owing to its techniques for efficient failure classification. Emulation experiments show the benefits of PR-Slice are realizable in practice.

5. CONCLUSIONS AND FUTURE DIRECTIONS

5.1 Conclusions

Robust certification is the first framework to certify worst-case performance of network across wide range of scenarios with flexible network response. Leveraging cutting-edge techniques from non-linear optimization community, together with network-inspired mechanisms, we have shown that the robust certification framework can achieve performance bounds dominating state-of-the-art [28], and these bounds surprisingly match the optimal for many practical networks we have experimented.

We have generalized the robust certification framework to apply to many more practical situations, such as multiple traffic classes with different priorities, richer failure patterns, routing restriction, and various performance metrics. To study the bound quality of these generalizations, as well as to automate the cumbersome manual process of relaxation and linearization, we have developed a toolkit which makes the robust certification framework more accessible to users. The bounds for different variants all match the optimal for the networks in our context, except for SRLG failure model. In a proof-of-concept implementation, the automation takes less than 40% extra running time for GEANT network (50 edges), compared to the running time of the same model with linearization and relaxation conducted manually.

Further, we have presented Slice, a general framework for certifying network performance objective beyond the worst-case. Slice can efficiently classify failure scenarios based on their performance, and can analyze the performance of given routing designs. Analysis enabled by Slice reveals large performance gaps between centralized network response and protection routing designed for the worst-case. Exploiting the analysis result from Slice's classification algorithm, we can provide a better protection routing design for a percentage of failure scenarios, which achieves close to

optimal performance, for both moderate-size and large networks. Slice has reasonable certification time: on a single-threaded 3.00GHz CPU, it takes 2.1 seconds for Slice to analyze a protection routing design for GEANT, and 80.8 seconds for Deltacom (150 edges). While Slice could take longer for Centralized, many opportunities for optimization exist in the future as discussed below.

Overall, we have shown the potential of network performance certification with formal and quantitative approaches.

5.2 Future directions

This thesis takes the first step in certifying network performance goals under failures with a principled approach, and there are many potential directions down the road. We will elaborate on three facets: scalability of Slice, network synthesis, and latency certification.

Scalability of Slice. As we have elaborated in §4, Slice's classification algorithm plays a vital role in performance analysis and protection routing design, and there is much room for further optimization when Slice is used with Centralized. Also, since the performance of classification algorithm is no worse than the enumeration approach, we can trade the bound quality of *DoAllCertify()*, a key procedure in the classification algorithm dominating the running time, for the computation efficiency. For example, assuming we use the RLT approach described in §2 to realize *DoAllCertify()*, instead of applying a full-blown cross product of all constraint pairs, we can take a subset of all the constraint products in order to improve the solving efficiency of the relaxed model, with the price of possible compromise on bound quality. As another example, we can use a specific protection routing to serve as *DoAllCertify()* of the centralized scheme, exploiting the fact that a specific protection routing always gives the same or more conservative bounds than the *DoAllCertify()* implemented in §4.3.3.

Network synthesis. In both robust certification and Slice, we have shown two concrete examples where our frameworks apply to network synthesis: (i) designing a plan of link capacity augmentation with least cost so that the new design certifies a given worst-case performance target; (ii) designing a protection routing for a percentage of failure scenarios so that the new routing handles more failure scenarios than a design merely optimizing the worst-case performance. The next intriguing question is that whether we can build a framework to generalize a series of design problems. One possible solution is to add a third design stage to the robust certification problem, and use approaches similar to robust certification to reformulate, relax, and solve for performance bounds. Synthesis is clearly a more difficult problems to solve than certification, and there are challenges which need further investigation: (i) since the inner stages need to remain simple, we have to use the relaxed formulation for the second stage. Will this degrade the bound quality of the final relaxation? (ii) Does first-level RLT still suffice?

Latency certification. Latency is a major concern in many business-critical applications today. It would be of great value and interest if we had a formal approach to quantitatively certifying whether a network routing design meets the latency requirement under failures. One question here is what is a reasonable metric for latency, i.e., how to accurately represent the latency in a network? It may be as simple as the aggregation of number of hops from source to destination, or we have to consider the impact of queuing on each links by modeling deterministically or statistically the queuing behaviors, assuming the characteristics of queues are known for each link. Our approach may also need to be informed by empirical validations to make sure the latency validation results reflect real-world considerations.

REFERENCES

REFERENCES

[1] "Cisco Visual Networking Index: Forecast and Trends, 20172022 White Paper," https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[2] C.-Y. Hong, S. Mandal, M. Al-Fares, M. Zhu, R. Alimi, K. N. B., C. Bhagat, S. Jain, J. Kaimal, S. Liang, K. Mendelev, S. Padgett, F. Rabe, S. Ray, M. Tewari, M. Tierney, M. Zahn, J. Zolla, J. Ong, and A. Vahdat, "B4 and after: Managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 74–87.

[3] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a globally-deployed software defined wan," in *Proceedings of ACM SIGCOMM*, 2013, pp. 3–14.

[4] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proceedings of ACM SIGCOMM*, 2013, pp. 15–26.

[5] "Building Express Backbone: Facebook's new long-haul network," https://code.facebook.com/posts/1782709872057497/building-express-backbone-facebook-s-new-long-haul-network/, 2017.

[6] "Inside AT&T's grand plans for SDN," https://www.networkworld.com/article/2866439/sdn/inside-atts-grand-plans-for-sdn.html, 2015.

[7] M. Birk, G. Choudhury, B. Cortez, A. Goddard, N. Padi, A. Raghuram, K. Tse, S. Tse, A. Wallace, and K. Xi, "Evolving to an SDN-enabled isp backbone: key technologies and applications," *IEEE Communications Magazine*, vol. 54, no. 10, pp. 129–135, 2016.

[8] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, and C. Diot, "Characterization of failures in an operational ip backbone network," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 749–762, 2008.

[9] D. Turner, K. Levchenko, A. C. Snoeren, and S. Savage, "California fault lines: Understanding the causes and impact of network failures," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10, 2010, pp. 315–326.

[10] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: Measurement, analysis, and implications," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11, 2011, pp. 350–361.

[11] R. Potharaju and N. Jain, "When the network crumbles: An empirical study of cloud network failures and their impact on services," in *Proceedings of the 4th Annual Symposium on Cloud Computing*, ser. SOCC '13, 2013, pp. 15:1–15:17.

[12] R. Govindan, I. Minei, M. Kallahalla, B. Koley, and A. Vahdat, "Evolve or die: High-availability design principles drawn from googles network infrastructure," in *Proceedings of ACM SIGCOMM*, 2016, pp. 58–72.

[13] P. Pan, G. Swallow, and A. Atlas, "Fast Reroute Extensions to RSVP-TE for LSP Tunnels," Internet Requests for Comments, RFC Editor, RFC 4090, May 2005. [Online]. Available: https://tools.ietf.org/html/rfc4090

[14] M. Shand and S. Bryant, "IP Fast Reroute Framework," Internet Requests for Comments, RFC Editor, RFC 5714, January 2010. [Online]. Available: https://tools.ietf.org/html/rfc5714

[15] K. Lakshminarayanan, M. Caesar, M. Rangan, T. Anderson, S. Shenker, and I. Stoica, "Achieving convergence-free routing using failure-carrying packets," in *Proceedings of ACM SIGCOMM*, 2007, pp. 241–252. [Online]. Available: http://doi.acm.org/10.1145/1282380.1282408

[16] B. Yang, J. Liu, S. Shenker, J. Li, and K. Zheng, "Keep forwarding: Towards k-link failure resilient routing," in *Proceedings of IEEE INFOCOM*, April 2014, pp. 1617–1625.

[17] K.-W. Kwong, L. Gao, R. Guérin, and Z.-L. Zhang, "On the feasibility and efficacy of protection routing in ip networks," *IEEE/ACM Transactions on Networking*, vol. 19, no. 5, pp. 1543–1556, October 2011. [Online]. Available: http://dx.doi.org/10.1109/TNET.2011.2123916

[18] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proceedings of ACM SIGCOMM*, 2014, pp. 527–538.

[19] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[20] R. S. Bhatia, M. Kodialam, T. V. Lakshman, and S. Sengupta, "Bandwidth guaranteed routing with fast restoration against link and node failures," *IEEE/ACM Transactions on Networking*, vol. 16, no. 6, pp. 1321–1330, December 2008. [Online]. Available: http://dx.doi.org/10.1109/TNET.2008.919325

[21] F. Hao, M. Kodialam, and T. V. Lakshman, "Optimizing restoration with segment routing," in *Proceedings of IEEE INFOCOM*, April 2016, pp. 1–9.

[22] M. Suchara, D. Xu, R. Doverspike, D. Johnson, and J. Rexford, "Network architecture for joint failure recovery and traffic engineering," *SIGMETRICS Perform. Eval. Rev.*, vol. 39, no. 1, pp. 97–108, 2011.

[23] D. Applegate, L. Breslau, and E. Cohen, "Coping with network failures: Routing strategies for optimal demand oblivious restoration," in *Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '04/Performance '04, 2004, pp. 270–281.

[24] J. Zheng, H. Xu, X. Zhu, G. Chen, and Y. Geng, "We've got you covered: Failure recovery with backup tunnels in traffic engineering," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, 2016, pp. 1–10.

[25] B. Fortz and M. Thorup, "Robust optimization of OSPF/IS-IS weights," in *Proceedings of International Network Optimization Conference*, 2003, pp. 225–230.

[26] "Cisco WAN automation engine (WAE)," 2016, http://www.cisco.com/c/en/us/products/routers/wan-automation-engine/index.html.

[27] A. K. Bangla, A. Ghaffarkhah, B. Preskill, B. Koley, C. Albrecht, E. Danna, J. Jiang, and X. Zhao, "Capacity planning for the google backbone network," in *ISMP 2015 (International Symposium on Mathematical Programming)*, 2015.

[28] Y. Wang, H. Wang, A. Mahimkar, R. Alimi, Y. Zhang, L. Qiu, and Y. R. Yang, "R3: Resilient routing reconfiguration," in *Proceedings of ACM SIGCOMM*, 2010, pp. 291–302.

[29] B. Schlinker, R. N. Mysore, S. Smith, J. C. Mogul, A. Vahdat, M. Yu, E. Katz-Bassett, and M. Rubin, "Condor: Better topologies through declarative design," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, 2015, pp. 449–463.

[30] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, and J. Rexford, "Optimizing bulk transfers with software-defined optical wan," in *Proceedings of ACM SIGCOMM*, 2016, pp. 87–100.

[31] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 113–126.

[32] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, 2013, pp. 15–27.

[33] A. Fogel, S. Fung, L. Pedrosa, M. Walraed-Sullivan, R. Govindan, R. Mahajan, and T. Millstein, "A general approach to network configuration analysis," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*. Oakland, CA: USENIX Association, 2015, pp. 469–483. [Online]. Available: https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/fogel

[34] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17. New York, NY, USA: ACM, 2017, pp. 155–168. [Online]. Available: http://doi.acm.org/10.1145/3098822.3098834

[35] "Topology zoo," http://www.topology-zoo.org/.

[36] "Abilene traffic matrices," http://www.cs.utexas.edu/~yzhang/research/AbileneTM/, 2014.

[37] Y. Zhang, Z. Ge, A. Greenberg, and M. Roughan, "Network anomography," in *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, 2005, pp. 30–30.

[38] R. Potharaju and N. Jain, "When the network crumbles: An empirical study of cloud network failures and their impact on services," in *Proceedings of ACM Annual Symposium on Cloud Computing*, 2013, pp. 15:1–15:17.

[39] D. Applegate and E. Cohen, "Making intra-domain routing robust to changing and uncertain traffic demands: Understanding fundamental tradeoffs," in *Proceedings of ACM SIGCOMM*, 2003, pp. 313–324.

[40] H. Räcke, "Minimizing congestion in general networks," in *43rd IEEE Symposium on Foundations of Computer Science*, 2002, pp. 43–52.

[41] ——, "Optimal hierarchical decompositions for congestion minimization in networks," in *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008, pp. 255–264.

[42] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg, "COPE: Traffic engineering in dynamic networks," in *Proceedings of ACM SIGCOMM*, 2006, pp. 99–110.

[43] A. Ben-Tal, L. E. Ghaoui, and A. Nemirovski, *Robust Optimization*. Princeton, NJ: Princeton University Press, 2009.

[44] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM Review*, vol. 53, no. 3, pp. 464–501, 2011.

[45] D. Bertsimas and V. Goyal, "On the power and limitations of affine policies in two-stage adaptive optimization," *Mathematical programming*, vol. 134, no. 2, pp. 491–531, 2012.

[46] H. D. Sherali and W. P. Adams, *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*. Springer Science & Business Media, 2013, vol. 31.

[47] S. Even, A. Itai, and A. Shamir, "On the complexity of time table and multicommodity flow problems," in *Foundations of Computer Science, 1975., 16th Annual Symposium on*, Oct 1975, pp. 184–193.

[48] V. Heorhiadi, M. K. Reiter, and V. Sekar, "Simplifying software-defined network optimization using sol," in *Proceedings of USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, March 2016, pp. 223–237.

[49] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol label switching architecture," *RFC 3031*, 2001, https://tools.ietf.org/html/rfc3031.

[50] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *Proceedings of ACM SIGCOMM*, 2013, pp. 27–38.

[51] B. Anwer, T. Benson, N. Feamster, and D. Levin, "Programming slick network functions," in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, 2015, pp. 14:1–14:13.

[52] A. Gupta, M. T. Hajiaghayi, and H. Räcke, "Oblivious network design," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2006, pp. 970–979.

[53] M. Casado, T. Koponen, S. Shenker, and A. Tootoonchian, "Fabric: A retrospective on evolving sdn," in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 85–90.

[54] R. Soulé, S. Basu, P. J. Marandi, F. Pedone, R. Kleinberg, E. G. Sirer, and N. Foster, "Merlin: A language for provisioning network resources," in *Proceedings of ACM CoNEXT Conference*, 2014, pp. 213–226.

[55] G. Dantzig, *Linear Programming and Extensions*. Princeton, NJ: Princeton University Press, 1963.

[56] D. Handelman, "Representing polynomials by positive linear functions on compact convex polyhedra," *Pacific Journal of Mathematics*, vol. 132, no. 1, pp. 35–62, 1988.

[57] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 6.

[58] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke, "Optimal oblivious routing in polynomial time," *J. Comput. Syst. Sci.*, vol. 69, no. 3, pp. 383–394, 2004.

[59] C. Harrelson, K. Hildrum, and S. Rao, "A polynomial-time tree decomposition to minimize congestion," in *Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2003, pp. 34–43.

[60] M. Bienkowski, M. Korzeniowski, and H. Räcke, "A practical algorithm for constructing oblivious routing schemes," in *Proceedings of the Fifteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2003, pp. 24–33.

[61] A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski, "Adjustable robust solutions of uncertain linear programs," *Mathematical Programming*, vol. 99, no. 2, pp. 351–376, 2004.

[62] "GEANT network," http://geant3.archive.geant.net/Network/NetworkTopology/pages/home.aspx.

[63] "IBM ILOG CPLEX optimization studio," http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer.

[64] "Mininet," http://mininet.org/.

[65] "Ostinato network traffic generator and analyzer," http://ostinato.org/.

[66] D. Sidhu, R. Nair, and S. Abdallah, "Finding disjoint paths in networks," in *Proceedings of the Conference on Communications Architecture &Amp; Protocols*, 1991, pp. 43–51.

[67] M. T. Hajiaghayi, R. D. Kleinberg, and T. Leighton, "Semi-oblivious routing: Lower bounds," in *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 2007.

[68] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov, "New variants of bundle methods," *Mathematical programming*, vol. 69, no. 1-3, pp. 111–147, 1995.

[69] A. Elwalid, C. Jin, S. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *Proceedings of IEEE INFOCOM*, 2001, pp. 1300–1309.

[70] S. Kandula, D. Katabi, B. Davie, and A. Charny, "Walking the tightrope: Responsive yet stable traffic engineering," in *Proceedings of the 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, 2005, pp. 253–264.

[71] C. Zhang, Z. Ge, J. Kurose, Y. Liu, and D. Towsley, "Optimal routing with multiple traffic matrices tradeoff between average and worst case performance," in *Network Protocols, 2005. ICNP 2005. 13th IEEE International Conference on*, 2005.

[72] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. D. Kleinberg, and R. Soulé, "Kulfi: Robust traffic engineering using semi-oblivious routing," *CoRR*, 2016.

[73] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proceedings of ACM SIGCOMM*, 2012, pp. 13–24.

[74] G. A. Hanasusanto, D. Kuhn, and W. Wiesemann, "K-adaptability in two-stage robust binary programming," *Operations Research*, vol. 63, no. 4, pp. 877–891, 2015.

[75] P. Awasthi, V. Goyal, and B. Y. Lu, "On the adaptivity gap in two-stage robust linear optimization under uncertain constraints," 2015, http://www.columbia.edu/~vg2277/column-wise.pdf.

[76] M. Ghobadi and R. Mahajan, "Optical layer failures in a large backbone," in *Proceedings of the 2016 Internet Measurement Conference*, 2016, pp. 461–467.

[77] F. Shahrokhi and D. W. Matula, "The maximum concurrent flow problem," *J. ACM*, vol. 37, no. 2, pp. 318–334, 1990.

[78] R. Wood, "Deterministic network interdiction," *Mathematical and Computer Modelling*, vol. 17, no. 2, pp. 1–18, January 1993.

[79] D. S. Altner, Ö. Ergun, and N. A. Uhan, "The maximum flow network interdiction problem: Valid inequalities, integrality gaps, and approximability," *Operations Research Letters*, vol. 38, no. 1, pp. 33 – 38, 2010.

[80] W. E. Hart, J.-P. Watson, and D. L. Woodruff, "Pyomo: modeling and solving mathematical programs in python," *Mathematical Programming Computation*, vol. 3, no. 3, pp. 219–260, 2011.

[81] J. C. Mogul, R. Isaacs, and B. Welch, "Thinking about availability in large service infrastructures," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, ser. HotOS '17, 2017, pp. 12–17.

[82] "Preliminary report of the effects of hurricane sandy on communication networks." 2012, http://users.ece.utexas.edu/~kwasinski/sandy.html.

[83] R. K. Sinha, F. Ergun, K. N. Oikonomou, and K. K. Ramakrishnan, "Network design for tolerating multiple link failures using Fast Re-route (FRR)," in *2014 10th International Conference on the Design of Reliable Communication Networks (DRCN)*, April 2014, pp. 1–8.

[84] Y. Chang, S. Rao, and M. Tawarmalani, "Robust validation of network designs under uncertain demands and failures," in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017, pp. 347–362.

[85] "Topology zoo," http://www.topology-zoo.org/.

[86] "NIST/SEMATECH e-Handbook of Statistical Methods," http://www.itl.nist.gov/div898/handbook/prc/section2/prc242.htm.

[87] G. O. Inc., "Gurobi optimizer reference manual," 2016, http://www.gurobi.com.

[88] P. Kumar, Y. Yuan, C. Yu, N. Foster, R. Kleinberg, P. Lapukhov, C. L. Lim, and R. Soulé, "Semi-oblivious traffic engineering: The road not taken," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 157–170.

[89] A. Gember-Jacobson, R. Viswanathan, A. Akella, and R. Mahajan, "Fast control plane analysis using an abstract representation," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 300–313.

[90] R. Beckett, R. Mahajan, T. Millstein, J. Padhye, and D. Walker, "Don't mind the gap: Bridging network-wide objectives and device-level configurations," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 328–341.

[91] R. Beckett, A. Gupta, R. Mahajan, and D. Walker, "A general approach to network configuration verification," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 155–168.

[92] N. Foster, D. Kozen, K. Mamouras, M. Reitblatt, and A. Silva, "Probabilistic netkat," in *Proceedings of the 25th European Symposium on Programming Languages and Systems - Volume 9632*, 2016, pp. 282–309.

[93] G. Angulo, S. Ahmed, S. S. Dey, and V. Kaibel, "Forbidden vertices," *Mathematics of Operations Research*, vol. 40, no. 2, pp. 350–360, 2015.

VITA

VITA

Yiyang Chang is a Ph.D. Candidate in the School of Electrical and Computer Engineering at Purdue University, advised by Professor Sanjay Rao. His research interests are in Computer Networking Systems. His research currently focuses on formal approaches to validate network performance under failures. Prior to his doctoral studies, he received his B.S. degree from School of EECS, Peking University, China.