Raha: A General Tool to Analyze WAN Degradation

Behnaz Arzani
Microsoft Research

Sina Taheri
Microsoft

Pooria Namyar
Microsoft Research, USC

Ryan Beckett
Microsoft Research

Siva Kesava Reddy Kakarla
Microsoft Research

Elnaz Jallilipour
Microsoft

Abstract

Raha is the first general tool that can analyze probable *degradation* of traffic engineered networks under *arbitrary* failures and traffic shifts to prevent outages. Raha addresses a significant gap in prior work which consider only (1) $\leq k$ failures; (2) specific traffic engineering schemes; and (3) the maximum impact of failures irrespective of the network design point.

Our insight is to formulate the problem in terms of heuristic analysis, where one seeks to maximize the performance *gap* between the network design point (*i.e.,* the network with no failures) and the network under failures. We invent techniques that allow us to exploit the mechanisms within these tools to encode the problem into components which can handle them. We present extensive experiments on Microsoft's production network and those of Topology Zoo that demonstrate Raha is scalable and can effectively solve the problem. We use Raha to propose capacity augments that allow operators to mitigate potential problems and avoid future outages. Our results show Raha can find $\geq 2\times$ higher degradations compared to those tools that only consider up to 2 failures.

CCS Concepts

• **Networks** → **Traffic engineering algorithms**; **Network design and planning algorithms**; **Network performance modeling**; *Network properties.*

Keywords

Traffic engineering, Network performance analysis, Network reliability

ACM Reference Format:
Behnaz Arzani, Sina Taheri, Pooria Namyar, Ryan Beckett, Siva Kesava Reddy Kakarla, and Elnaz Jallilipour. 2025. Raha: A General Tool to Analyze WAN Degradation. In *ACM SIGCOMM 2025 Conference (SIGCOMM '25), September 8–11, 2025, Coimbra, Portugal.* ACM, New York, NY, USA, 19 pages. https://doi.org/10.1145/3718958.3754348

1 Introduction

Failures and unanticipated traffic shifts in wide-area networks (WANs) continue to cause outages [13, 21, 28, 31, 34] because it is hard for operators to predict how much these changes can *degrade* a network's performance (compared to what operators designed it for). It is hard to provision and plan capacity in a WAN because operators must consider how the large space of possible (and variable) traffic patterns and failure scenarios jointly impact network performance.

Today, operators capacity plan [2, 12] (ensure there is enough capacity to route peak traffic), traffic engineer [6, 27] (route traffic subject to capacity constraints), and simulate (test the WAN performs well against common changes [9, 26, 38]) their WAN to avoid outages. While each of these techniques is effective in isolation, the WAN's performance can still degrade after a series of failures occurs *concomitantly* with variable traffic.

For example, link failures caused significant downtime in our production WAN in Africa. Like most cloud providers we capacity plan our WAN so that it is resilient to up to k failures [2, 6, 12, 26, 27], perform safety checks [31, 39], and rely on resilient traffic engineering (TE) pipelines [22]. We also simulate the WAN traffic under different failure scenarios and augment capacity if failures impact the WAN's performance given the *maximum* traffic we expect it to carry. But multiple consecutive natural disasters and changing demands caused our network to become unable to carry the traffic we expected it to and our simulator (which simulates each possible failure combination under peek load) failed to detect it in time.

Existing tools can not help avoid such incidents (Table 1). This is because none can *simultaneously* account for (1) arbitrary failure scenarios, (2) arbitrary changes in traffic demands (*e.g.,* shifts of up to 30% from average), (3) a wide variety of traffic engineering formulations and objectives such as total demand met, max-min fairness, or maximum link utilization (MLU), (4) any tunnel selection scheme, and

Approach	Relative Impact	Arbitrary Failures	Variable Demands	General Objectives
TeaVaR [6]	✗	✓	✗	✗
QARC [38]	✗	✓	✓	✗
Yu [26]	✗	✗	✓	✗
Robust [9]	✗	✓	✓	✗
RAHA (ours)	✓	✓	✓	✓

Table 1: The capabilities and generality of RAHA compared to prior worst-case TE analysis work.

(5) the *degradation* in performance relative to what the network design allows. The latter is particularly hard: if we naively solve for the set of failures and demands that jointly cause the worst objective (*e.g.,* total demand routed) we often find a scenario that is not practically relevant. For example, one can trivially reduce the traffic the network routes to zero if we set the demands to zero.

What operators care about is how the network performs for the given traffic demands *relative* to the network without failures – *i.e.,* the gap in performance due to the failures. Some of these aspects (*e.g.,* the total demand met objective and accounting for arbitrary changes in demand) conflict with each other and are hard to *jointly* account for (see §4).

Prior work such as TeaVaR [6] and FFC [27] can model scenarios with limited number of failures (FFC puts a limit k on the number of failures, TeaVaR prunes the failure scenarios to reduce the combinations it models) but require a concrete set of demands as input. More recent work can solve for both failures and changes in demand, but only for specific aligned objectives such as MLU [9, 38] or for networks with simple path selection schemes such as shortest path routing [38].

RAHA[1] is the first general tool that allows operators to simultaneously consider (1) all failure scenarios (and the probability with which they happen), (2) the *degradation* in network performance under *variable* traffic, (3) any tunnel selection (*e.g.,* oblivious routing [4] or k shortest path) and commonly used traffic engineering scheme such as those that optimize for max-min fairness [32] (Appendix A), total demand met [15, 16] (§5), or those that minimize the MLU [9, 33, 38] (Appendix A), (4) how the traffic engineering solution adapts to failures, and (5) augments to network capacity to mitigate probable degradations. RAHA can model partial failures — it can capture models when a portion of the LAG capacity goes down — and shared risk groups (SRLGs). We built RAHA to avoid incidents like the one above.

Operationally we use RAHA both offline to provision the network (see § 7) and online to raise alerts when it finds failures can cause significant impact. RAHA produces two types of alerts: (1) it alerts within 10 minutes if there exists a probable failure scenario that would cause the network performance to significantly degrade when it routes the peak traffic between each node pair (based on past history); and, if not, (2) it continues to check whether there exists *any* demand[2] that can cause such a problem and alerts (within ≤ an hour) if so.

RAHA solves a hard problem. To find if failures can cause the network's performance to degrade we need to simultaneously model and compare the performance of both the designed network and the network under failure. We also have to account for *all possible failure scenarios* and the network's reaction to each under all possible traffic patterns — the search space grows exponentially with the number of nodes, links, and fail-over policies.

The insight that allowed us to overcome these challenges was to view the problem as a heuristic analysis one: where we find the set of inputs (failures and demands) that maximize the gap between an optimal algorithm and an algorithm (heuristic) that approximates that optimal. Here, this "optimal" is the network's design point and the "heuristic" is the network under failure. This view allows us to use existing tools such as MetaOpt [29, 30] to find whether there exists a *probable* failure scenario that can impact the network: the heuristic analyzer is able to navigate the large search space and finds the failures and the demands that maximize the reduction in the failed network's ability to carry traffic compared to it's original design.

To use MetaOpt[3] we need to model the "heuristic" (the network under failure) as either a convex or feasibility problem. While the optimal in most traffic engineering solutions is already convex [6, 15, 16, 32]; this is not true of the failed network. This network is hard to model as a convex problem because we have to model (1) all possible link failure combinations and their probability (which involve binary variables and non-convex constraints); and (2) the network's reaction to them (which again involves non-convex constraints). To solve this problem we exploit the bi-level structure of the optimization MetaOpt solves and "extract" the non-convexity out of the heuristic model and move it to "outer" constraints where MetaOpt can handle them. Our contributions are:

- We build RAHA that models any number of link/LAG failures (and their probability), all possible traffic demands, and many common TE solutions (see § 5 and Appendix A). RAHA is open source [3].
- We use heuristic analysis tools to analyze the performance degradation of networks by exploiting the inner mechanics

[1] RAHA means free. We dedicate this work to all those around the world who are far from home. We hope for a day where we all live in a world where distance from family, friends, and "home" is a choice.

[2] Operators can choose to run this step with additional constraints that restrict the type of demands RAHA considers.

[3] Other SMT-based analyzers, are harder to apply and scale.

of these tools to extract non-convexities out of our heuristic model and into constraints these tools can handle.

- We describe a novel approach to augment existing LAG capacities to remove all *probable* performance degradations.
- We evaluate RAHA on our production topology in Africa and on topologies from Topology Zoo [20]. Overall we conducted over 10,000 experiments. As part of these we discuss how to estimate Link failure probabilities in practice §8 and lessons learned from running RAHA §9. Our results show that we find $\geq 2\times$ higher degradations compared to prior works that consider up to $k = 2$ failures.

Ethics. This work does not raise any ethical issues.

2 Motivation

The incident in our WAN shows why it is important to quickly detect whether a network is vulnerable to upcoming failures. A seismic event caused multiple fiber cuts, which alongside changing demands, and a faulty line card caused our WAN to become congested.

2.1 RAHA example

We show a highly simplified traffic engineering (TE) example (Figure 1) to motivate why we need to jointly reason over the network design point, the worst case failures, and traffic changes. The network has four nodes (A, B, C, D) and routes demands from B to D and C to D. We configure two paths for each node pair. B can use paths BD and BAD while C can use paths CD and CAD. We consider all single-link failures and the TE maximizes the total demand the network carries. We show three scenarios for the same network from left to right with the worst case failure and the network design point (the network with no failures) below it.

In the first scenario (left) we set the demands to fixed values equal to "typical" demands from B to D (d_{BD}) and C to D (d_{CD}). The network with no failures routes all 22 units, whereas the worst case failure of the BD link causes it to only route 15. This degrades the network performance by **7 units** (the difference in what the two networks carry).

The middle scenario shows what happens if we allow the demands to vary by up to 50% from the average ($6 \leq d_{BD} \leq 18$ and $5 \leq d_{CD} \leq 15$) — it shows if we naively optimize for the demands and failures that produce the worst case objective. The solution will find demands $d_{BD} = 6$ and $d_{CD} = 5$ and will fail link CD. The network routes only 10 units but this "poor performance" is really just because of the choice of (small) demands and not because of the failures. The network will route 11 units when all it's links are up — the network performance degrades only by **1 unit**!

RAHA (right), which considers the full space of all possible demands between each pair, finds demands $d_{BD} = 13$ and $d_{CD} = 12$ and brings the AD link down. The failed network can carry only 16 units whereas the network with no failures carries 25 for a maximal performance degradation of **9 units**.

2.2 Limitations of existing work

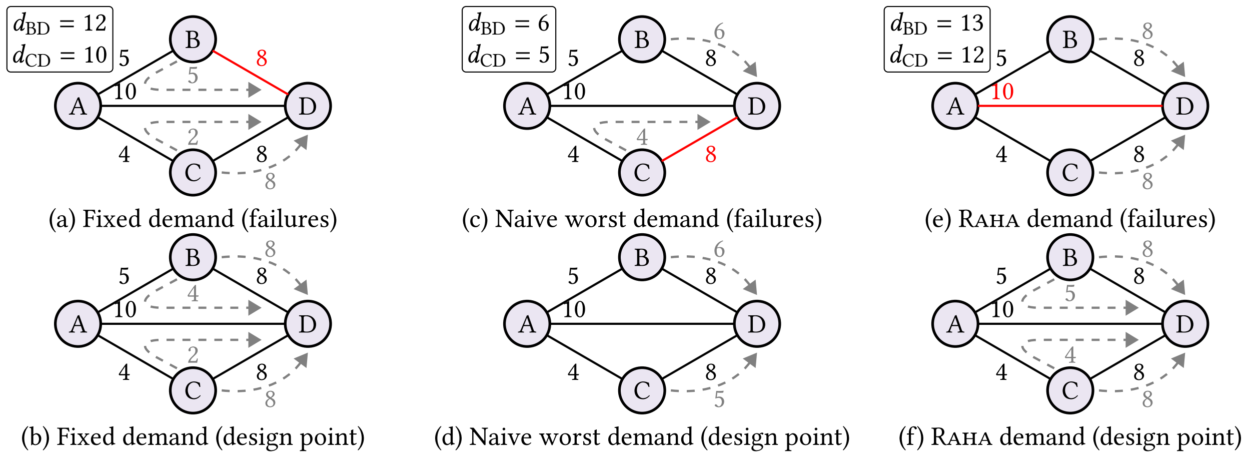
One may ask whether the tools the community has built such as those for traffic engineering [6, 27] or verification [9, 26, 38] could solve this problem and could have prevented the outage in our WAN. The answer is no (see Table 1). Tools [26, 27] that only consider up to k-failures, where k is typically ≤ 2, would not prevent the incident.

Authors of these tools argue "probable" failures are those where k is small. We show even when we target a *moderate* availability of 99%, the maximum number of link failures (we model edges as LAGs that consist of multiple links) that can *simultaneously* occur within this probability constraint can be as high as 15–20 (Figure 2). While such *probable* failures are rarely impacting (this is why incidents like the one above don't happen every day) there are situations where they can degrate performance by $\geq 30\%$.

Today, most operators aim to provide $\geq$ 4-9's availability which means the number of link failures we need to consider is even higher. A naive solution to this problem may be to rerun TE solutions such as FFC [27] or TeaVaR [6] immediately after the failure happens but, as incident above showed, this can lead to a situation where there is not enough capacity left for these solutions to solve the issue (and by the time this happens we would have lost our lead time to react). RAHA aims to complement these solutions and runs immediately after each failure occurs to check whether there exists a probable failure that can significantly impact our network.

Those works that consider any number of failures [9, 38] focus on specific TE algorithms. For instance, QARC [38] only models networks that route traffic over their shortest path. Both QARC and [9] only model those TE solutions that minimize the maximum link utilization (MLU). The MLU objective is "aligned" to this problem: to maximize the MLU the optimization benefits from increasing the demands between node pairs. This is not true of other objectives such as those that maximize total flow — without any additional incentives the optimization sets the demand to zero and produces a trivial solution (Figure 1)!

This leads to a second, more fundamental limitation of these works: none find the failure scenario that truly matters in practice — they focus on the failures and demands that minimize the performance of the failed network but do not consider how this failed network performs relative to its design point. Most operators (including us) design their network so that it is resilient to failure scenarios within a demand envelope [2, 12] — chances are, the original network also underperforms in the scenarios these tools find. These



d_{BD} = 12
d_{CD} = 10
d_{BD} = 6
d_{CD} = 5
d_{BD} = 13
d_{CD} = 12
(a) Fixed demand (failures)
(c) Naive worst demand (failures)
(e) Raha demand (failures)
(b) Fixed demand (design point)
(d) Naive worst demand (design point)
(f) Raha demand (design point)

Figure 1: Examples that show why we need to jointly analyze the worst-case impact of both failure and traffic demand changes. We show three cases: a fixed traffic matrix (a, b), naively finding the worst case failures and demands (c,d), and Raha's solution in (e,f). All links are bidirectional with link capacities in black. Failed links are red and we use dashed gray lines to indicate the demand the network routes on different paths.

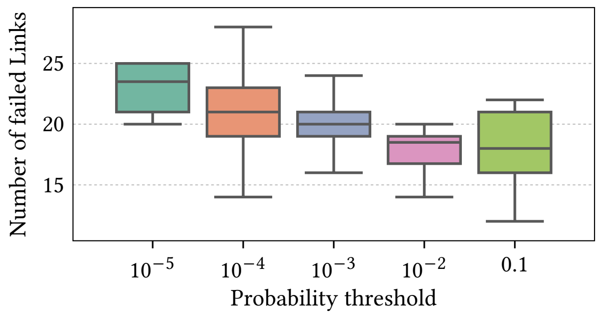


Figure 2: The x-axis shows failure probability thresholds: we consider failures with at least this probability ($\geq$ threshold). The number of links that can *simultaneously* fail decreases as the threshold increases, since higher thresholds exclude rare large-scale failures. Experiments include different numbers of paths.

are scenarios operators need to (and do) address in capacity planning and where these tools provide value but alerts based on them would cause too many false positives.

In contrast, Raha allows operators to analyze the failed network: (1) under *any* demand with the option to narrow the space to those that are of concern in practice (*e.g.*,, through a slack parameter Figure 3); (2) under *all* failure scenarios; and (3) relative to the healthy network.

2.3 How Raha helps

We use Raha to show the importance of accurate joint analysis in Figure 3. We find the scenario that minimizes the network's performance (*i.e.,* similar to [9, 38] we do not consider the degradation). We use a fixed demand and set it

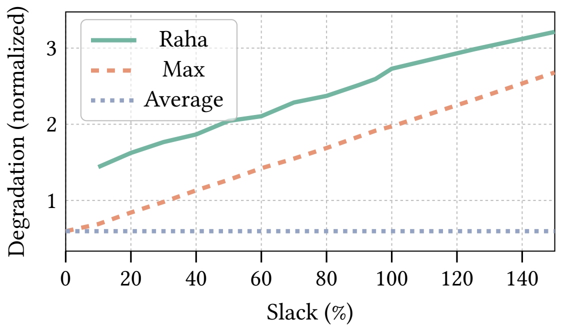


Figure 3: We compare RAHA with naive baselines that use the peak demand within the range to find the worst-case failures for a given topology. The non-Raha approaches minimize network performance (and not the degradation). A degradation of 2 here means the network drops traffic equivalent to 2× the average capacity of a LAG in the network.

to the average over a month-long period, progressively allow it to increase by up to some increment (the slack), and each time search for the set of failures that minimize the network's performance. We then subtract this value from the performance of that of the network's design point with the same demands but no failures. We compare the outcome to what Raha finds when it searches for both the demand and the failure scenario and directly maximizes the degradation within that demand range.

These results reveal another subtlety about why the choice of demands needs to depend on the network's design point. Intuitively one may think: "the goal is to route as much demand as possible, if we set the demand for both networks

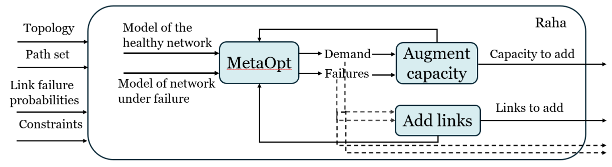


Figure 4: Raha and its internals. It produces an example demand and a set of failures that cause the worst-case impact. Raha can also produce where to add capacity and/or links in order to mitigate this impact.

to it's peak then that should also reveal the maximum performance degradation (if we find the right set of failures)". Clearly, this is not what we find. This is because of how our network adapts to failures: each node-pair has a number of backup paths available to it (1 in this experiment) which it cannot use unless its primary path fails. This is why we may not be able to degrade performance if we increase the demand between certain node-pairs — it all depends on the amount of capacity on the backup path between those nodes, the LAGs they share with other primary path, etc.

3 Raha Overview

Our insight is to use heuristic analysis tools to help us navigate the large search space which includes all possible failure combinations, demands, and the network's reactions to each.

Figure 4 shows Raha's design. It takes as input: (1) the network topology; (2) a set of paths (this is why Raha supports any path selection policy — it runs k shortest path if this input is missing); (3) the link failure probabilities (this is optional, without it we revert to $\leq k$ failure analysis); and (4) any constraints operators want to add (*e.g.,* they may want to only find scenarios that are likely to happen with probability $\geq T$). It then uses these inputs to model the healthy network and the network under failure (and it's reaction to these failures) and feeds them into a heuristic analysis tool which then produces an example demand matrix (operators can also specify it) and a set of failures that cause the maximum impact given the constraints operators imposed. If the impact goes beyond the operator's tolerance levels, then Raha raises an alert to notify them.

Raha can also find where operators should augment capacity and mitigate *probable* scenarios that degrade performance. Raha supports any WAN that uses a single shot optimization for traffic engineering (*e.g.,* the geometric or equi-depth binning WANs in [32], MLU [9, 38], SWAN [15], or B4 [16]). It also can support many different types of reactions to failures, whether local or global. Without loss of generality, in this paper, we use our production WAN as an example of how to model such problems and discuss how to extend this formulation to a few other objectives in Appendix A.

4 Background

We next provide the necessary background: (1) how MetaOpt analyzes heuristics (see §4.1); and (2) how our WAN routes traffic and reacts to failures (see §4.2).

4.1 How MetaOpt analyzes heuristics

We model Raha with MetaOpt [29, 30]. MetaOpt solves the heuristic analysis problem as a Stackelberg game. The game involves an adversary leader who controls the input ($I \in \mathcal{I}$) to the optimal and heuristic algorithms and aims to maximize the difference in performance between the two. The optimal and heuristic algorithms take the leader's input and maximize their own respective objectives through other variables they can control. We refer to the problem the leader solves as the outer problem and the ones the heuristic and optimal solve as the inner problems.

The game translates to a bi-level optimization:

$$\begin{aligned} \text{Outer problem:} \quad & \max_{I \in \mathcal{I}} \quad \mathrm{H}(I) - \mathrm{H}'(I) \qquad (1)\\ & s.t. \quad \text{Constraints}(I) \\ \text{Inner problem 1:} \quad & H(I) = \max_{f^o} \quad \text{Optimal}(I, f^o) \\ \text{Inner problem 2:} \quad & H'(I) = \max_{f^h} \quad \text{Heuristic}(I, f^h), \end{aligned}$$

where the optimal and the heuristic optimizations may themselves contain additional constraints on f^o and f^h. Operators can add constraints on the types of inputs the outer problem can choose from (Constraints(I)). For example, they may restrict the space of inputs to ensure there are only those that they expect to encounter in practice (refer to [30] for more details).

In Raha, the input variables $\mathcal{I}$ are network demands (how much traffic each node wants to send to each other node) and the set of failures. The variables f^o and f^h are the amount of flow each of the two networks routes for each demand. The formulation (by definition) ensures the solution is indeed the failure scenario and the demand matrix that cause the worst-case performance degradation.

The outer problem, the first inner problem, and their constraints need not be convex and can include binary and integer variables but the second inner problem has to be either a convex optimization problem or a feasibility problem (*i.e.,* a satisfiability or SAT [10] problem). We exploit this to model the network under failures in a way that MetaOpt can solve: we use the fact that the variables of the outer problem (*i.e.,* those we have highlighted in blue) are treated as constants by the inner problems to "extract" out the non-convexity into the outer problem where MetaOpt can handle them.

4.2 How our WAN operates

We next describe our WAN traffic engineering (TE) pipeline. We only describe components which we need to model in Raha and omit details about our dataplane simulator, capacity planner, and safety checker due to space constraints.

The WAN of our large public cloud serves millions of customers each day and routes multiple petabytes of traffic. We use a centralized TE algorithm similar to SWAN [15] and B4 [16] to route traffic. These algorithms take the topology where each edge is a link aggregation group (LAG) comprised of a bundle of physical links, a set of paths over these LAGs, and a demand matrix, $\mathcal{D}$, as input and find the optimal flow allocations on each path f_{kp} which routes the most traffic without going over the capacity of any LAG:

$$\max_{f_k} \sum_{k \in \mathcal{D}} f_k, \quad s.t. \qquad (2)$$

$$\texttt{Demand constraints:} \quad 0 \leq f_k \leq d_k \quad \forall k \in \mathcal{D}$$

$$\texttt{Flow constraints:} \quad f_k = \sum_{p \in \mathcal{P}_k} f_{kp} \quad \forall k \in \mathcal{D}$$

$$\texttt{Capacity constraints:} \sum_{\forall k \in \mathcal{D}, p \in \mathcal{P}_{ke}} f_{kp} \leq C_e \quad \forall e \in \mathcal{E}$$

where $\mathcal{P}_k$ is the set of paths available to demand k (see Table 2 for notation). To adapt to changes to the demand or topology we resolve this optimization periodically. This model also describes the optimal we input into MetaOpt as it models how the healthy network routes traffic.

The network reacts to failures through backup paths: paths that we configure for each source-destination pair (demand) and which the network fails-over to one after the other when each primary path between the source-destination pair fails. We next describe how we model the network under failure in MetaOpt and then describe how we move the non-convexities involved into the outer problem.

5 Modeling unhealthy networks

We need to capture: (1) the change in capacity failures cause; and (2) how the network adapts to them. To model both of these aspects we need to control which links, LAGs, and which paths can carry traffic. But we need to do so in a way that preserves the convexity of the inner problem.

We use the model in Equation 2 and modify the capacity constraints (which ensure the flow going through a LAG is below its capacity). We introduce a new continuous variable for each LAG in the *outer* problem, c_e, which is the variable capacity of LAG e. We also add "path extension capacity variables", c_{kp}: artificial LAGs that we add to each path which are also variables of the outer problem. These artificial LAGs control whether traffic can flow through the paths — their

Variable	Description
$\mathcal{E}$	The set of all the edges, or LAGs, in the network.
$\mathcal{D}$	The set of all demands.
d_k	The traffic volume source s_k has for destination t_k.
$\mathcal{P}_k$	The set of paths available for the k^{th} demand.
$\mathcal{P}_{ke}$	The paths that go through LAG e for the k^{th} demand.
$\mathcal{B}_k$	The set of backup paths for the k^{th} demand.
C_e	The capacity of LAG e.
c_{le}	The capacity of link l of LAG e.
c_e	The variable capacity of LAG e which is in $\{0, C_e\}$.
c_{kp}	The path extension capacity variable for path p_k.
u_{kp}	Whether the path p_k is down or not (binary variable).
u_e	Whether LAG e is down or not (binary variable).
u_{le}	Whether link l from LAG e is down or not (binary variable).
f_k	The total flow we route for demand k in the network.
f_{kp}	The flow for demand k routed on path p.
f_{ijk}	The flow for demand k on the edge between nodes i and j.
N_e	The number of links that are part of LAG e.
N_{kp}	The number of edges/LAGs on path p for demand k.
n_{kp}	The number of primary paths for demand k.
p_{kj}	The j^{th} path for demand k where the first n_{kp} are primary.
π_{le}	The probability that link l on LAG e is failed.
T	Threshold on the failure probability scenario.

Table 2: Encoding variables. We highlight the variables that are constant values in red, and those that are treated as variables for the outer problem but constants for the inner problem in blue. Black values are variables in both the inner and outer problem.

capacity constraints ensure the flow going through them and the corresponding path, p_k, does not exceed c_{kp}.

The inner problem treats these new variables as constants and therefore it remains convex. But we need to add constraints to the *outer* problem to ensure the inner problem accurately models the unhealthy network — this is how we extract the non-convexity into the outer problem.

The capacity of each regular LAG (in the original topology) depends on which of its constituent links have failed:

$$c_e = \sum_{l \in e} c_{le} \cdot (1 - u_{le}) \quad \forall e \in \mathcal{E},$$

where u_{le} is binary (it is 1 when the link l is down).

Setting the path extension capacity is harder. We need to find: (1) when the path is down; (2) and when we are allowed to use backup paths (we can only use the r^{th} backup path if r higher priority paths are down).

A path is down when a LAG is down and a LAG is down when all of its links are down:

$$N_e \cdot u_e + \text{aux} = \sum_{l \in e} u_{le} \quad 0 \leq \text{aux} \leq N_e - 1,\ \forall e \in \mathcal{E}, \tag{3}$$

$$N_{kp} \cdot u_{kp} \geq \sum_{e \in p} u_e \quad \forall k \in \mathcal{D},\ p \in \mathcal{P}_k \tag{4}$$

where N_e is the number of links for LAG e and Equation 3 ensures a LAG goes down only when all of its links go down, and Equation 4 ensures the path goes down when any of its LAGs go down. The outer problem decides the values for the variables in blue and they are constants in the inner problem. The values in red are constants in both problems.

With u_{kp} to indicate if a path is down, we can set the path extension capacities in a way only allows the r^{th} backup path to become active when *at least* r other primary and higher priority backup paths have failed. We create an ordered list of paths where the first n_{kp} are the primary path and the remaining are an ordered list of backups, then:

$$c_{kp_j} = d_k \cdot \mathcal{I}(\sum_{0 \leq i \leq j} u_{kp_i} + n_{kp} - j)$$
$$\forall k \in \mathcal{D}, p \in \mathcal{P}_k, j \in \{0..n_{kp} + n_{kb}\} \tag{5}$$

where $\mathcal{I}(x)$ is an indicator function which we linearize through standard optimization techniques [7]. Notice the output of $\mathcal{I}$ is always 1 if j is one of the primary path ($n_{kp} - j \geq 0$) and for the r^{th} backup path is 1 iff at least r higher priority paths have failed ($n_{kp} - j = -r$). We used n_{kp} and n_{kb} to indicate the total number of primary and backup paths.

5.1 How to constrain the problem

Often, we want to enforce other constraints on this problem. For example, we run RAHA to consider all *probable* failures (those that can happen with probability ≥ some threshold). We discuss example constraints but users can add others:

Only consider probable failures or scenarios. Users can constrain RAHA to only consider probable scenarios (which occur with probability $\geq T$). For instance, we can set T to 0.00001. This translates to:

$$\prod_{e \in \mathcal{E}, l \in e} \pi_{le}{}^{u_{le}} \cdot (1 - \pi_{le})^{(1-u_{le})} \geq T$$

where π_{le} is the probability that a link fails and is constant. This formulation is non-convex. To turn it into a convex problem we take the log of both sides (log is monotonic):

$$\sum_{e \in \mathcal{E}, l \in e} u_{le} \cdot \log \pi_{le} + \sum_{e \in \mathcal{E}, l \in e} (1 - u_{le}) \cdot \log(1 - \pi_{le}) \geq \log T$$

Notice how the above formulation automatically captures *the entire* space of failures and then ensures the optimization only allows those that are above the probability threshold — this is what allows us to go beyond k failure analysis.

Only allow up to k failures. Similar to prior work [26]:

$$\sum_{e \in \mathcal{E}, l \in e} u_{le} \leq k$$

Consider the worst case failure for a specific demand. Operators can set the demands in Equation 2 (d_k becomes constant d_k). The healthy network's inner optimization turns into a constant and RAHA only outputs the failure scenario that creates the maximum impact with the given demand.

Enforce a strongly connected network. This constraint is one we occasionally use to analyze our production network: we do not consider failures which cause all paths between a source-destination pair to go down. We model it as:

$$\sum_{p \in \mathcal{P}_k} u_{kp} < |\mathcal{P}_k| \quad \forall k \in \mathcal{D}$$

Consider naive fail-overs. Our model assumes the network optimally uses backup paths (the best case reaction). But operators may choose to model a naive reaction where the backup path is only a direct fail-over (or other variants). This introduces additional constraints:

$$f_{kp_{n_{kp}+r}} \leq f^o_{kp_r} \quad \forall k \in \mathcal{D}, \forall r \in \{0..|\mathcal{B}_k|\}$$
$$f_{kp} \leq f^o_{kp} \quad \forall k \in \mathcal{D},\ p \in \mathcal{P}_k \setminus \mathcal{B}_k$$

where p_{n_p+r} is the r^{th} backup path for demand k and p_r is the r^{th} primary path. The variable f^o denotes the flow assignment in the healthy network.

6 How to Scale

If the operator provides a fixed demand matrix to RAHA, scaling is easy. This is because the solution to the optimal problem (healthy network) effectively becomes a constant value that RAHA solves independently. Maximizing the gap between the optimal and heuristic (network with failures) then translates to the problem of finding the failures that maximize the gap between a constant and the heuristic encoding.

Raha can take longer (≥ 2 hours) if we use it to simultaneously find both the demand and the failure scenario on a large topology. In our network, we use RAHA to analyze each continent's WAN network (see § 9) and the network that connects these continents together separately. We also

Algorithm 1: Finding demands through clustering.

```
Input: Topology ($T$)
Input: Link failure probabilities ($P$)
Input: Paths ($\mathcal{P}$)
Input: Operator specified constraints ($C$)
Output: Demand Matrix ($\mathcal{D}$)
1  $F \leftarrow$ Model(T,P, $\mathcal{P}$, C)
2  Clusters $\leftarrow$ GetClusters(T)
3  $F \leftarrow$ SetDemands(F, 0) //initialize to
     0
4  foreach $C_i \in$ Clusters do
5      foreach $C_j \in$ Clusters do
           // Notice we can have $C_i = C_j$.
6          foreach $s \in C_i$, $d \in C_j$ do
7              // set $d_{s,d}$ as unknown
8              $F \leftarrow$ UnSetDemand(F, s, d)
9          end
10         $\mathcal{D}_{cross\_cluster} \leftarrow$ Solve(F)
11         $F \leftarrow$ SetDemands(F, $\mathcal{D}_{cross\_cluster}$)
12     end
13 end
14 return $GetDemands(F)$
```

devise a clustering scheme to scale it further[4]. The clustering scheme finds a demand matrix that approximates the demand that causes the worst case impact and then, given this now constant demand, finds the failure scenario that maximizes the impact overall (Algorithm 1).

To find this approximate demand matrix we first model the problem on the full topology (line 1). Next, we divide the topology into (disjoint) clusters (line 2). We then search within and across clusters for local (to those clusters) demand matrices that maximize the overall impact of failures in the network (we still consider the full topology where the paths of these demands can go across multiple clusters; and all failures that can happen topology wide).

We go cluster by cluster and run MetaOpt to find local demands (where both the source and destination fall within the cluster) which cause the worst impact on the network when failures happen. Here, we fix all other demands to the value we found for them so far or zero if we have not yet done so. We then repeat this step for those demands where their source and destinations are in different clusters. We run MetaOpt again on the full problem with this fixed demand to find the failure scenario that causes the worst impact.

With this careful clustering we ensure we only approximate the demand: when we analyze each cluster, we still consider all failure scenarios, all paths (even those that exit the cluster), and all other demands that we have set so far.

[4]MetaOpt [30] also has a clustering scheme for TE, we extend it here.

We further scale our approach with MetaOpt's `timeout` feature which stops the solver if it has not made progress [30] after the time we specify (in such cases, the solver usually has found the optimal solution and uses the time to prove this is the case). We evaluate both the clustering approach and the impact of the timeout feature in §8.

7 How to augment capacity

Once Raha finds the network is at risk operators may want to augment the network's capacity to eliminate that risk: they can add more cables to LAGs to increase capacity, bring back into service links that are down for maintenance, or dynamically change an optical link's capacity [38, 43]. Raha helps find where to add this capacity.

In Raha, we model two types of capacity augments: (1) by adding capacity to existing LAGs and (2) by adding new LAGs. Operators can choose which to use (or both).

We devise an iterative algorithm where we first run MetaOpt to find if there exists a failure scenario with probability $\geq T$ which would impact the network. If so, then we solve for where to add capacity to most effectively mitigate it's impact, and then repeat this process until no such failures with probability $\geq T$ remain. We find in many cases this approach converges in just 2-3 iterations.

To model augmentations (of either form) we define a variable u_e which specifies how many links we want to introduce on a LAG (whether that LAG exists or not). We then set the capacity of the LAG to $C_e + (u_e)c$ (where c is the capacity of a single link, and C_e is zero if e did not exist in the original topology). We then enforce that the network after it fails should be able to carry at least the same amount of flow for each demand as the healthy network.

It is straight-forward to solve the problem when operators only want to augment existing LAGs [9, 38]: We just plug in the new capacities in the formulation in Equation 2 and modify the objective to minimize $\sum_e u_e$.

Operators usually also prefer this type of augment. This is because to add a new LAG (edge) they often have to consider the physical distance between nodes, whether it is possible to add a physical cable between them because of the geographical terrain, what the impact of the LAG is on their ability to manage the network (*e.g.,* how it impacts the structure of the topology), *etc.* Raha allows operators to encode the LAGs they consider viable and only considers those when it finds augments that add a new LAG to the topology.

But we need a different formulation for when we want to find where to add such new LAGs. This is because the set of paths available to each demand change when we introduce a new LAG and it is hard to account for that change as part of a single optimization. We circumvent this problem through the edge-formulation of the multi-commodity flow problem but modify it to closely tie it to the path form (to find the minimum sufficient augmentation). Due to space limitations we show how to do this in Appendix C.

8 Evaluation

We implement Raha on top of MetaOpt [30].

Summary of results. Raha can consider all possible failures and demands and find if there exists a *probable* failure scenario that can impact the network and the demand that causes that impact in $\leq$ 30 minutes[5]. For a fixed demand, Raha quickly (within 10 minutes) finds whether there exists a *probable* failure that can impact the network.

Raha can always find a *probable* failure scenario that causes higher impact compared to what tools that only consider up to k failures find (for $k \in \{1, 2, 3, 4\}$).

Our use of timeout does not impact the quality of the results Raha produces. Even though we use clustering to approximate demands, Raha finds scenarios that significantly impact the network (in some cases the scenarios it find cause the network to drop 30% of the traffic which the healthy network would have carried).

We find the augmentation algorithm is effective — it augments the network so that there no longer exists a probable failure that can impact it. We fully evaluate other aspects of Raha and our design choices next.

Due to lack of space we will also present an extended evaluation in the appendix (Appendix D).

8.1 Setup, benchmarks, and metrics

Setup. We evaluate Raha on our WAN in Africa: it has $\sim$ 70 nodes and $\sim$ 270 edges which expands to 76 nodes, 334 LAGs and 382 links when we modify it to reflect production constraints (see §9). We also share results on topologies from the topology Zoo with $\geq$ 200 nodes and $\geq$ 486 edges.

We consider link failures in all of our experiments (each LAG has 1 to k links and k may be different for each LAG). The number of failures refers the total number of failed links. We do not have link information for the topologies from the topology zoo and we use LAG failures in those experiments.

Most of our experiments use 2 clusters, $T = 10^{-4}$, 8 primary, and 1 backup paths (we use the k shortest path algorithm and include the time to compute them in our run-times); and a timeout set to 1000 seconds.

MetaOpt solves an optimization under a hood and requires an optimization solver for its backend. We use Gurobi [14] in our experiments.

How to estimate failure probabilities. We estimate the probability a link goes down after it is repaired. We use the renewal reward theory (see Appendix B) to compute these probabilities from the data we collect from production [35] (we know when a link goes down and when it is repaired).

We do not have failure probabilities about the LAGs in the topology Zoo topologies. We instead set these probabilities based on the data from our own production network.

Benchmark. We compare our solution to those which only consider up to k failures can find and consider $k \in \{1, 2, 3, 4\}$. These are the only other works that can solve the same problem and also can model the TE pipeline we use in production.

Metric [6]. We measure the healthy network can carry but that the failed network drops. To protect critical information about our cloud, we normalize this metric by the average capacity across all LAGs. We also present a limited set of results for MLU: for these we show the actual degradation but generate the demand from a gravity model with a scale factor of 100 Gbps so that we do not leak information about our network.

Connected enforced graphs. We show a number of experiments where we only allow failures that do not cause all of a demands' path to go down. We refer to these scenarios as "connected enforced" (CE). We use CE in production.

8.2 Probabilities matter!

Most prior work in this space ignore the probability with which failures happen and only ensure the network is safe when it is subject to $\leq k$ failures. These works argue that most probable failures are those where k is small (≤ 2) [26, 27]. We show this is not the case (Figure 5). As we can see, whether we restrict the demand to fixed values (the way [27] does) or not the worst-case degradation is much higher (at least 2× higher) when we consider other failures that can happen with the same probability but that involve a higher number of link failures. We show how these numbers change under CE constraints (Figure 6).

8.3 On demand variation

One of Raha's strengths compared to prior work is that it can model variations in demand (Table 1). In Figure 5c, we showed results for when we allow *any* demand (from $[0, \infty]$). In this section, we go deeper and analyze how the range of demands an approach considers impacts degradations it finds.

We design the following experiment: we allow Raha to consider variable demands but only allow it to choose demands such that each demand falls in the interval $[0, d_k]$, where d_k is a fixed upper bound on the demand k. We then progressively increase d_k (by a percentage we call "slack")

[5]Africa topology and on a laptop with 32GB RAM and 16 cores (Intel Ultra 7).

[6]We repeat each experiment multiple times and find the results and runtimes stable in most cases. Hence error bars are not visible in many cases.

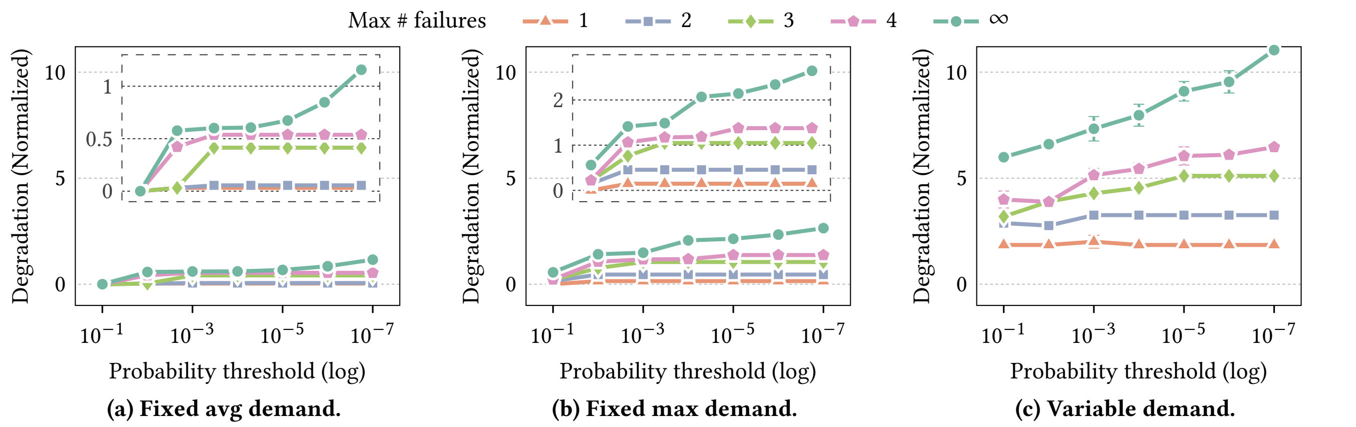


Figure 5: The maximum degradation with (a) fixed average, (b) fixed maximum demands (over a month), and (c) variable demands. We may miss *probable* scenarios that can cause significant degradation in network performance if we limit our analysis to $k \leq 2$ failures: the degradation we find is 10.9×, 4.5×, and 1.9× higher in scenarios (a), (b), and (c) respectively when we consider arbitrary failure (with probability $\geq 10^{-4}$) scenarios vs only up to 2 failures (these numbers grow to 20.8×, 5.8×, and 2.3× when we consider failures with probability $\geq 10^{-7}$).

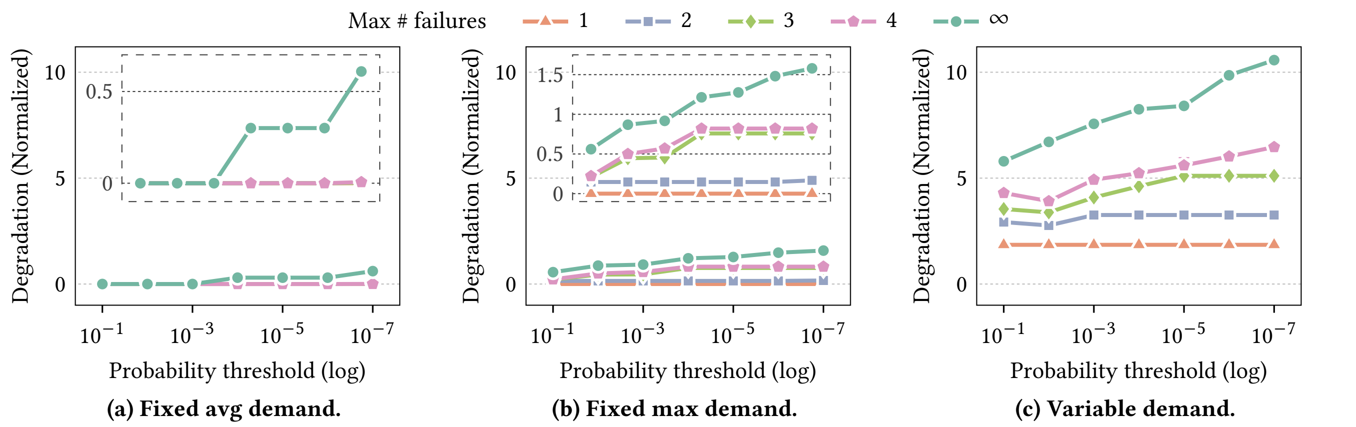


Figure 6: The experiments in Figure 5 but we enforce CE constraints. The worst-case degradation decreases but we still find higher degradations compared to those solutions that limit the number of failures they allow.

to see how this impact's the outcome RAHA produces. We show (Figure 7) RAHA can exploit this increase in range to find scenarios that cause higher degradations.

8.4 On topologies

In addition to experiments on our production WAN, we also conducted experiments on a few topologies from the topology zoo. We show results for one small (B4) and one very large (Cogentco) topology in Appendix D.2 but focus on a medium size one in this section (Figure 8).

We use the Uninett2010 case to demonstrate why we need to cluster the topology when the search space is large: we see the solver fails to make enough progress in the time we allot it when we reduce the probability threshold to 10^{-4}.

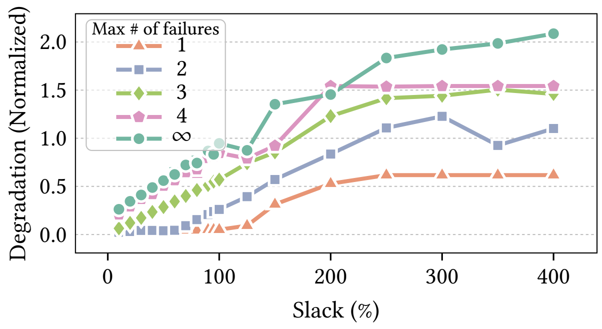
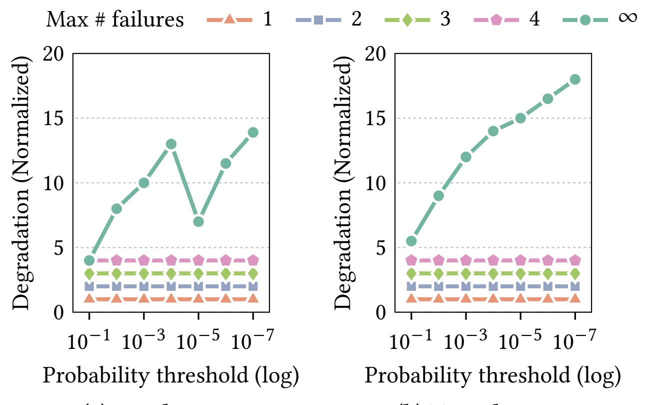


Figure 7: RAHA can find higher and higher degradations if it searches across a larger space of demands.

8.5 On runtime and what influences it

We next evaluate RAHA's ability to scale. For a fixed, given demand, we find RAHA always finishes within 2.68 ± 0.35



(a) No clusters. **(b) Two clusters.**

Figure 8: Uninett2010 with 74 nodes and 202 edges. We use 4 primary and 1 backup path. The performance degradation is normalized by the average LAG capacity (=1000). To ensure a single demand does not create a bottleneck we enforce an upper bound on the demands (half the average LAG capacity).

minutes on our continental topology no matter what the setting. We next show how RAHA scales and what influences its ability to do so under variable demands.

On publicly available topologies. Under variable demands, and arbitrary number of failures, RAHA easily scales to support topologies in the topology zoo: on the $B4$ topology (from TEAVAR) it takes 25.01 ± 0.00 minutes, and on the Uninet2010 topology it takes 28.86 ± 3.95 minutes without any clustering.

On our production network. We next look at the factors that influence RAHA's ability to scale (Figure 10 and Figure 14). In all of the scenarios we consider RAHA finds a solution in ≤ 50 minutes. The run time increases with the number of primary paths: the number of variables we need increases (and so does the time to compute paths). Most practitioners use ≤ 16 paths and we see that RAHA can solve this scenario in less than 25 minutes. The runtime also increases as we decrease the probability threshold we consider (as long as it is greater than zero): the number of feasible solutions increases and Gurobi [14] has to spend more time to prove the solution it finds is optimal. RAHA's runtime improves if we remove the constraints on the probability of failures it considers or those that restrict the maximum number of failures: this allows us to reduce the number of variables and constraints.

On timeouts. Timeouts do not impact the quality of the results RAHA produces no matter what constraints we run it under (as long as we start with a reasonable timeout). We ran RAHA with timeouts between $500 - 4000$ seconds and found it finished within 100 minutes even with the highest

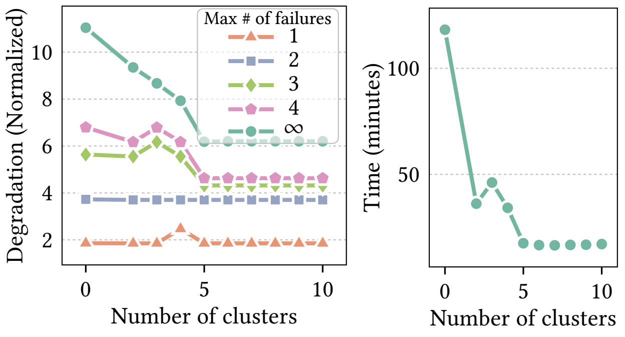


Figure 9: The impact of clustering on performance degradation (left) and RAHA runtime (right).

timeout setting (it finishes in 11.83 minutes[7]) when we use a timeout of 500 seconds. The degradation it found did not change irrespective of the timeout we used (see Figure 16).

On other objectives. We conduct limited experiments where we changed RAHA's objective to find the worst-case MLU degradation. Without clustering, it finished in 15 minutes in all cases and found a degradation of 1.06, 1.32, 1.26 for 0, 10%, and 20% slack respectively. Degradation jumps to 3.12 when we set slack to 40%.

On clustering. Clustering is not always necessary and we do not recommend it when operators want to use RAHA offline. RAHA also does not require clustering to scale in cases where we do not need to enforce a constraint on the probability of the failure scenarios it considers or when we do not consider traffic variations. But in the general case, for large topologies, and when we run RAHA online (*e.g.,* in our use-case where we use it to analyze the network after each failure) we need clustering. Clustering sacrifices optimality but allows RAHA to find potential degradations more quickly.

We run an experiment where we set Gurobi's timeout to t once with no clusters and once with n (we divide t by the number of times we run Gurobi). We set $t = 7000s$. We find using 2 clusters does not impact results when we limit the number of failures (Figure 9). When we analyze arbitrary failure scenarios, RAHA sacrifices optimality for runtime: it finishes 69% faster but sacrifices 15% degradation (Figure 9).

8.6 On capacity augments

So far we discussed one of the two usage modes for RAHA — where it checks whether the network is at risk. Operators can also use it to find how to augment capacity to mitigate that risk. We evaluate this aspect next.

Operators can choose whether to add capacity to existing LAGs or to check what new LAGs they can add to bring the *probable* (we use $T = 10^{-4}$) degradation to zero. We

[7]This time is the sum of the solver time (bounded by the timeout), the time to encode the problem in Gurobi, and the path computation time.

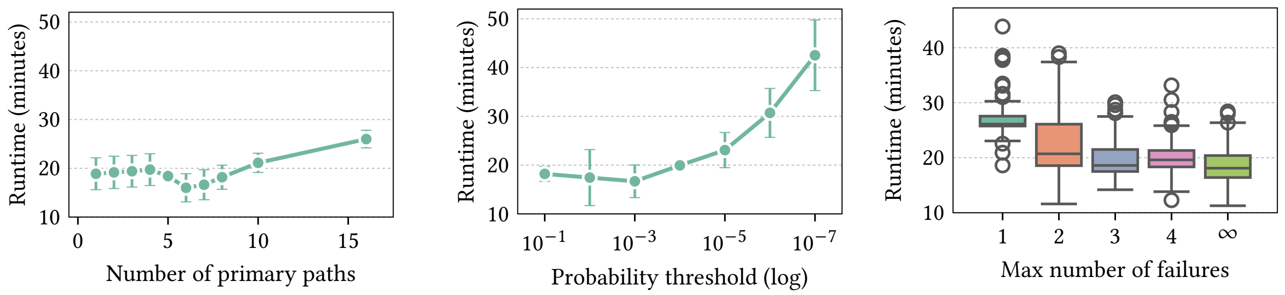


Figure 10: We show how the number of primary paths, the probability threshold, and the maximum number of failures RAHA considers impact it's runtime. We see that in all cases RAHA finds a solution in less than 1 hour. RAHA scale's better when it does not limit the number of failures it considers or puts a threshold on the probability of them (we do not show this point in the middle graph: RAHA finishes in less than 2 minutes in this scenario). This is because we can remove constraints and variables in these cases which reduce the size of the problem.

find for fixed demands (maximum or average) RAHA can find the sufficient capacity augment in 2 steps. We devote this section to evaluate the more interesting case where we augment the capacity so that the network is resilient under arbitrary demands.

We need to assign a failure probability to the new capacity we add (QARC [38] and [9], the only two work that model augments, do not consider probabilities. We analyze that case in Figure 17 and show RAHA easily handles it in 2 steps): we use the average across the failure probability of other links on the same LAG. Our results show RAHA can find capacity augments that fully remove the network's vulnerability to all *probable* degradations in less than 6 steps (Figure 11).

9 Practical considerations

We next describe practical aspects we considered for RAHA.

On continental analysis. We analyze the WAN in each of our continents separately and then the network that connects them. This helps scale and allows us to quickly find a mitigation, isolate, and explain where the network degrades. For example, in the incident that happened in our WAN, we moved our first party services (and their traffic) out of the continent (where our network had degraded) to change the demand and reduce it to what we had capacity for.

On "equivalences". Certain sources and destinations are sometimes "equivalent". The case above is one such example: this means demands that enter or exit the continent can go to one of multiple "gateways" (multi-source or multi-destination). Each of these gateways has a capacity for how much traffic it can help transit.

We add virtual nodes in the network to support this type of analysis. These virtual nodes represent these gateways but have a crucial difference compared to other nodes in the network: they have more path available to them — we allow them access to all paths that their immediate neighbors have access to. We enforce CE constraints on non-virtual nodes.

On paths. A higher number of primary or backup paths does not always reduce the degradation we find. There are multiple reasons for this. First, if operators do not select edge-disjoint paths then RAHA can create larger degradations when it picks links that participate in a larger number of paths. Second, failures can have cascading effects — when all links in a LAG fail, some traffic flows to its backup path which then impacts other paths that share capacity with it (this also means demands that do not have paths that go over the failed LAG may also get impacted by the LAG failure). RAHA can help operators refine their path selection schemes and identify paths that reduce the degradation their network may experience in the face of failures.

On runtime. We included the time to compute path in our runtime evaluations. In practice, we can compute paths offline, once, and speed up RAHA further.

On probabilities. RAHA does not always find scenarios that fail more links if we reduce T. It may pick a different failure scenario that involves fewer link failures but where some of those links are less likely to fail — these are cases where it can create higher degradation if it removes those links but it was not allowed to do so with the higher T.

10 Related work

To the best of our knowledge, RAHA is the first tool that can evaluate how a WANs performance degrades under arbitrary failures and demands. RAHA applies to WANs with any *single-shot* TE algorithm. Prior work has studied network performance in the presence of failures before:

WAN performance verification. Prior work [9, 25, 26, 38, 41] verify WAN performance under failures — they validate the traffic load properties and check whether there exist

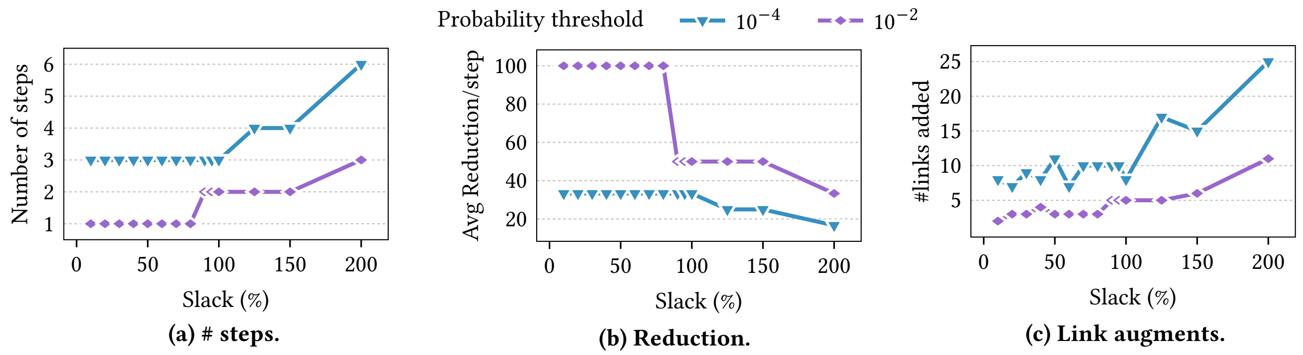


Figure 11: Raha can augment LAGs until *probable* failures can not degrade performance. We show the number of steps Raha needs to achieve this (a); the average reduction in the normalized degradation across these steps where we normalize by the initial degradation at step 0 (b); the total number of links added across all steps (c).

scenarios that can cause network overload. But these works cannot assess how the network performance degrades for arbitrary TE objectives (see §2).

Yu [26] and Jingubang [25] assume routers use distributed routing protocols and do not apply to centralized TE solutions that most cloud providers use [11, 15, 16, 23]. QARC [38] assumes traffic always follows the shortest path available to it, whereas most TE algorithms [1, 15, 32] split traffic over multiple paths. Pita [41] cannot scale to large topologies — it enumerates all failure scenarios.

The closest approach to Raha, [9], which also uses bi-level optimization cannot solve the problem exactly because they cannot handle the non-convexity in the problem [30]. They instead try to find an upper bound on the maximum link utilization which in practice means their approach may cause too many false positives. Other work do not consider performance metrics [17, 37, 40].

WAN capacity planning. Most cloud providers provision their WANs to carry the typical demands they expect to route between data centers and also ensure their WAN is resilient to specific failure scenarios [2, 5, 12, 36, 44]. Raha complements these solutions and provides insights into the demands and failure scenarios that may still impact their final design and cause it's performance to degrade.

Traffic engineering (TE). Operators use TE algorithms with different objectives [1, 4, 6, 15, 16, 18, 19, 24, 27, 32, 43]. Some of these algorithms (*e.g.,* [4, 6, 8, 18, 19, 27, 42, 43]) employ failure recovery mechanisms or optimize for failure resilience to minimize the impact of failures on the demands they route. But, as we saw in the incident that happened in our production network (§2), there is a point where the network no longer has sufficient capacity available for these algorithms to use and these solutions can no longer mitigate the issue. Raha identifies whether such cases are likely to happen and alerts operators to mitigate the issue before they do.

11 Conclusion

We introduced Raha the first tool that can model and analyze the worst-case *degradation* of a WAN's performance (*i.e.,* the difference in performance between the healthy network and the same network with failures) under arbitrary probabilistic failures, variable traffic, and many TE objectives. Our evaluation shows today's solutions often underestimate the severity of performance degradation because they solely focus on the worst-case performance of the network with failures. Raha models a WAN's performance degradation as a bi-level optimization through the lens of existing heuristic performance analyzers like MetaOpt. We carefully extract non-convexities into the outer problem so that Raha can model a wide-variety of TE objectives and retain support from existing off-the-shelf optimization solvers. We showed how Raha allows operators to find vulnerabilities in their networks, quantify their impact, and augment capacity to mitigate them.

References

[1] Firas Abuzaid, Srikanth Kandula, Behnaz Arzani, Ishai Menache, Matei Zaharia, and Peter Bailis. 2021. Contracting Wide-area Network Topologies to Solve Flow Problems Quickly. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, 175–200. https://www.usenix.org/conference/nsdi21/presentation/abuzaid

[2] Satyajeet Singh Ahuja, Varun Gupta, Vinayak Dangui, Soshant Bali, Abishek Gopalan, Hao Zhong, Petr Lapukhov, Yiting Xia, and Ying Zhang. 2021. Capacity-efficient and uncertainty-resilient backbone network planning with hose. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 547–559.

[3] Behnaz Arzani and Pooria Namyar. 2025. MetaOpt and Raha source code. (2025). https://github.com/microsoft/MetaOpt

[4] Yossi Azar, Edith Cohen, Amos Fiat, Haim Kaplan, and Harald Racke. 2003. Optimal oblivious routing in polynomial time. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*. 383–388.

[5] Ajay Kumar Bangla, Alireza Ghaffarkhah, Ben Preskill, Bikash Koley, Christoph Albrecht, Emilie Danna, Joe Jiang, and Xiaoxue Zhao. 2015. Capacity planning for the Google backbone network. In *ISMP 2015 (International Symposium on Mathematical Programming)*.

[6] Jeremy Bogle, Nikhil Bhatia, Manya Ghobadi, Ishai Menache, Nikolaj Bjørner, Asaf Valadarsky, and Michael Schapira. 2019. TEAVAR: striking the right utilization-availability balance in WAN traffic engineering. In *Proceedings of the ACM Special Interest Group on Data Communication*. 29–43.

[7] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex optimization*. Cambridge university press.

[8] Yiyang Chang, Chuan Jiang, Ashish Chandra, Sanjay Rao, and Mohit Tawarmalani. 2020. Lancet: Better network resilience by designing for pruned. *SIGMETRICS Perform. Eval. Rev.* 48, 1 (July 2020), 53–54. https://doi.org/10.1145/3410048.3410079

[9] Yiyang Chang, Sanjay Rao, and Mohit Tawarmalani. 2017. Robust validation of network designs under uncertain demands and failures. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 347–362.

[10] Stephen A Cook. 1971. The complexity of theorem-proving procedures. *Proceedings of the third annual ACM symposium on Theory of computing* (1971), 151–158.

[11] Marek Denis, Yuanjun Yao, Ashley Hatch, Qin Zhang, Chiun Lin Lim, Shuqiang Zhang, Kyle Sugrue, Henry Kwok, Mikel Jimenez Fernandez, Petr Lapukhov, Sandeep Hebbani, Gaya Nagarajan, Omar Baldonado, Lixin Gao, and Ying Zhang. 2023. EBB: Reliable and Evolvable Express Backbone Network in Meta. In *Proceedings of the ACM SIGCOMM 2023 Conference (ACM SIGCOMM '23)*. Association for Computing Machinery, New York, NY, USA, 346–359. https://doi.org/10.1145/3603269.3604860

[12] John P Eason, Xueqi He, Richard Cziva, Max Noormohammadpour, Srivatsan Balasubramanian, Satyajeet Singh Ahuja, and Biao Lu. 2023. Hose-based cross-layer backbone network design with Benders decomposition. In *Proceedings of the ACM SIGCOMM 2023 Conference*. 333–345.

[13] Ramesh Govindan, Ina Minei, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. 2016. Evolve or die: High-availability design principles drawn from googles network infrastructure. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 58–72.

[14] Gurobi Optimization, LLC. 2023. Gurobi Optimizer Reference Manual. (2023). https://www.gurobi.com

[15] Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, and Roger Wattenhofer. 2013. Achieving high utilization with software-driven WAN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. 15–26.

[16] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. 2013. B4: Experience with a globally-deployed software defined WAN. *ACM SIGCOMM Computer Communication Review* 43, 4 (2013), 3–14.

[17] Peter Gjøl Jensen, Dan Kristiansen, Stefan Schmid, Morten Konggaard Schou, Bernhard Clemens Schrenk, and Jiří Srba. 2020. AalWiNes: a fast and quantitative what-if analysis tool for MPLS networks. In *Proceedings of the 16th International Conference on emerging Networking EXperiments and Technologies*. 474–481.

[18] Chuan Jiang, Zixuan Li, Sanjay Rao, and Mohit Tawarmalani. 2022. Flexile: meeting bandwidth objectives almost always. In *Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT '22)*. Association for Computing Machinery, New York, NY, USA, 110–125. https://doi.org/10.1145/3555050.3569119

[19] Chuan Jiang, Sanjay Rao, and Mohit Tawarmalani. 2020. PCF: Provably Resilient Flexible Routing. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*. Association for Computing Machinery, New York, NY, USA, 139–153. https://doi.org/10.1145/3387514.3405858

[20] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. 2011. The internet topology zoo. *IEEE Journal on Selected Areas in Communications* 29, 9 (2011), 1765–1775.

[21] Alexander Krentsel, Rishabh Iyer, Isaac Keslassy, Sylvia Ratnasamy, Anees Shaikh, and Rob Shakir. 2024. The Case for Validating Inputs in Software-Defined WANs. In *Proceedings of the 23rd ACM Workshop on Hot Topics in Networks*. 246–254.

[22] Umesh Krishnaswamy, Rachee Singh, Nikolaj Bjørner, and Himanshu Raj. 2022. Decentralized cloud wide-area network traffic engineering with {BLASTSHIELD}. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 325–338.

[23] Umesh Krishnaswamy, Rachee Singh, Paul Mattes, Paul-Andre C Bissonnette, Nikolaj Bjørner, Zahira Nasrin, Sonal Kothari, Prabhakar Reddy, John Abeln, Srikanth Kandula, Himanshu Raj, Luis Irun-Briz, Jamie Gaudette, and Erica Lan. 2023. OneWAN is better than two: Unifying a split WAN architecture. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. USENIX Association, Boston, MA, 515–529. https://www.usenix.org/conference/nsdi23/presentation/krishnaswamy

[24] Praveen Kumar, Yang Yuan, Chris Yu, Nate Foster, Robert Kleinberg, Petr Lapukhov, Chiun Lin Lim, and Robert Soulé. 2018. Semi-Oblivious Traffic Engineering: The Road Not Taken. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. USENIX Association, Renton, WA, 157–170. https://www.usenix.org/conference/nsdi18/presentation/kumar

[25] Ruihan Li, Fangdan Ye, Yifei Yuan, Ruizhen Yang, Bingchuan Tian, Tianchen Guo, Hao Wu, Xiaobo Zhu, Zhongyu Guan, Qing Ma, Xianlong Zeng, Chenren Xu, Dennis Cai, and Ennan Zhai. 2024. Reasoning about Network Traffic Load Property at Production Scale. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, Santa Clara, CA, 1063–1082. https://www.usenix.org/conference/nsdi24/presentation/li-ruihan

[26] Ruihan Li, Yifei Yuan, Fangdan Ye, Mengqi Liu, Ruizhen Yang, Yang Yu, Tianchen Guo, Qing Ma, Xianlong Zeng, Chenren Xu, et al. 2024. A General and Efficient Approach to Verifying Traffic Load Properties under Arbitrary k Failures. In *Proceedings of the ACM SIGCOMM 2024 Conference*. 228–243.

[27] Hongqiang Harry Liu, Srikanth Kandula, Ratul Mahajan, Ming Zhang, and David Gelernter. 2014. Traffic engineering with forward fault

correction. In *Proceedings of the 2014 ACM Conference on SIGCOMM*. 527–538.
[28] Wade Tylar Milward. 2023. January Microsoft Service Outages. (2023). https://www.crn.com/news/cloud/the-15-biggest-cloud-outages-of-2023?page=2
[29] Pooria Namyar, Behnaz Arzani, Ryan Beckett, Santiago Segarra, Himanshu Raj, and Srikanth Kandula. 2022. Minding the gap between fast heuristics and their optimal counterparts. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks*. 138–144.
[30] Pooria Namyar, Behnaz Arzani, Ryan Beckett, Santiago Segarra, Himanshu Raj, Umesh Krishnaswamy, Ramesh Govindan, and Srikanth Kandula. 2024. Finding adversarial inputs for heuristics using multi-level optimization. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 927–949.
[31] Pooria Namyar, Behnaz Arzani, Daniel Crankshaw, Daniel S Berger, Kevin Hsieh, Srikanth Kandula, and Ramesh Govindan. 2025. Mitigating the Performance Impact of Network Failures in Public Clouds. *NSDI* (2025).
[32] Pooria Namyar, Behnaz Arzani, Srikanth Kandula, Santiago Segarra, Daniel Crankshaw, Umesh Krishnaswamy, Ramesh Govindan, and Himanshu Raj. 2024. Solving {Max-Min} Fair Resource Allocations Quickly on Large Graphs. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. 1937–1958.
[33] Yarin Perry, Felipe Vieira Frujeri, Chaim Hoch, Srikanth Kandula, Ishai Menache, Michael Schapira, and Aviv Tamar. 2023. {DOTE}: Rethinking (Predictive){WAN} Traffic Engineering. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1557–1581.
[34] Microsoft Reactor. 2024. Microsoft Azure Outage in Africa. (2024). https://www.youtube.com/watch?v=ypU_UuwW_w8
[35] Sheldon M Ross. 2014. *Introduction to probability models*. Academic press.
[36] Rachee Singh, Nikolaj Bjorner, Sharon Shoham, Yawei Yin, John Arnold, and Jamie Gaudette. 2021. Cost-effective capacity provisioning in wide area networks with Shoofly. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 534–546. https://doi.org/10.1145/3452296.3472895
[37] Brent Stephens, Alan L Cox, and Scott Rixner. 2013. Plinko: Building provably resilient forwarding tables. In *Proceedings of the Twelfth ACM Workshop on Hot Topics in Networks*. 1–7.
[38] Kausik Subramanian, Anubhavnidhi Abhashkumar, Loris D'Antoni, and Aditya Akella. 2020. Detecting network load violations for distributed control planes. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*. 974–988.
[39] Xin Wu, Daniel Turner, Chao-Chih Chen, David A Maltz, Xiaowei Yang, Lihua Yuan, and Ming Zhang. 2012. NetPilot: Automating datacenter network failure mitigation. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. 419–430.
[40] Baohua Yang, Junda Liu, Scott Shenker, Jun Li, and Kai Zheng. 2014. Keep forwarding: Towards k-link failure resilient routing. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 1617–1625.
[41] Yunmo Zhang, Hong Xu, Chun Jason Xue, and Tei-Wei Kuo. 2022. Probabilistic Analysis of Network Availability. In *2022 IEEE 30th International Conference on Network Protocols (ICNP)*. 1–11. https://doi.org/10.1109/ICNP55882.2022.9940438
[42] Jiaqi Zheng, Hong Xu, Xiaojun Zhu, Guihai Chen, and Yanhui Geng. 2016. We've got you covered: Failure recovery with backup tunnels in traffic engineering. In *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. 1–10. https://doi.org/10.1109/ICNP.2016.7784449
[43] Zhizhen Zhong, Manya Ghobadi, Alaa Khaddaj, Jonathan Leach, Yiting Xia, and Ying Zhang. 2021. ARROW: restoration-aware traffic engineering. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 560–579.
[44] Hang Zhu, Varun Gupta, Satyajeet Singh Ahuja, Yuandong Tian, Ying Zhang, and Xin Jin. 2021. Network planning with deep reinforcement learning. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference (SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 258–271. https://doi.org/10.1145/3452296.3472902

APPENDIX

Appendices are supporting material that has not been peer-reviewed.

A Modeling other TE algorithms

In this section we show how to extend the model we introduced for our production traffic engineering pipeline to other TE pipelines with different objectives.

Single-shot max-min fairness [32]. RAHA can support any traffic engineering solution that we can formulate as a convex optimization problem. The single-shot max-min fair solution from Soroush [32] (namely their Geometric or Equi-depth binner algorithms) are one such approach where it is very straight forward to use RAHA: all we need to do is to use the optimization they introduce for these algorithms and replace the constant capacity with a variable one where we model the network under failure (similar to our approach in §5). We still need to add the constraints in the outer problem that sets the values for these capacity variables.

Modeling MLU (minimizing maximum link utilization). Once again, this is an easy extension of RAHA. To model MLU we need to replace the total demand met objective in both the healthy and the failed network models with a variable $-U$ (because lower MLU is better we need to solve a minimization problem) and enforce in each optimization:

$$UC_e \geq \sum_{p|e\in p} f_p \quad \forall e \in \mathcal{E}.$$

MLU optimizations do not model capacity constraints — we need to rely on path extension capacities to ensure when a LAG goes down the flow on that LAG also goes down (we only enforce capacity constraints on these LAGs) and we need to add a constraint that enforces $C_{kp} \leq u_{kp}$.

Note, the inner problem considers the variable C_{kp} as a constant which means the problem is still convex and we can apply KKT and primal-dual techniques to convert the problem into a single-shot one. We still need to linearize the multiplication. We achieve this through the helper functions MetaOpt provides.

Another important requirement when we model MLU is to use CE constraints: MLU models become infeasible when two nodes become fully disconnected (these formulation require the network carry the full demand but there are no path for the traffic to use).

B Using renewal reward theory

We use the renewal reward theorem [35] to compute the probability with which a link is down (we can use the same process to compute the probability of a link going down).

The renewal reward theorem considers a renewal process in time where time is split into intervals of X_i duration. These X_i are independent and identically distributed. For each interval we define a reward function R_i where these R_i are also independent and identically distributed. The theorem then ensures

$$\frac{E(R)}{E(X)} = \lim_{t\to\infty} \frac{R(t)}{t},$$

where $R(t)$ is the reward we accumulate up to time t.

To compute the probability with which a link goes down after it is repaired, we define the X_i as the time between each repair of the link and R_i as the duration in which the link is down within X_i. If consecutive link failures are independent then so are R_i and X_i.

C Augments with new links

We need to modify our formulation if we want to consider where to add new capacity into our network (where a LAG previously did not exist). This is because each new LAG changes the set of paths that are available between each source-destination pair. We use the edge-formulation of the multi-commodity flow problem to address this.

The primary difference between the edge form and path form (Equation 2) are the flow conservation constraints (what comes into a node goes out):

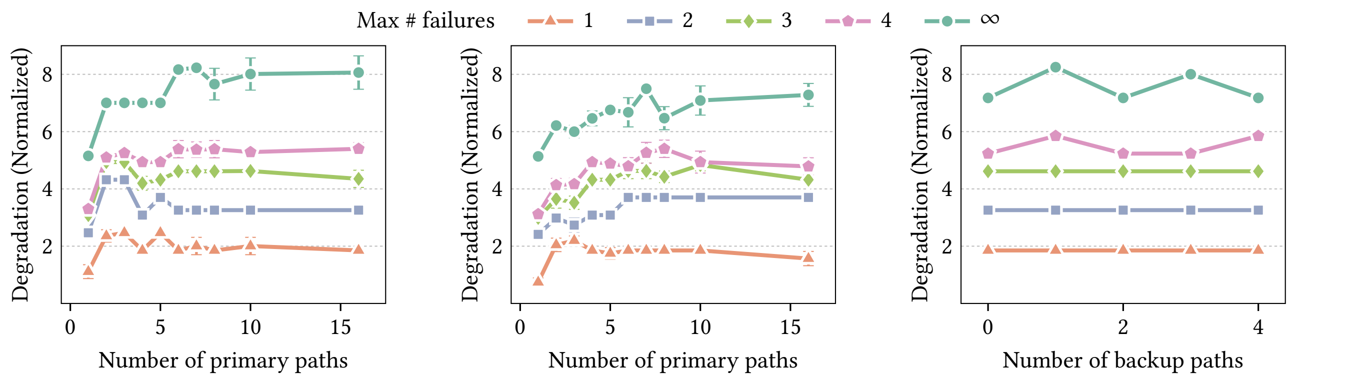
$$\sum_{j|(i,j)\in\mathcal{E}} f_{ijk} + f_k\mathcal{I}(i = t_k) = \sum_{j|(j,i)\in\mathcal{E}} f_{ijk} + f_k\mathcal{I}(i = s_k) \quad \forall i \in N, k \in \mathcal{D}, \tag{6}$$

where s_k and t_k are the source and destination for demand k respectively and $f_{(i,j,k)}$ is demand k's flow on LAG (i, j).

In the edge form we define the flow variables per edge (LAG) instead of per path. The only restriction on where traffic can flow comes from the flow conservation constraints and capacity constraints: this is why the edge form can route more flow — it has all possible paths between each source-destination available to it. This means the solution we find with the edge form is an upper bound on what our network would be able to route. We apply techniques to tighten this bound so that we only add capacity if/when it is needed:

First, for each demand, k, we only define the values $f_{(i,j,k)}$ on those paths that existed before the failure happened *and* for new LAGs which didn't exist in the original topology.

Second, we weigh capacity augments to prefer those which will likely be part of a demand's path: for example, if the network uses the k-shortest path, we prioritize LAGs based on their minimum distance to the source-destination pairs whose demand was impacted by the failures MetaOpt found.



(a) Degradation vs # paths. (b) Degradation vs # paths (CE). (c) Degradation vs # backup paths.

Figure 12: (a) and (b) show how the number of primary paths available changes the degradation we can find; (c) shows the same as we vary the number of backup paths. Surprisingly, in some examples, we increase the degradation if we increase the number of available primary paths: this is because in our experiments we compute paths through the k shortest paths algorithm and the paths we find often share LAGs — the algorithm exploits the increase in shared failure modes to increase the degradation.

Finally, we progressively increase the demand we consider in the augment step if we find the edge form can match the performance of the healthy network despite the link failures while Raha indicates the path-form cannot (through this step we match the demand to the degree which the edge-form makes additional paths available).

D Extended evaluation

We continue our evaluation in this section due to lack of space in the main body of the paper.

D.1 Paths and degradation.

We next investigate how path selection influences the degradation in performance the network can experience (Figure 12). Perhaps surprisingly, we see the degradation *does not* always decrease as we increase the number of paths. This is because of our path selection algorithm which picks the top k shortest paths: many of these paths may share common edges (LAGs) and as we increase the number of paths the possibility of their fate-sharing increases — we can bring down more paths through the failure of all links in a single LAG.

We investigate this further and repeat the experiment in Figure 12b but with paths which we select differently (we apply weights to LAGs to change which paths we select). The results show there is a point after which the degradation decreases as we add more path (Figure 13).

We also show results where we look at the fixed, maximum demand instead. Here the number of path matters less, since Raha cannot manipulate the demands to take advantage of "shared" failure modes.

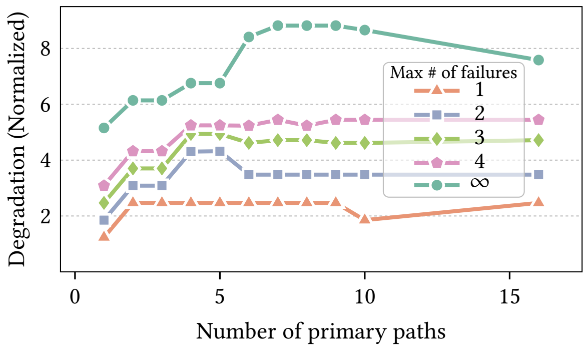


Figure 13: Impact of our path selection scheme on our earlier results. We see there is a point after which more paths helps reduce the degradation Raha can find.

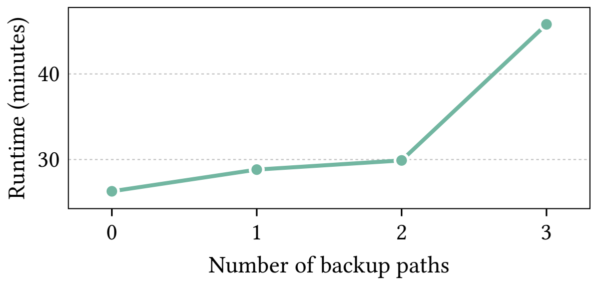
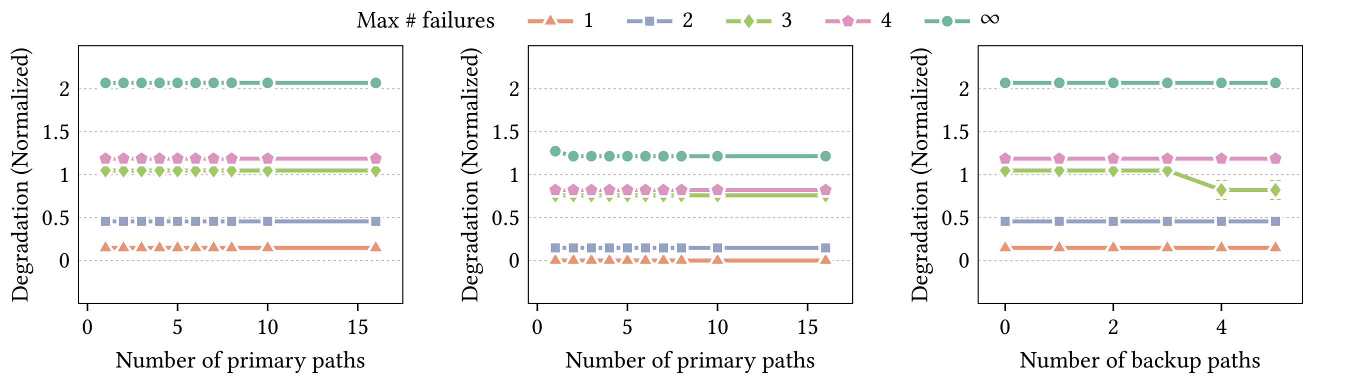


Figure 14: We evaluate the ability to scale as we increase the number of backup path (remember we include path computation in this runtime). Raha finishes in under 50 minutes in all cases. If we exclude the path computation time the computation finishes even sooner: 33.33 ± 0.14 when we have 4 backup paths.

D.2 More on topology zoo

We show more results on the B4 topology from the topology zoo. We constrain the demands to be below half the average



(a) Degradation vs # paths. **(b) Degradation vs # paths (CE).** **(c) Degradation vs # backup paths.**

Figure 15: We repeat the experiments in Figure 12 but we fix the demands to the maximum traffic we observe between each pair in a month-long duration. We see here the degradation does not depend on the number of paths because RAHA cannot manipulate the demand.

T	# backup	max failures	Degradation
10^{-1}	1	$\in \{1, 2, 4, \infty\}$	1
10^{-1}	2	$\in \{1, 2, 4, \infty\}$	1
10^{-2}	$\in \{1, 2\}$	1	1
10^{-2}	$\in \{1, 2\}$	2	2
10^{-2}	$\in \{1, 2\}$	$\in \{4, \infty\}$	3
10^{-4}	$\in \{1, 2\}$	1	1
10^{-4}	$\in \{1, 2\}$	2	2
10^{-4}	$\in \{1, 2\}$	$\in \{4, \infty\}$	4

Table 3: Results on the B4 topology. All the experiments hit the 25-minute timeout we used and finished within 25 minutes. The performance degradation is normalized by average LAG capacity (=5000).

LAG capacity to ensure a single demand does not create a bottleneck. Because we do not have probability estimates nor link information for these topologies we assumed each LAG only consists of a single link and assigned the link failure probabilities randomly and based on values from our production network. We see that without clustering, RAHA always hits the 25-minute timeout we used in this case (Table 3).

We also run experiments on a large topology (CogentCo). This topology has 197 nodes and 486 edges. We use 4 main paths and 1 backup. To scale, we use 8 clusters. Our results Table 4 show RAHA can find a higher degradation compared to other tools, which focus on a limited number of failures.

D.3 More on scale!

Impact of the number of backup paths. We show the impact of the number of backup path on our runtime (*remember we include the path computation as part of our runtime*) in Figure 14. The runtime increases as we increase the number of backup paths but we find the big reason for this is the path

T	# backup	max failures	Degradation
$\in \{10^{-1}, 10^{-2}\}$	1	1	1
$\in \{10^{-1}, 10^{-2}\}$	1	2	2
$\in \{10^{-1}, 10^{-2}\}$	1	4	4
10^{-1}	1	∞	6
10^{-2}	1	∞	10.5

Table 4: Results on the Cogentco topology with 197 nodes and 486 edges. We use 4 primary paths, 1 backup path, and 8 clusters. We normalize the performance degradation by average LAG capacity (=1000).

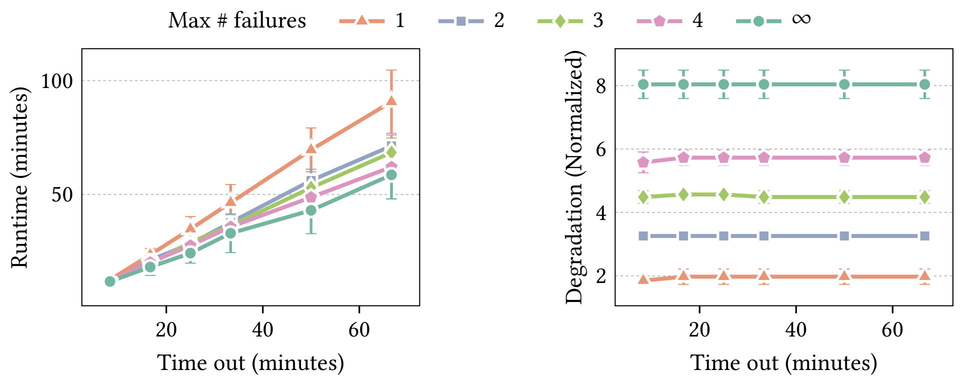
computation itself: RAHA finishes in 33.3 ± 0.14 minutes with 4 backup paths when we input the paths instead of having it compute them.

Impact of timeout. We discussed the impact of timeouts on scale in § 8. We show the detailed results here for the interested reader. We see the timeout does not influence the quality of the degradation we find (Figure 16b).

The type of path we use impact's scale. In our experiments we used the k shortest path where we use the number of paths as the weight of each LAG. For variable demands our runtime remains within 25 minutes on our continental topology (22.39 ± 8.39 minutes).

D.4 More on capacity augments

Capacity augments on existing LAGs. We showed RAHA goes beyond existing works when it augments LAG capacities and is able to consider the probability of these new capacities failing as part of its analysis. Here we also evaluate it in the scenario which prior work also consider: one where it augments the capacity of existing LAGs and assumes this new capacity cannot fail.



(a) Impact of timeout on runtime. (b) Impact of timeout on the solution.

Figure 16: Timeouts do not impact degradation we find but do impact the runtime of Raha.

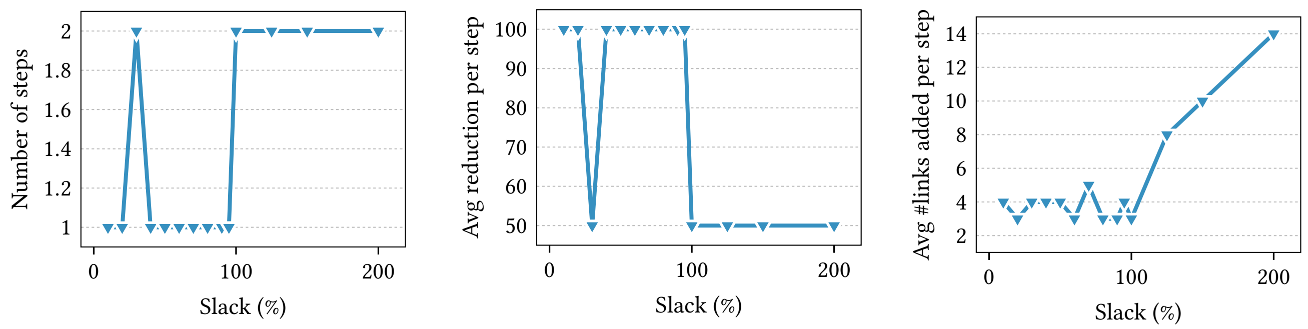


Figure 17: Raha can augment LAGs until *probable* failures can no longer degrade performance. We show the number of iterations Raha needs to achieve this (a); the average reduction in the normalized degradation across these steps (b); and the total number of links it adds (c). Unlike in Figure 11 where we allowed this augmented capacity to also fail, here we assume they cannot. Here the failure probability threshold we enforce is 10^{-4}.

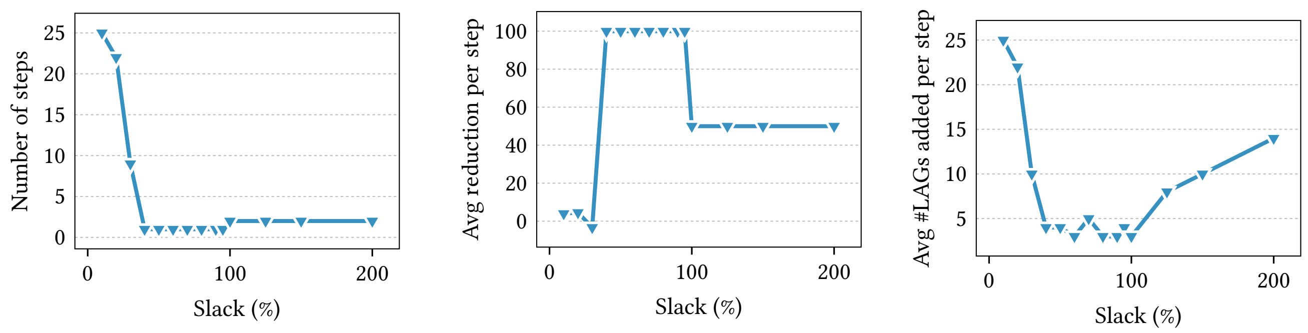


Figure 18: Raha can add new LAGs (edges) to the topology until *probable* failures can no longer degrade performance. We evaluate this behavior here where we assign the probability of these failures to zero.

Adding new capacity. Operators may want to consider where to add new edges. To do so, they first need to identify where it is *possible* to add such edges (LAGs): does the terrain allow it? What is the cost? What is the impact on their ability to manage the network (*e.g.*, does it break the symmetry they had baked into the topology design?). Once they identify the set of edges they consider feasible to add they can use Raha to find which (smallest) subset would reduce the degradation to zero (Figure 18).