Fast Computation Flow Restoration with Path-based Two-stage Traffic Engineering

Xiaotian Li
New York University
Brooklyn, New York, USA
xl3399@nyu.edu

Yong Liu
New York University
Brooklyn, New York, USA
yongliu@nyu.edu

Abstract

The emerging edge networks are cloud-native. Flows with computation needs are processed in-flight by compute nodes inside the network. Routing with In-Network Processing (RINP) not only has to maintain network-wide load balance on communication and computation elements, but also has to quickly restore flows upon various types of failures. In this paper, we propose a novel path-based two-stage traffic engineering scheme to trade-off between routing model complexity, network performance in the normal stage, and restoration efficiency upon failures. For the normal stage, our model jointly optimizes computation demand allocation and traffic flow routing. We further speed-up RINP calculation by controlling the path budget and decoupling computation allocation and traffic routing. For the restoration stage, we develop a fast restoration scheme that only re-routes the flows traversing the failed elements to achieve close-to-optimal network delay performance while minimizing the fraction of unrestored flows. Evaluation results on real network instances demonstrate that in the normal stage, our scheme achieves near-optimal performance with up to **50-100x** speedup compared to link-based routing models. In the restoration stage, our scheme can restore most of the affected traffic with up to **10x** speedup compared to globally rerouting all the flows.

CCS Concepts: • **Networks** → **Traffic engineering algorithms**; **Network resources allocation**; **Network control algorithms**.

Keywords: Edge Computing, In-network Processing, Routing, Traffic Engineering, Restoration

ACM Reference Format:
Xiaotian Li and Yong Liu. 2023. Fast Computation Flow Restoration with Path-based Two-stage Traffic Engineering. In *The Eighth ACM/IEEE Symposium on Edge Computing (SEC '23), December 6–9, 2023, Wilmington, DE, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3583740.3626615

1 Introduction

The emerging edge networks are *cloud-native*, with computational capabilities and services embedded at different levels of the device-edge-cloud continuum. Leveraging on the fast-developing virtualization technologies, such as container, virtual machine, and Network Function Virtualization [1], generic compute servers at the network edge can be flexibly configured to conduct various *in-network processing* [2], ranging from software-based 5G core network functions, virtual middle-boxes for intrusion detection and prevention, to application performance enhancement and mobile device offloading. Routing in such computing-minded networks is no longer just establishing an end-to-end pipe to transmit user data. It also needs to find sufficient computation resources along the route for the required in-flight processing of each data flow. For example, AR/VR are expected to be the killer applications for future edge networks. To facilitate such applications, lots of computation and data-intensive tasks, e.g., object detection [3], 3D video coding, super-resolution [4] and rendering [5], and user behavior data analytics, have to be offloaded to servers embedded at different levels of the device-edge-cloud continuum [6]. How to route AR/VR application flows in edge networks to satisfy their communication and computation demands, while maintaining good balance on edge links and servers is a central problem for such applications.

In general, **Routing with In-Network Processing (RINP)** has to simultaneously satisfy the communication and computation demands generated by users using the communication and computation resources available on links, routers, and compute nodes. A small such routing example is illustrated in Fig. 1. *An*

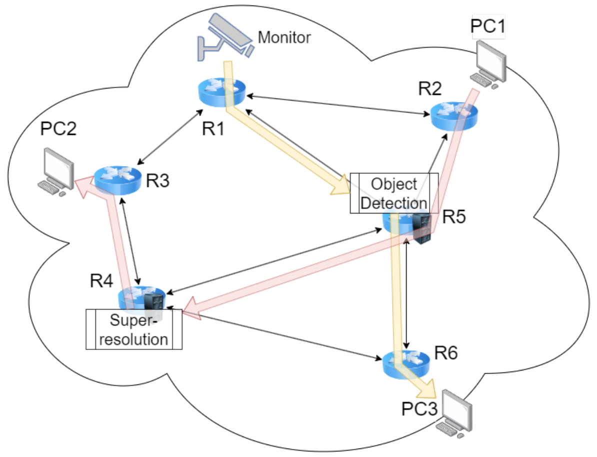


Figure 1. An example of RINP in an edge network. R4 and R5 are routers with compute nodes attached. The live video stream sent from PC1 to PC2 needs to be transcoded, and the transcoding (super-resolution) VM is spawned on R4. The surveillance camera's video feeds are processed by an object detection model before being sent to the human monitor on PC3. The Object Detection VM is placed on R5.

important design objective of RINP is to maintain network-wide load balance on all communication and computation elements. Towards this goal, different variations of the RINP problem have been studied recently [7–9]. It was shown that a feasible RINP route may have to contain loops, and the unsplittable RINP problem in general is NP-Hard. In [8, 9], the optimal splittable RINP was studied using *link-based* routing formulation, where the traffic of a flow can be allocated to any link and the computation demand of a flow can be processed by any reachable compute node. On one hand, their models can reach the theoretical optimum for different design objectives under the given communication and computation resource constraints. On the other hand, the solving time for the link-based routing models may increase dramatically as the network size and number of demands grow, and it is cumbersome to implement and update link-based routing configuration. *Another equally important design objective of RINP is to be resilient against the possible failures on communication links, routers, and compute nodes.* The resilience of RINP at network edge is critical, given the volatile nature of wireless access links, the limited degree of user traffic multiplexing, and the reduced over-provisioning that is economically viable on distributed edge compute nodes. To minimize the interruptions to user applications, after a failure occurs, RINP has to quickly restore the affected application flows by finding sufficient communication and computation resources for them in the surviving network.

To address the scalability and resilience issues of link-based routing models, a lot of modern centralized traffic engineer (TE) systems use *path-based* routing models to reduce the operational overheads [7, 10–13]. A set of candidate paths are pre-configured for each flow. The traffic splitting ratios among candidate paths can be adapted to network and traffic dynamics in real time. Compared to link-based models, path-based models have fewer routing variables, and thus take much shorter time to solve. Deploying new routing solutions is also easier, since paths are predetermined and only the traffic splitting ratios at ingress nodes need to be adjusted, whereas link-based systems may require re-configuring the forwarding rules on many switches. The advantages of the path-based routing model over the link-based routing model make it a better fit for the RINP problem, especially for fast restoration after failures.

For path-based TE, the quality of the pre-configured candidate paths plays a key role in the trade-off between model solving speed, routing performance in normal operation, and restoration efficiency after failures. Both the number of candidate paths and the diversity among them need to be carefully engineered. On one hand, larger candidate path sets lead to a larger routing design space to achieve better network performance in the normal stage. Additionally, if a flow has more pre-configured candidate paths, it is more likely that some of the paths will survive the failures, and can be used to restore the flow. On the other hand, the routing optimization complexity and the implementation overhead increase significantly with the path budget of each flow. The ideal solution is to construct a minimal number of paths for each flow that provide sufficient routing diversity in both normal operation and restoration so that a desirable trade-off can be achieved.

In this paper, we propose a novel path-based two-stage Traffic Engineering (TE) scheme to quickly restore flows with computation needs after failures on communication links, routers and compute nodes. In the normal stage, we use the **segment routing** idea to formulate a path-based routing model (called PRINP) to jointly optimize the

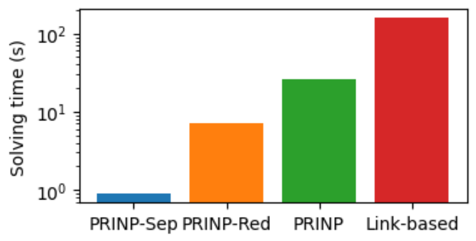


Figure 2. Comparison of the solving time on a network with 39 nodes, 172 edges, and 1,471 directed demands. Link-based refers to the link-based RINP model proposed in [9].

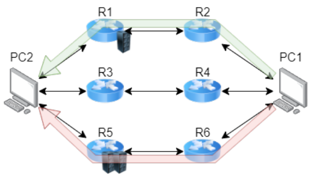
computation demand allocation and traffic flow routing to minimize the network delay while maintaining bounded utilization on all compute nodes. Path-based routing calculation is significantly faster than link-based routing calculation. To speed-up further, we propose two variants, namely the PRINP-Red model, which reduces the number of candidate paths on each routing segment, and the PRINP-Sep model, which first solves for a target computation demand allocation using a simplified network traffic model, then optimizes traffic routing to realizes the target computation demand allocation. A preview of the solving time of each model is presented in Fig. 2. After unexpected failures on communication and/or computation elements, we maximally restore the affected flows using the survived resources. To minimize the disruption to users and achieve fast restoration, only flows traversing the failed elements will be re-routed, and the computation demand allocation and traffic routing of the other flows remain unchanged. Our partial re-routing scheme is much faster than global re-routing which recalculates routing for all the flows. We develop a partial restoration routing model to achieve close-to-optimal network delay performance while minimizing the fraction of flows that cannot be fully restored.

Contributions. Our contributions are as follows,

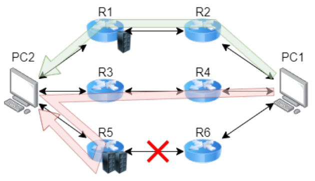
1. We develop a novel path-based two-stage TE scheme for the RINP problem to trade-off between routing model complexity, network performance in the normal stage, and resilience against unexpected failures on communication and computation elements.
2. In the normal stage, our PRINP model jointly optimizes the computation demand allocation and traffic flow routing to minimize the network delay, while maintaining bounded utilization on all compute nodes. By controlling path budget and decoupling computation allocation and traffic routing, our scheme achieves near-optimal performance, with about 50-100x speedup compared to the link-based routing models.
3. In the restoration stage, we develop a fast restoration scheme that reroutes only the flows traversing the failed elements, and can restore a great part of them in a very short time. Through an extensive evaluation, we demonstrate that our restoration scheme can achieve close-to-optimal network delay performance while minimizing the fraction of unrestored flows.
4. We employ different path selection strategies for different types of flows to achieve the desirable trade-off between solving speed, performance, and resilience of RINP.

Table 1. The comparison between link-based and path-based routing models for standard MCF problem

Model Type	Number of variables	Number of constraints
Link-based	$D \times E$	$E + D \times V$
Path-based	$D \times \hat{k}$	$E + D$



(a) Traffic/Computation Splitting in Normal Scenario



(b) Flow Restoration after Failure on Link (R5, R6)

Figure 3. An illustration of the RINP problem. R1 and R5 are routers with compute nodes attached. There is 1 unit of computation resource on R1 and 2 units on R5. The link capacity is 1 unit for all links. PC1 wants to send 1 unit of traffic to PC2, with 1 unit of computation resource required.

The rest of the paper is organized as follows. In Section 2, the background and related work are discussed. We introduce the path-based two-stage TE scheme in Section 3, followed by evaluations using real network instances in Section 4. The paper is concluded in Section 5.

2 Background and Related Work

2.1 Optimal Routing

Routing is an essential problem in communication networks and is also well-studied. The traditional routing problem seeks to optimize some network performance objective function, e.g. Maximum Link Utilization (MLU), network-wide delay, total routed traffic flows, etc., under the link/router bandwidth constraints. The optimal routing can be obtained by solving a Multi-Commodity Flow (MCF) problem [14].

There are mainly two types of linear programming (LP) models to solve the MCF problem, one is the path-based model, and the other is the link-based model[1]. In the path-based model, there is a set of candidate paths for each demand. The demand must be realized on paths in the set, thus limiting the search space. In the link-based model, the traffic of a demand can be allocated on any link, as long as the flow conservation holds. The link-based model can implicitly explore all the possible paths, thus promising optimality [14]. The optimality guarantee in the path-based model is weaker, but with a limited size of the path set, the number of variables and constraints can be greatly reduced, leading to a shorter solving time. A comparison of the two types of models is shown in Table 1, where the number of demands is D, the number of nodes is V, the number of edges is E, and the average number of paths for each demand is $\hat{k}$.

2.2 Routing with In-Network Processing

RINP is different from the traditional routing. In addition to the potential loops in the routes [8] and the complicated routing models [9], the objective of load balance on compute nodes and on links may conflict with each other. A toy RINP example is illustrated in Fig. 3a. On one hand, if we want to achieve load balance on links, we should distribute the traffic evenly between the green and the red sub-flows. On the other hand, if we pay more attention to load balancing on compute nodes, the split ratio should be 1:2. To address this issue, we limit the maximum utilization of the compute nodes and optimize for the load balance on links. Limiting utilization on compute nodes can help bound the queuing delay that a flow experiences in the computation job queue.

There has been a number of studies on the RINP problem, with different assumptions on the ability to split flows and the homogeneity of processing resources (e.g. [7–9, 15–18] and reference therein). The authors in [7] assume each flow has to go through a service chain of middle-boxes in a fixed order, and they formulate the problem as a mixed integer programming (MIP) using graph layering. Their model is path-based, but they mainly focus on budget and latency requirements as well as online routing, without resilience considerations. In [8], a router can split a flow arbitrarily into fractions to be forwarded to different next-hops. Instead of traversing a predefined set of middle-boxes, they consider the allocation of processing resources along the routes of each flow, so that the required network functions are set up, and establish a link-based LP model. In [9], the authors study combinations of different assumptions. For the same scenario as in [8], [9] also uses a link-based LP, and the complexity is reduced by taking advantage of segment routing and destination-based aggregation. The RINP problem has also been studied as service chaining, e.g. [15, 16], and virtual network embedding, e.g., [17, 18].

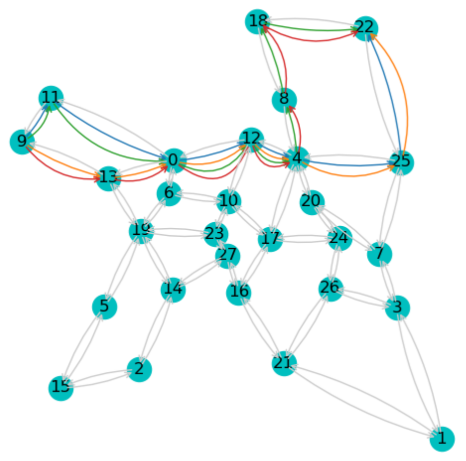
The above-mentioned schemes have a large computation overhead, and they do not consider failures or only consider link failures. In contrast, not only do we formulate the problem in a path-based LP model, but also we focus on the resilience upon failures, including link and router failures, as well as computation resource failures. Fig. 3b is an example of computation flow restoration after failure. A restoration route with loop has to be employed to utilize computation resources on R5. We want to make sure the restoration can be done in a short time to minimize the disruption to user applications.

2.3 Resilient Routing

Failures in the networks are rare events, but may cause severe performance degradation. To protect the networks from the long-lasting impacts of failures due to slow reaction, many schemes, e.g. [19–21], were proposed to provide fault tolerance and/or reduce the disruptions to user applications.

The authors of [19] built an LP model so that as long as the number of failures is within the configurable thresholds, the admitted traffic volume will not be impacted after ingress routers naively re-split traffic to surviving tunnels. Their routing decision guarantees that there is still adequate redundant bandwidth to tolerate failures within the predetermined thresholds. In [21], the authors introduced more flexible network response strategies, called logical

[1]In [14], the path-based model is called the link-path formulation, while the link-based model is called the node-link formulation.


(a) KSP paths from node 9 to node 22

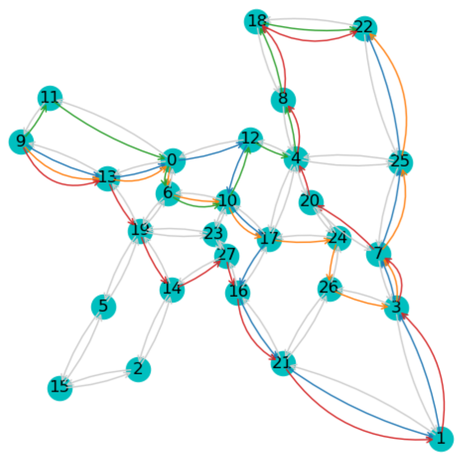

(b) SMORE paths from node 9 to node 22

Figure 4. An example of the difference between KSP paths and SMORE paths. Consider the paths from node 9 to node 22. All KSP paths go through the link (10, 12), so the link is of high risk. If it fails, the traffic can not be restored. There is no such high-risk link for SMORE paths because they are spread out.

sequences, to do better than naively re-scaling traffic after failures. Their approach improves resource utilization efficiency, allowing more traffic admitted while not violating the fault-tolerance guarantee. Instead of providing a hard guarantee, the authors in [20] took additional inputs of failure probabilities, and optimize the flow allocation to minimize the *Conditional Value at Risk*, a concept from financial risk management.

The above-mentioned schemes provide protection from failures at the cost of less admitted traffic. They incur heavy computation overhead and are too slow for fast restoration when the network topology is large. Therefore, the recent studies, e.g. [12, 13], focused on reducing the time to calculate the restoration routes, and quickly deploying the newly-computed solution to routers and switches by the means of Software-Defined Networking (SDN) [10]. Authors of [12] observed that in a nationwide backbone network, a large portion of traffic is within nearby nodes. By clustering, they proposed a four-step hierarchical approach with much faster computation speed and near-optimal performances. The authors of [13] utilized the massive historical data and machine learning techniques to further speed-up. Both schemes can admit more traffic than the schemes in [20] if no failure or after reconfiguration upon failure. The drawback of their fast re-computation approach is that they do not consider whether there will be too many routing reconfigurations, which is also bad for network operation. Following this trend, we target at speeding up the restoration calculation, to shorten the control loop of TE and accelerate the reconfiguration when a failure happens. Upon a failure, we only reroute the affected flows to avoid unnecessary churns.

2.4 Path Selection

In recent centralized TE systems, the phases of path selection and traffic splitting among paths are usually decoupled. To help reduce the real-time optimization overhead, the desired path set should be limited in size, but it should also contain all the "good" candidate paths needed for the optimal solution in any possible scenario. Intuitively, "good" paths should both incur low routing cost and provide route diversity in different scenarios. The most popular path selection strategy is K-Shortest-Paths (KSP) [10, 22]. By taking the top-k shortest loop-less paths for each node pair, KSP ensures the paths selected are cheap, while the diversity is controlled by the parameter k, referred to as the path budget. However, k is usually small for computation and operation efficiency, and in some large topologies, the paths selected tend to share a set of "low-cost" links, leading to overloading on those links and pure diversity if those links fail.

In a recent work [11], the authors found that traditional oblivious routing could be useful in selecting paths. Oblivious routing is to compute a probability distribution on paths, and the traffic is forwarded according to the distribution regardless of the actual volume. They use the iterative approach by Räcke [23] to construct a probability distribution over a set of low-stretch routing trees, and then extract the path set, called SMORE, from it. Compared to the KSP path set, the SMORE path set is more diverse and low-risk, at the cost of slightly longer path lengths. An illustration is shown in Fig. 4.

While the previous path selection methods provide ways to construct path sets, in this work we focus on how to fit them into our RINP problem, and how to design a TE scheme with good resilience against failures of different types.

3 2-Stage Traffic Engineering Scheme

The network is modeled as an undirected mesh-connected graph $G = (V, E)$, where V denotes the set of nodes and E denotes the set of undirected edges. Each edge, $e = (u, v)$, represents two directed IP links between the endpoints, and the two IP links share a total capacity of C_e. There is a subset of nodes called compute nodes, Z, in which each node z has some computation capacity N_z. With virtualization technologies, each computing node can implement multiple software-based network functions and virtual middle-boxes for various in-network processing.

The demands in this network are aggregated directed traffic flows and can be arbitrarily split among multiple routing paths.[2] They are categorized into two sets, D_0 and D_1, depending on whether they require computation resources[3] or not. The source and destination nodes of demand d are $s(d)$ and $t(d)$, respectively. For each demand $d_0 \in D_0$ that does not require processing, the corresponding traffic volume is h_{d_0}. For each demand $d_1 \in D_1$ that requires processing, the amount of computation resources needed is W_{d_1}, and the traffic volume before and after computation may vary and are denoted as $h^1_{d_1}$ and $h^2_{d_1}$, respectively. Note that there is no direct relationship between the computation resource requirement and the traffic volume.

The candidate paths for each demand are computed in advance. For a regular demand $d_0 \in D_0$, we choose the SMORE path set [11] for its better diversity and resilience. However, for a demand $d_1 \in D_1$ that requires processing, following the segment routing framework for RINP [9], an end-to-end path consists of two segments: from the source to some compute node, and from the compute node to the destination. Since the compute nodes act as the intermediate nodes of Valiant Load-Balancing (VLB) routing [24], they have already provided some diversity to the routing problem. We use KSP paths for both segments, since they are normally shorter than SMORE paths, and the end-to-end diversity can be enhanced by employing multiple compute nodes in the middle. More details about path setting will be discussed in section 4.

We also made the following assumptions. First, the traffic matrices (TM) are known in advance and do not change until failures happen. One of our design goals is to accelerate the solving time and shorten the TE control loop. While perfect knowledge of the future TM dynamics is impossible, we can expect that the TM is relatively stable and can be predicted with good accuracy in a short period. Second, the computation resources are homogeneous. With the fast-developing NFV techniques, different computation tasks could be carried out as network functions, implemented on virtual machines that can be elastically provisioned on commercial servers. The amount of computation resources needed could be measured as CPU, GPU, and storage requirements of the VMs, quantified using some standard units. The notations are summarized in Table 2.

3.1 Routing in Normal Stage

3.1.1 Path-based RINP Model. In the normal stage without failure, our objective is to achieve good network and computation performance at the same time for the RINP problem. Example objective functions include minimizing the total network/processing delay, minimizing the maximum link/server utilization, maximizing the concurrent flow, etc.

For illustration purposes, we choose to minimize the total network delay. The delay on a link is estimated using the M/M/1 queue formula, and the delay of the whole network is the average of all the link delays weighted by the link traffic volume. Regarding computation performance, we introduce a constraint to bound the utilization of the compute nodes. Empirically, computation delay is determined by more factors than the computation server utilization and is application-specific. By keeping the server utilization within a threshold, we expect the processing delay to be kept low. Our RINP models can be extended to explicitly minimize computation delays when the application-specific computation delay models are available. Note that our model can also be easily customized for other TE objectives.

[2]One aggregated flow consists of multiple individual user application flows share the same ingress and egress points of the network, and optionally have the same type of processing needs.
[3]Requiring computation resources is equivalent to requiring processing.

Table 2. Notations

Symbol	Description
V	Set of nodes
$Z \subset V$	Set of nodes with computation resources
N_z	Computation capacity on node $z \in Z$
ρ	Upper bound of computation resources utilization
E	Set of undirected edges
C_e	Capacity of edge e
D_0	Set of demands that do not need processing
D_1	Set of demands that need processing
$s(d), t(d)$	Source and destination nodes of a demand
$P_i(s,t)$	Pre-computed paths for flow $s \longrightarrow t$ in demand set D_i
δ_{ep}	Indicator constant, whether path p traverses edge e
h_{d_0}	Traffic volume of flow $d_0 \in D_0$
$h^1_{d_1}$	Traffic volume in the first segment of $d_1 \in D_1$
$h^2_{d_1}$	Traffic volume in the second segment of $d_1 \in D_1$
W_{d_1}	Computation resources required by demand $d_1 \in D_1$
f_e	Total traffic volume on edge e
Variables	
α_{d_0p}	Routing fraction on path $p \in P_0(s(d_0), t(d_0))$ of d_0
$\beta^z_{d_1pq}$	Routing fraction on path $p \in P_1(s(d_1), z)$ in segment 1 and path $q \in P_1(z, t(d_1))$ in segment 2 of d_1 through z
$r^z_{d_1}$	The computation resources allocated on node $z \in Z$ for demand $d_1 \in D_1$
x_{d_0p}	The traffic volume on path $p \in P_0(s(d_0), t(d_0))$ to realize the demand $d_0 \in D_0$
$x^{zi}_{d_1p}$	The traffic volume on path p in segment i, $(i = 1, 2)$ through $z \in Z$ to realize the demand $d_1 \in D_1$

When demands are infinitely splittable, the *n-stop* RINP path can be decomposed into *n 1-stop* RINP paths [9]. With this property, each demand that needs processing can be routed using two segments, corresponding to the routes before and after the processing. Therefore, we can formulate a path-based routing model for RINP problem as follows.

$$\underset{\alpha,\beta\geq 0}{\text{minimize}} \quad \mathcal{F} = \sum_e \frac{f_e}{C_e - f_e} \tag{1a}$$

$$\text{subject to} \quad \sum_p \alpha_{d_0p} = 1, \quad \forall d_0 \in D_0; \tag{1b}$$

$$\sum_{z,p,q} \beta^z_{d_1pq} = 1, \quad \forall d_1 \in D_1; \tag{1c}$$

$$\sum_{d_1} (\sum_{p,q} \beta^z_{d_1pq}) W_{d_1} \leq \rho N_z, \quad \forall z \in Z; \tag{1d}$$

$$f_e = \sum_{d_0,p} \delta_{ep}\alpha_{d_0p}h_{d_0} + \sum_{d_1,z,p,q} (\delta_{ep}\beta^z_{d_1pq}h^1_{d_1} + \delta_{eq}\beta^z_{d_1pq}h^2_{d_1}), \quad \forall e \in E; \tag{1e}$$

$$f_e \leq C_e, \quad \forall e \in E; \tag{1f}$$

The objective function (1a) is the total network delay. Constraint (1b) and (1c) dictate that the routing fractions of one demand should sum up to 1. Constraint (1d) states that the utilization of computation resources on a node should not exceed the utilization upper bound. Similar to previous work [7–9], we assume that, for each demand d_1, the amount of processing needed on a computing node is proportional to the traffic routed to it. Constraint (1e) is to calculate the traffic volume on each link. It is possible that some link is used by both segments of a demand that requires processing, which is considered by the second summation term of (1e). Note that the utilization of computation resources is limited by some configurable threshold, ρ, rather than being optimized in the objective. By changing the value of ρ, we can trade-off between the resilience to computation failures and the efficiency of the deployed computation resources. The number of variables in the formulation is $O(|V|^2|Z|k^2)$, approximately reduced by a factor of $|E|$ compared to $O(|V|^2|Z||E|)$ in the node-link formulation in [9], since k is usually a small constant compared to the number of edges.

The bottleneck of the above model is the dimension of variable β, which is quadratic to the path budget k. By decoupling the routing variables for both segments and introducing a new variable, $r^z_{d_1}$, the total amount of processing of demand d_1 done on computing node z, we can reduce the total number of routing variables to be linear with the path budget k, and the total number of variables becomes $O(|V|^2|Z|k)$. The resulting PRINP model is as follows:

$$\underset{x,r\geq 0}{\text{minimize}} \quad \mathcal{F} = \sum_e \frac{f_e}{C_e - f_e} \tag{2a}$$

$$\text{subject to} \quad \sum_p x_{d_0 p} = h_{d_0}, \quad \forall d_0 \in D_0; \tag{2b}$$

$$\sum_z r^z_{d_1} = W_{d_1}, \quad \forall d_1 \in D_1; \tag{2c}$$

$$\sum_{d_1} r^z_{d_1} \leq \rho N_z, \quad \forall z \in Z; \tag{2d}$$

$$\sum_p x^{z1}_{d_1 p} = \frac{r^z_{d_1}}{W_{d_1}} h^1_{d_1}, \quad \forall d_1 \in D_1, z \in Z; \tag{2e}$$

$$\sum_p x^{z2}_{d_1 p} = \frac{r^z_{d_1}}{W_{d_1}} h^2_{d_1}, \quad \forall d_1 \in D_1, z \in Z; \tag{2f}$$

$$f_e = \sum_{d_0,p} \delta_{ep} x_{d_0 p} + \sum_{d_1,z,p} \delta_{ep} x^{z1}_{d_1 p} + \sum_{d_1,z,q} \delta_{eq} x^{z2}_{d_1 q}, \quad \forall e \in E; \tag{2g}$$

$$f_e \leq C_e, \quad \forall e \in E; \tag{2h}$$

where $x_{d_0 p}$ is the traffic allocation of d_0 on its candidate path p, and $x^{z1}_{d_1 p}$ and $x^{z2}_{d_1 p}$ are the traffic allocations of d_1 on a candidate path to/from a computing node z on its first and second segments respectively. In constraint (2d), the computation resources are allocated among all compute nodes explicitly. Instead of managing the traffic split ratio of the two segments together, each segment is considered independently in constraint (2e) and (2f). The other constraints are similar to those in the model (1).

To further speed up, we propose two ways to further reduce the dimension and speed up the calculation.

3.1.2 Reduce the Size of the Path Set.

For a regular demand $d_0 \in D_0$, there are at most k different paths. For a demand $d_1 \in D_1$ that requires processing, the total number of paths is $k^2|Z|$, considering the combination of compute nodes and paths of the two segments. The latter dominates and is beyond adequate. To reduce the number of paths for $d_1 \in D_1$ and accelerate the resolution time, it is reasonable to cut down the path budget for the P_1 path set. Specifically, the PRINP-Red model uses SMORE path set with path budget of k_0 for demands in D_0, while using KSP path set with budget of k_1 for demands in D_1, where $k_1 < k_0$.

3.1.3 Separated Computation Demand Allocation and Traffic Routing.

The previous optimization model solves for the optimal computation allocation and traffic allocations jointly. To reduce the computation time, we could separate the computation resource allocation and routing decisions into two sub-problems, each of which is an LP with a much smaller size. In the first sub-problem, we allocate the computation demands to compute nodes by solving a simplified allocation problem. With computing demands allocated to all compute nodes, the RINP problem can be reduced to a standard MCF problem. The second sub-problem is then the resulting MCF problem and can be solved with the regular path-based formulation. We call this separated allocation the PRINP-Sep model.

We construct the first sub-problem as follows,

$$\underset{\eta\geq 0}{\text{minimize}} \quad \sum_{d_1,z} \eta^z_{d_1} (h^1_{d_1} l_{s(d_1),z} + h^2_{d_1} l_{z,t(d_1)}) \tag{3a}$$

$$\text{subject to} \quad \sum_z \eta^z_{d_1} = 1, \quad \forall d_1 \in D_1; \tag{3b}$$

$$\sum_{d_1} (\eta^z_{d_1} W_{d_1}) \leq \rho' N_z, \quad \forall z \in Z; \tag{3c}$$

where $\eta^z_{d_1}$ is the ratio of demand $d_1 \in D_1$ that goes through the computing node z, $l_{s,t}$ represents the shortest path length (hop-count) between node s and t, and is pre-computed. In the objective (3a), instead of minimizing the congestion delay on each link, we minimize the total traffic imposed on the network by the two-segment routing by calculating the product of traffic volume and path length. To obtain the computation demand allocation solution, we assume the least-hop paths on both segments are always feasible for all demands in this stage. The routing solution will be optimized in the second stage. In constraint (3c), the parameter ρ' controls the load balancing of compute nodes. We calculate the ρ' as follows,

$$\rho' = \min\left\{(1+\epsilon)\frac{\sum_{d_1} W_{d_1}}{\sum_z N_z}, \rho\right\},$$

where ϵ is a hyper-parameter of the margin allowed for surpassing the network-wide average computation utilization on any computing node.

With the computation demand allocation solution η, we find the routing solution for each demand d_1 consisting of the traffic flows from the source $s(d_1)$ to each of the computation node z with $\eta^z_{d_1}$, then from z to the destination $t(d_1)$. More specifically, for the purpose of traffic routing, we replace the set D_1 of demands with processing needs by a new set $\bar{D}_1$ of demands without processing needs:

$$\begin{aligned}
\bar{D}_1 &= \bar{D}^1_1 \cup \bar{D}^2_1 \\
\bar{D}^1_1 &= \{d | \forall d_1 \in D_1, z \in Z : \eta^z_{d_1} > 0, s(d) = s(d_1), t(d) = z, \} \\
\bar{D}^2_1 &= \{d | \forall d_1 \in D_1, z \in Z : \eta^z_{d_1} > 0, s(d) = z, t(d) = t(d_1)\}
\end{aligned}$$

The traffic volumes of the demands in $\bar{D}_1$ are denoted as $\bar{h}$, and $\bar{h}(d) = h^1_{d_1}\eta^z_{d_1}$ if $d \in \bar{D}^1_1$, and $\bar{h}(d) = h^2_{d_1}\eta^z_{d_1}$ if $d \in \bar{D}^2_1$. Then the routing solution can be obtained by solving the following MCF without considering processing needs:

$$\underset{\alpha,\bar{\alpha}\geq 0}{\text{minimize}} \quad \mathcal{F} = \sum_e \frac{f_e}{C_e - f_e} \tag{4a}$$

$$\text{subject to} \sum_p \alpha_{d_0 p} = 1, \quad \forall d_0 \in D_0; \tag{4b}$$

$$\sum_p \bar{\alpha}_{d_1 p} = 1, \quad \forall d_1 \in \bar{D}_1; \tag{4c}$$

$$f_e = \sum_{d_0,p} \delta_{ep}\alpha_{d_0 p} h_{d_0} + \sum_{d_1,p} \delta_{ep}\bar{\alpha}_{d_1 p}\bar{h}_{d_1} \tag{4d}$$

$$f_e \leq C_e, \quad \forall e \in E; \tag{4e}$$

where $\bar{\alpha}$ is the routing variable of similar meaning to α, but the corresponding path set is P_1 instead of P_0.

Note that these two modifications are independent of each other. The final variation is a model that contains both, called the PRINP-Max model. It first allocates the computation demands by solving the optimization model (3), and transforms the demands that require processing into new regular demands as in model (4). These transformed demands use smaller path sets than the original regular demands.

3.2 Restoration After Failure

Failures can happen to both communication and computation elements. The goal of resilient RINP is to quickly and maximally restore flows affected by failures.

3.2.1 Failure Scenarios. We consider four types of failures, as listed below:

I *Link Failure.* A communication link is down and all link capacity is lost.

II *Computation Resource Failure.* All computation resources on a computing node are not available, but the node can still forward traffic.

III *Simple Node Failure.* A node that has no computation resources is called a simple node. In this case, the failed simple node cannot forward traffic anymore, and all the links attached to it could be viewed as failed.

IV *Computing node failure.* A node with computation resources is down. It loses all functions, including computation and forwarding.

Under failures, the capacity of the failed links or nodes changes. In the following, we focus on complete failures, i.e. the capacity of failed network element drops to zero. Our model can also be extended to work with partial failure scenarios, where only a fraction of the original capacity is available. We will refer to the failures as type-I to type-IV failures as defined above.

3.2.2 Restoration Approaches. Since resilience is not explicitly considered in the normal stage, the TE system reacts in a best-effort way to restore the flows upon failures. There are mainly three restoration approaches.

The first one is to do a **global rerouting**, i.e., recalculate the computation allocation and traffic routing for all flows under the new resource constraints after each failure. It can promise a globally optimal solution for each failure scenario. However, it can cause many route changes, especially for flows that would otherwise not be affected. Recalculating the solutions and implementing them for all flows may take a long time, especially in large networks.

The second approach is to reroute only computation and traffic loads on the failed elements by means of **backup paths**. Handling link failures is easy, where all traffic on a failed link is treated as a new demand to get routed in the residual graph. Node failures or even computation resource redistribution can be handled in a similar way, more complicated but acceptable. The main disadvantage of this approach is that it requires establishing new tunnels to restore the traffic. It takes up to a few seconds to update a single switch rule [10], making the restorations too slow. An alternative way to avoid this latency is to build all restoration tunnels in advance, but it would result in very large flow tables in switches. Additionally, when the affected traffic cannot be fully restored, further adjustments on the computation allocation and traffic routing of other unaffected flows may be necessary to find a feasible restoration solution.

The last approach is to reroute the affected flows in an end-to-end manner, called **partial rerouting**. Traffic volumes on failed paths are counted and then rerouted by other surviving paths. If the path set is diverse and resilient to failures, we could expect that most demands still have candidate paths that survived the failure and can be used to restore them. The traffic split ratios on candidate paths of the affected flows need to be adjusted on their ingress routers. The latency of updating the traffic split ratio at the ingress is usually a few RTTs [12], so the response time depends mainly on the optimization solving time. In practice, the solving time observed is low (see section 4.3 for details), thus the restoration runs fast.

3.2.3 Restoration Problem. Based on the above analysis, we choose the last approach, partial rerouting to do the restoration. The restoration scheme is separated into two phases, the preparation phase, and the solving phase. To minimize service interruption, we will not change computation allocation and routing for flows not traversing the failed elements. As a result, restoration can only use computation and communication resources not used by unaffected flows. After having the solution in the normal stage, we generate the residual graph with the remaining communication and computation capacities to save time for the restoration stage.

After a failure occurs, the affected flows are identified. They become the demands for the restoration RINP. The restoration demands are also in two sets, namely $\tilde{D}_0$ and $\tilde{D}_1$. The bandwidth used by them in normal operation is added back to the residual graph, and the computation resources used by them are also added back to the residual graph. If the source node or the destination node of a demand fails, this demand will not be counted. The capacities of the failed elements are updated accordingly. Results of the preparation phase are (1) the restoration demand sets $\tilde{D}_0$ and $\tilde{D}_1$, (2) the updated capacities of all edges $\tilde{C}_e$ and the remaining computation resources on compute nodes $\tilde{N}_z$, and (3) the traffic volume from all unaffected flows on all edges g_e. The additional notations are summarized in Table 3.

Then, the restoration demands are rerouted in the solving phase by solving a routing optimization model called the RRINP problem, which is similar to the optimization model (1). Note that, each demand to be restored still uses the same set of candidate paths as in the normal stage. Only the traffic split ratios among candidate

Table 3. Additional notations for restoration

Symbol	Description
$\tilde{N}_z$	Remaining computation capacity on node $z \in Z$
$\tilde{\rho}$	Upper bound of resources utilization in restoration
$\tilde{C}_e$	Updated capacity of edge e
g_e	Unaffected traffic volume on edge e
$\tilde{D}_0$	Set of restoration demands that do not need processing
$\tilde{D}_1$	Set of restoration demands that need processing
s_d	Slack variable for the unrestored fraction of a demand d
$\tilde{\mathcal{F}}$	The network delay term in the objective
P	Penalty weight for unrestored demand

paths need to be adjusted at ingress points. This leads to a very short reconfiguration time.

$$\underset{\alpha,\beta,s\geq 0}{\text{minimize}} \quad \hat{\mathcal{F}} = \sum_e \frac{f_e}{C_e - f_e} + P \times (\sum_{d_0} s_{d_0} + \sum_{d_1} s_{d_1}) \tag{5a}$$

$$\text{subject to} \quad \sum_p \alpha_{d_0 p} + s_{d_0} = 1, \quad \forall d_0 \in \tilde{D}_0; \tag{5b}$$

$$\sum_{z,p,q} \beta^z_{d_1 pq} + s_{d_1} = 1, \quad \forall d_1 \in \tilde{D}_1; \tag{5c}$$

$$\sum_{d_1} (\sum_{p,q} \beta^z_{d_1 pq}) W_{d_1} \leq \tilde{\rho} \tilde{N}_z, \quad \forall z \in Z; \tag{5d}$$

$$f_e = g_e + \sum_{d_0,p} \delta_{ep} \alpha_{d_0 p} h_{d_0} + \sum_{d_1,z,p,q} (\delta_{ep} \beta^z_{d_1 pq} h^1_{d_1} + \delta_{eq} \beta^z_{d_1 pq} h^2_{d_1}), \quad \forall e \in E; \tag{5e}$$

$$f_e \leq \tilde{C}_e, \quad \forall e \in E; \tag{5f}$$

In the objective function (5a), besides the first term corresponding to the network delay $\tilde{\mathcal{F}}$, there is an additional penalty term $\tilde{\mathcal{P}}$ for the unrouted demands. With a large weight P for the penalty term $\tilde{\mathcal{P}}$, the solution will try to restore as many demands as possible. Our penalty term seeks to minimize the number of demands that are not restored. It can also be customized for other objectives, e.g. minimizing the volume of unrestored demands. The constraints are similar to those in model (1). In (5e), the traffic rate on edge e is the summation of traffic generated by flows not affected by the failure (the first term g_e) and the restoration traffic routed on this link. After solving the restoration model, the performance is evaluated by two metrics: the changes in network delay and the amount of unrestored demands.

4 Evaluations

4.1 Simulation Setups

4.1.1 Dataset. We select 12 topologies from SNDlib [25]. The topologies have TMs corresponding to normal traffic, and we use them to generate more demand sets. 1-degree nodes are removed and demands from or to a 1-degree node are merged to its only neighbor. Demands with traffic volume less than 5% of the maximum traffic volume are dropped, and the remaining demands account for more than 80% of the total traffic in most topologies. Lastly, topologies with too few nodes, edges, or demands are dropped. The statistics of the used topologies are reported in Table 4.

Table 4. Topology information. The last column is the number of demands after filtering.

Name	Nodes	Edges	Demands
cost266	37	57	758
france	25	45	251
germany50	50	88	92
giul39	39	172	1471
india35	35	80	595
janos-us-ca	39	122	64
janos-us	26	84	236
nobel-eu	28	41	183
norway	27	51	702
pioro40	40	89	780
ta2	64	107	156
zib54	53	80	114

Next, we generate computing node sets and demand sets. We randomly select some nodes as compute nodes in each topology, and the number of compute nodes selected is 8 for all topologies. To generate one demand set for a topology, 90% of its filtered demands are selected and each selected demand is split into two parts. One part with 25-50% volume becomes demands without processing needs (demand set D_0), and the other part are demands with processing needs (demand set D_1) if neither of the source or destination is a computing node. The traffic volume before and after processing may be different. For each demand in D_1, we randomly choose a factor between 0.5 and 2 to be the ratio between the volume before and after the processing.

Our model makes no assumption about the relationship between the amount of computation resources needed W_{d_1} and the traffic volumes $h^1_{d_1}$ and $h^2_{d_1}$. Two demands with the same traffic volume can have very different computation resource requirements. For the convenience of presentation, in our experiments, the computation resource requirement of a demand is assumed to be identical to the traffic volume before processing. For each topology, 40 different demand sets are generated, and the compute nodes are assigned enough amount of computation resources to handle the total computation needs.

We scale the traffic and computation volumes of demands to have different values of Maximum Link Utilization (MLU), to simulate different load-level of the overall network.

4.1.2 Baselines and Metrics. We focus on the performances of the following models.

1. **PRINP**. The PRINP model as discussed in section 3.1. For regular demands, it uses the SMORE path set, while for demands that require processing, it uses the KSP path set. The path budget is set to 8.
2. **KSP(4)**, **KSP(8)**, **SMORE(4)**, **SMORE(8)**. The PRINP model with different path sets and path budgets. In **KSP(4)**, we use the KSP path set for all demands, with path budget as 4. The meanings of notations are similar for the other three models.
3. **PRINP-Red**. The model with reduced size of path set for demands that require processing. For regular demands, it uses the SMORE path set with path budget $k_0 = 8$. For demands that require processing, it uses the KSP path set with path budget $k_1 = 4$.

4. **PRINP-Sep**. The model that separates the computation demand allocation and the traffic routing decisions. The path sets and the parameters are the same as in the PRINP model.
5. **PRINP-Max**. The final model that utilizes both separated allocations and reduced path set. The path sets and the parameter are the same as in the PRINP-Red model.

In the normal stage, we evaluate the models based on the speedup of the optimization solving time and the normalized network delay. It is meaningless to directly compare the network delays and optimizer running times, because the values are related to the traffic volumes and the topology sizes. Since the link-based RINP formulation can provide the theoretical optimal network performance, we first solve for the optimal network delay $\mathcal{F}_{opt}$ and the corresponding solver running time $\mathcal{T}_{opt}$ using the model in [9], with some modifications to include the two types of demand sets. The speedup is calculated as $\frac{\mathcal{T}_{opt}}{\mathcal{T}}$, indicating the computation time reduction factor of the model over the link-based model. The normalized network delay is calculated as $\frac{\mathcal{F}}{\mathcal{F}_{opt}}$, indicating the network delay increase ratio over the minimal delay. The value is at least 1, and the smaller the normalized network delay, the better the model's performance.

In the restoration stage, we look at the delay change and the number of unrestorable demands. The delay change is calculated as $\frac{\tilde{\mathcal{F}}-\mathcal{F}}{\mathcal{F}_{opt}}$. $\mathcal{F}$ and $\mathcal{F}_{opt}$ are of the same meaning as in the normal stage before failure happens, and $\tilde{\mathcal{F}}$ is the network delay after restoration, the first term in the RRINP objective in (5a). This term is meaningful only when the fraction of unrestored demands is low, otherwise, the decrease in the network delay could result from the much lower restored traffic volume.

4.1.3 Other Parameters.

The simulation is carried out in Python on a Laptop, with AMD Ryzen 7-5800H CPU. The optimization solver is Gurobi [26]. Piece-wise linear approximation of convex objective function is utilized to turn the convex optimization models into LP models. We set $C_e = 10,000$ for all e, $\rho = 0.8$ and $\tilde{\rho} = 1$.

To set the parameter ϵ for the PRINP-Sep model, we run an experiment with only the germany50 topology and the corresponding demand sets. The results are in Fig. 5, where the normalized performance is the resulting network delay compared to that of $\epsilon = 0.2$. The network delay slightly reduces with a larger value of ϵ when the network load is low, while it increases when the load is high. The variation is within a small range, so we choose $\epsilon = 0.2$ in the following experiments.

4.2 Performance in the Normal Stage

A good path set should help to reduce the solving time while achieving near-optimal network performance. The results from different models are compared in Fig. 6.

In Fig. 6a, the Cumulative Distribution Function (CDF) of the speedup by each model is shown, and the x-axis is in log scale. The models can be categorized into three groups in terms of the speedup. The first group contains KSP(8), SMORE(8), and PRINP, and their speedup is mostly within 20x over the link-based RINP model. The second group contains KSP(4), SMORE(4), and PRINP-Red. They can provide a speedup of 20-70x because the sizes of the path set they use are much smaller than those in the first group. The last group consists of PRINP-Sep and PRINP-Max, and they can provide a speedup of up to over 100x. It demonstrates that the separated computation and traffic allocations can greatly reduce the size of LP and shorten the solving time.

In Fig. 6b, we plot the CDF of the normalized network delay. The KSP(4) and KSP(8) models are the closest to the optimal. This is because the KSP path set has the shortest path lengths, thus reducing the routing cost and the network delay if there is no failure. The network delays of SMORE(4) and SMORE(8) models are higher

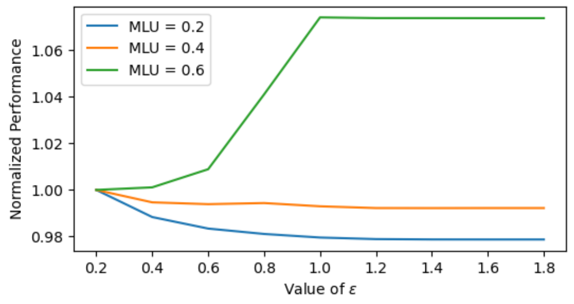
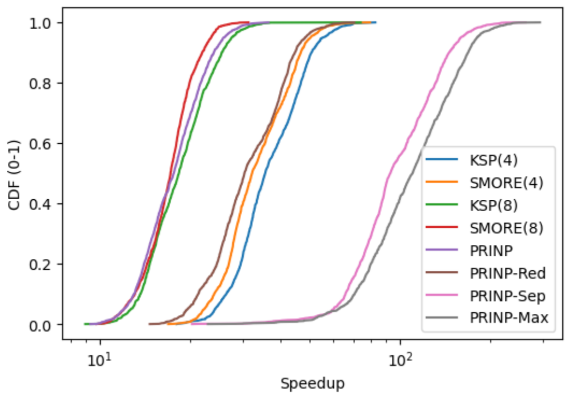
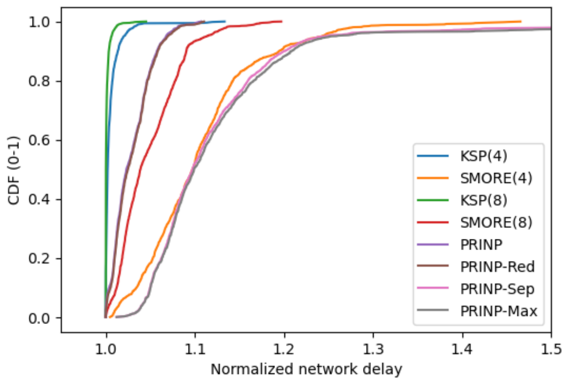


Figure 5. The influence of computation utilization margin ϵ in PRINP-Sep model on the overall performance. Different MLU curves are for different traffic scales on the same topology.



(a) CDF of Speedup



(b) CDF of Normalized Network Delay

Figure 6. Comparisons of Network Performance and Solution Speed between Different Models in Normal stage

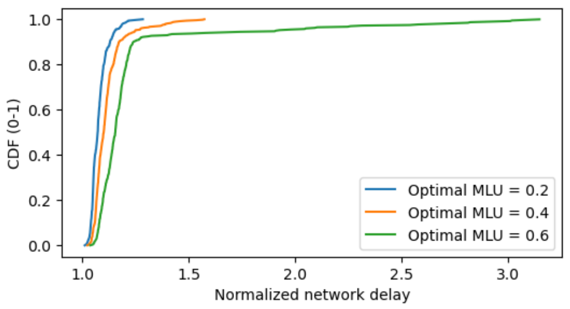


Figure 7. The influence of network load on the performance of the PRINP-Sep model

Table 5. 90th percentile of RRINP problem solving time

Model in the normal stage	Failure types			
	type-I	type-II	type-III	type-IV
PRINP	0.407	1.166	0.434	0.851
PRINP-Red	0.133	0.312	0.149	0.271
PRINP-Sep	0.430	1.083	0.473	0.897
PRINP-Max	0.140	0.277	0.148	0.260
PRINP-Max Global	3.043	2.829	2.704	2.650

than those of the models using KSP paths, since SMORE paths sacrifice path length for better path diversity, which will play an important role in the restoration stage. The PRINP and PRINP-Red models have similar performances, which are better than models using SMORE paths and not far from those using KSP paths. Even though the size of the path set in the PRINP-Red model is greatly reduced, its performance is still as good as that of the PRINP model, while it has a substantial speedup. The performance degradation of the PRINP-Sep and PRINP-Max models comes from the trade-off for solving speed over network delay. The 90th percentile of the normalized network delay of both models is about 1.25x, and the tail goes to 3x.

In Fig. 7, we plot separately the CDF of the normalized network delay of the PRINP-Sep model under different network load levels. We can see the performance of the PRINP-Sep model degrades as the network load level increases, because the network delay is more sensitive to routing when the network load level is high. It can also be seen that the normalized network delay is within 1.25x in most cases, even if the network load level is high.

4.3 Resilience during the Restoration Stage

After having the normal-stage solution for the models, we inject failures into the network system. We assume no a priori knowledge about the failure distribution, and each failed network element is drawn totally at random. We generate 5 link failure scenarios for each demand set, with total failure on one randomly selected link in each scenario. For the other 3 types of failures, we generate 3 scenarios for each demand set, again total failure on a randomly selected element. In total, there are 14 failure scenarios for each demand set. We evaluate the resilience performance over a small yet representative subset of the topologies, which consists of germany50, india35, and janos-us-ca.

Table 6. Restoration information of the PRINP-Red model

Total Number, Percentage or Average Delay Change	Failure types			
	type-I	type-II	type-III	type-IV
Affected Demands	29995	20008	25495	33052
Affected Percentage	4.62%	5.63%	5.51%	8.48%
Unrestored Demands	31	0	45	52
Delay Change	+0.090	+0.084	+0.015	+0.241
Unrestored Demands (Global)	34	0	45	51
Delay Change (Global)	+0.049	+0.061	-0.033	+0.196

4.3.1 Rerouting Time and Effectiveness.

Since the time for routing update is short [12], we mainly focus on the time spent in solving the RRINP problem. The solving time includes the preparation time. The 90th-percentiles of model solving times are listed in Table 5. The differences in solving time of models mainly come from the different path set sizes, so the models with a smaller size of path sets are generally faster in restoration. For type-II and type-IV failures, the rerouting times are longer than those of the other two types. This is because the failures with computation resources may involve much more demands, leading to a larger restoration demand set in the RRINP problem. The last row is the rerouting time if we reroute globally all the demands under failure, with the path set in the PRINP-Red model. It is obtained by solving the RRINP problem when the $\tilde{D}_i$ contains all the demands in D_i whose source or destination node does not fail and $g_e = 0$ for all edges $e \in E$. The rerouting time is much longer in global rerouting compared to the same setting in partial rerouting, because only a small portion of the demands are affected. The small size of the demand set also helps to reduce the restoration time.

To evaluate the effectiveness of the restoration stage, we measure the total number of demands affected by failures, and the fraction of them that our schemes cannot restore. For the PRINP-Red model, the numbers are shown in Table 6. The row of the affected demands contains the numbers of the affected demands under all failure scenarios of each type. The averages of the affected demands as a percentage of all demands are in the second row, and we can see most demands are not affected under failures. The total number of the affected demands under type-I failures is larger than that under type-II failures while the percentage is smaller. This is because there are more type-I failure scenarios than the others, so the total number is larger. The last four rows are the number of the demands that are not fully restored in the restoration stage and the delay change, resulting from partial rerouting in the PRINP-Red model and global rerouting. Only a few demands are not restored, either because there is no candidate path that survived the failure, or there is not enough bandwidth on the survived paths. Global rerouting has a lower delay change than partial rerouting, meaning that the network delay after restoration is lower. There is not a large gap between the partial rerouting approach and the global rerouting approach.

4.3.2 Restoration Performance.

Now we compare the restoration performance of different models under different types of failures. After failure and restoration, the network delay should increase if all the affected demands get restored, because the total available communication/computation resources in the network are reduced and the new solution should be at least no better than

Table 7. The 90th percentile delay changes and the number of unrestored demands of the models under different types of failures

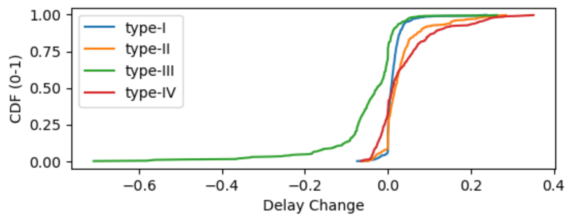
Model	90th Percentile of Normalized Delay	90th Percentile of Delay Changes				Number of Unrestored Demands			
		Type-I	Type-II	Type-III	Type-IV	Type-I	Type-II	Type-III	Type-IV
KSP(4)	1.014	+0.072	+0.155	+0.059	+0.276	855	0	2880	1938
SMORE(4)	1.248	+0.162	+0.212	+0.232	+0.512	202	0	856	842
KSP(8)	1.004	+0.067	+0.152	+0.062	+0.270	79	0	1106	979
SMORE(8)	1.107	+0.096	+0.173	+0.098	+0.351	31	0	**45**	34
PRINP	1.058	+0.080	+0.158	+0.082	+0.295	31	0	**45**	**30**
PRINP-Red	1.060	+0.083	+0.160	+0.084	+0.306	31	0	**45**	52
PRINP-Sep	1.220	+0.060	+0.109	+0.047	+0.254	31	0	47	35
PRINP-Max	1.233	+0.063	+0.116	+0.055	+0.263	**30**	0	48	56

the original optimal solution. However, if all the paths of some demand failed, the demand can no longer be routed, thus reducing the total traffic volume and the network delay. To address the issue, we consider the network delay term $\tilde{\mathcal{F}}$ in the RRINP solution and the number of unrestored demands together. The results are shown in Table 7.

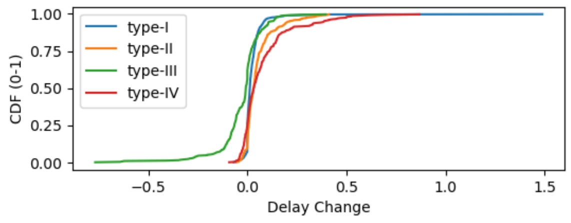
The KSP(4) and KSP(8) models have a good normalized delay and small delay changes, but they also have a large number of unrestored demands. The path lengths in the KSP path sets are short, but the paths may overuse some short edges, so the resilience is bad. The normalized delay and delay changes of the SMORE(4) and SMORE(8) models are worse than the ones with KSP path sets, but the unrestored demands under the SMORE paths set are much less. The SMORE path sets trade off the normal-stage performance for better resilience in the restoration stage.

The SMORE(8), PRINP, PRINP-Red, PRINP-Sep, and PRINP-Max models have a similar number of unrestored demands. The PRINP model outperforms the SMORE(8) model in both the normal stage and the restoration stage. This demonstrates that the SMORE path sets for the demands that require processing only bring additional cost but do not add to the resilience. With traffic relayed by compute nodes, the KSP paths can provide enough diversity for resilience. The difference between the PRINP model and the PRINP-Red mode is small, meaning that the reduced size of the path set does not add much to the delay or harm the resilience. The PRINP-Sep and PRINP-Max models have larger normalized delays than the PRINP and PRINP-Red models become they don't jointly optimize computation demand allocation and traffic routing. This is the trade-off between the performance and the fast resolution time.

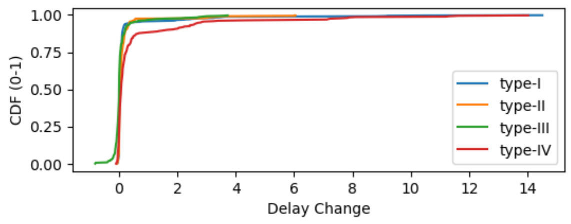
In terms of the failure types, type-II failures have the lowest harm to the network, while type-IV failures produce the highest negative impact. In type-II failures, only some computation resources are down. All the links and nodes can still forward traffic and no path fails. As long as there are still enough computation resources in the network to do the processing and enough bandwidth on the links, there will be no failed demands. The other three types of failures could all lead to some failed paths. Node failures are of higher impact because the number of paths involved is generally larger than that of a link failure. In general, the numbers of unrestored demands in type-III and type-IV failures are higher than in type-I failures. Type-IV failures have a larger delay change because all the demands that take computation resources at the node would need rerouting. This would involve a greater number of demands as shown in Table 6, bringing higher loads to the network during restoration.



(a) Optimal MLU = 0.2



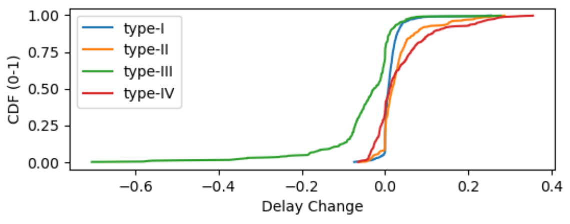
(b) Optimal MLU = 0.4



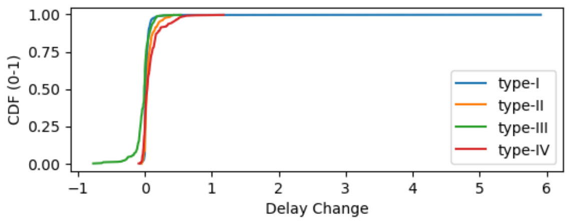
(c) Optimal MLU = 0.6

Figure 8. The delay change of the PRINP model under different network load levels

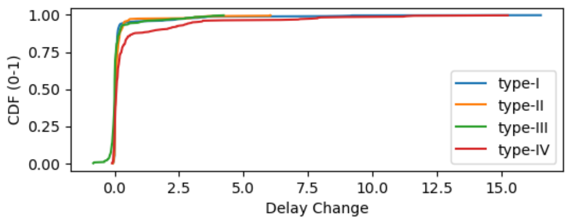
To understand how the network load influences restoration performance, we plot the CDF of delay changes separately. Fig. 8 is the result for the PRINP model and Fig. 9 is the result for the PRINP-Red model. In most failure scenarios, there is not a huge change in the network delay after the restoration for both models, because their path sets are resilient to survive the failures and avoid congestion. The network delays may even decrease, mostly for two reasons. One is for type-III or type-IV failures, the source or destination of some demands may fail. These failed demands could reduce the total traffic volume, leading to a decrease in the network delay. The other is the change of the limit on compute node utilization. In the



(a) Optimal MLU = 0.2



(b) Optimal MLU = 0.4



(c) Optimal MLU = 0.6

Figure 9. The delay change of the PRINP-Red model under different network load levels

normal stage, the limit is $\rho = 0.8$, while in the restoration stage, the limit changes to $\tilde{\rho} = 1$. Some demands have far-away compute nodes in the normal stage, because the computation resources on compute nodes with much lower costs have reached the limit. These demands could benefit from the additional computation resources to have a lower delay. However, when the network load level is high, the decrease in the volume cannot compensate for the increase in congestion. We can observe that for both models, the endpoint of the CDF curve goes to a high value when the network load is high.

In terms of the worst-case performance, the PRINP model is better than the PRINP-Red model, especially in Fig. (8b) and Fig. (9b). The reduced path set size in the PRINP-Red model does help to accelerate the model solution, but a larger path set could still be more resilient upon failures.

5 Conclusions

While applications can benefit from various processing inside edge networks, routing of flows with computation needs is significantly more complex than the traditional traffic routing, and has to be resilient against unexpected failures on communication and computation elements. In this paper, we developed a novel path-based two-stage traffic engineering scheme to trade-off between routing model complexity, network performance in the normal stage, and restoration efficiency upon failures. Our path-based routing formulation significantly speedups routing calculation, and path-based traffic splitting can be easily implemented/updated on ingress nodes. We demonstrated that computation allocation and traffic routing can be conducted sequentially to further speed-up routing calculation. After network failures, our fast restoration routing scheme minimizes the disruption to application flows and maximizes the restored traffic using the survived communication and computation resources. Our two-stage TE schemes were evaluated extensively using real network instances. In the normal stage, our scheme achieves near-optimal performance with up to 50-100x speedup compared to link-based optimal routing models. In the restoration stage, in various evaluated failure scenarios, our scheme can restore most of the affected traffic with up to 10x speedup compared to globally rerouting all the flows. Our fast restoration routing schemes are ready to be implemented in SDN/NFV-equipped edge networks to improve their performance and resilience.

Acknowledgments

This research was partially supported by the USA National Science Foundation (NSF) under award # 2148309.

References

[1] B. Han, V. Gopalakrishnan, L. Ji, and S. Lee, "Network function virtualization: Challenges and opportunities for innovations," *IEEE Communications Magazine*, vol. 53, no. 2, pp. 90–97, 2015.
[2] B. Varghese, N. Wang, S. Barbhuiya, P. Kilpatrick, and D. S. Nikolopoulos, "Challenges and opportunities in edge computing," in *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, 2016, pp. 20–26.
[3] R. Huang, J. Pedoeem, and C. Chen, "Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers," in *2018 IEEE International Conference on Big Data (Big Data)*, 2018, pp. 2503–2510.
[4] H. Yeo, H. Lim, J. Kim, Y. Jung, J. Ye, and D. Han, "Neuroscaler: Neural video enhancement at scale," in *Proceedings of the ACM SIGCOMM 2022 Conference*, ser. SIGCOMM '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 795–811. [Online]. Available: https://doi.org/10.1145/3544216.3544218
[5] P. Hedman, P. P. Srinivasan, B. Mildenhall, J. T. Barron, and P. Debevec, "Baking neural radiance fields for real-time view synthesis," *ICCV*, 2021.
[6] L. Lin, X. Liao, H. Jin, and P. Li, "Computation offloading toward edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1584–1607, 2019.
[7] Z. Cao, S. S. Panwar, M. Kodialam, and T. V. Lakshman, "Enhancing mobile networks with software defined networking and cloud computing," *IEEE/ACM Transactions on Networking*, vol. 25, no. 3, pp. 1431–1444, 2017.
[8] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-commodity flow with in-network processing," in *Algorithmic Aspects of Cloud Computing*, Y. Disser and V. S. Verykios, Eds. Cham: Springer International Publishing, 2019, pp. 73–101.
[9] L. Mei, J. Gou, J. Yang, Y. Cai, and Y. Liu, "On routing optimization in networks with embedded computational services," 2022.
[10] 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," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, p. 3–14, aug 2013. [Online]. Available: https://doi.org/10.1145/2534169.2486019
[11] 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)*. Renton, WA: USENIX Association, Apr. 2018, pp. 157–170. [Online]. Available: https://www.usenix.org/conference/nsdi18/presentation/kumar
[12] F. Abuzaid, S. Kandula, B. Arzani, I. Menache, M. Zaharia, and P. Bailis, "Contracting wide-area network topologies to solve flow problems quickly," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, Apr. 2021, pp. 175–200. [Online]. Available: https://www.usenix.org/conference/nsdi21/presentation/abuzaid
[13] Z. Xu, F. Y. Yan, R. Singh, J. T. Chiu, A. M. Rush, and M. Yu, "Teal: Learning-accelerated optimization of wan traffic engineering," 2023.
[14] M. Pióro and D. Medhi, *Routing, Flow, and Capacity Design in Communication and Computer Networks*. San Francisco: The Morgan Kaufmann Series in Networking, 2004.
[15] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the nfv service distribution problem," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, 2017, pp. 1–9.
[16] K. Poularakis, J. Llorca, A. M. Tulino, and L. Tassiulas, "Approximation algorithms for data-intensive service chain embedding," in *ACM Mobihoc 2020.*

[17] M. Rost and S. Schmid, "Charting the complexity landscape of virtual network embeddings," in *2018 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 2018, pp. 1–9.

[18] ——, "Virtual network embedding approximations: Leveraging randomized rounding," *IEEE/ACM Transactions on Networking*, vol. 27, no. 5, pp. 2071–2084, 2019.

[19] H. H. Liu, S. Kandula, R. Mahajan, M. Zhang, and D. Gelernter, "Traffic engineering with forward fault correction," in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 527–538. [Online]. Available: https://doi.org/10.1145/2619239.2626314

[20] J. Bogle, N. Bhatia, M. Ghobadi, I. Menache, N. Bjørner, A. Valadarsky, and M. Schapira, "Teavar: Striking the right utilization-availability balance in wan traffic engineering," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 29–43. [Online]. Available: https://doi.org/10.1145/3341302.3342069

[21] C. Jiang, S. Rao, and M. Tawarmalani, "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*, ser. SIGCOMM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 139–153. [Online]. Available: https://doi.org/10.1145/3387514.3405858

[22] J. Y. Yen, "Finding the k shortest loopless paths in a network," *Manage. Sci.*, vol. 17, no. 11, p. 712–716, jul 1971. [Online]. Available: https://doi.org/10.1287/mnsc.17.11.712

[23] H. Räcke, "Optimal hierarchical decompositions for congestion minimization in networks," in *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, ser. STOC '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 255–264. [Online]. Available: https://doi.org/10.1145/1374376.1374415

[24] R. Zhang-Shen and N. McKeown, "Designing a predictable internet backbone network." HotNets, 2004.

[25] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009. [Online]. Available: http://www.zib.de/orlowski/Paper/OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC.pdf.gz

[26] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com