ReNoVatE: Recovery from Node Failure in Virtual Network Embedding

Nashid Shahriar*, Reaz Ahmed*, Aimal Khan*, Shihabur Rahman Chowdhury*, Raouf Boutaba*, Jeebak Mitra†
*David R. Cheriton School of Computer Science, University of Waterloo
{nshahria | r5ahmed | a273khan | sr2chowdhury | rboutaba}@uwaterloo.ca
†Huawei Technologies, Canada
jeebak.mitra@huawei.com

***Abstract*—Network virtualization (NV) has evolved as a key enabling technology for offering the next generation network services. Recently, it is being rolled out in data center networks as a means to provide bandwidth guarantees to cloud applications. With increasing deployments of virtual networks (VNs) in commercial-grade networks with commodity hardware, VNs need to tackle failures in the underlying substrate network. In this paper, we study the problem of recovering a batch of VNs affected by a substrate node failure. The combinatorial possibilities of alternate embeddings of the failed virtual nodes and links of the VNs makes the task of finding the most efficient recovery both non-trivial and intractable. Furthermore, any recovery approach ideally should not cause any service disruption for the unaffected parts of the VNs. We take into account these issues to design a recovery approach for maximizing recovery and minimizing the cost of recovery and network disruption. We provide an Integer Linear Programming (ILP) formulation of our recovery scheme. We also propose a fast and scalable heuristic algorithm to tackle the computational complexity of the ILP solution. Evaluation results demonstrate that our heuristic performs close to the optimal solution and outperforms the state-of-the-art algorithm.**

I. Introduction

Rapid proliferation of the Internet is continuously increasing our dependence on networked services. Consequently, diverse Quality of Service (QoS) guarantees are required from the underlying network infrastructure. Network Virtualization (NV) [1] is evolving as a key technology for allowing a wide variety of online services with diverse reliability and performance requirements to co-exist and seamlessly operate on top of the same network infrastructure. An Infrastructure Provider (InP) manages a network infrastructure also known as the Substrate Network (SN), and leases network slices in the form of Virtual Networks (VN) to multiple Service Providers (SPs). An SP offers customized services on top of its VN, and is free to deploy any technology and/or communication protocol in the VN. By allowing heterogeneous VNs to coexist on a shared SN, the goal of NV is to provide flexibility, diversity, security, and manageability. Several new challenges need to be addressed to achieve these goals.

An important challenge in NV is to efficiently allocate substrate resources to VNs. This is known as the VN embedding (VNE) problem [2] that maps virtual nodes and links of a VN request on substrate nodes and paths (a sequence of substrate links), respectively, while satisfying physical resource constraints. The VNE problem is $\mathcal{NP}$-hard and has been studied from various perspectives [2]. One particular aspect of VNE that has received much attention recently is Survivable Virtual Network Embedding (SVNE). SVNE approaches deal with substrate resource (i.e., nodes or links) failures that are not a rare event in large networks [3], [4]. Surviving failures is even more challenging in NV, since the shared nature of VNs exposes them to a more vulnerable state than that of a non-virtualized network. For instance, a link failure in the SN may cause multiple virtual links to fail, which may significantly degrade service performance and reliability of VNs.

A number of mechanisms have been proposed to increase VN reliability against substrate resource failures. These mechanisms can be broadly classified into two categories [5]: a) proactively provision disjoint redundant resources as backup [6], [7] and b) reactively re-embed the failed nodes and links of a VN on the available resources after a failure has occurred [8], [9]. Proactive approaches offer immediate recovery from failures at the expense of backup resource reservation [10]–[12]. However, preallocating backup resources for multiple failures resulting from a substrate node failure can be extremely expensive [11], [13]. Instead, an SP may prefer to reactively re-embed the failed part of its VN to avoid the huge cost of preallocated backup resources in a failure-prone SN. The SP can adopt a load balanced VN embedding strategy to leave higher amount of available resources during re-embedding. Moreover, in case of permanent substrate node failures (*e.g.*, hardware malfunction), the nodes and links of all the affected VNs have to be re-embedded. These scenarios motivate us to study the problem of re-embedding a batch of VNs affected by a substrate node failure.

While VN embedding is already intractable, the combinatorial number of sequences of VNs in a batch re-embedding further increases the complexity [14]. In addition, any solution must be significantly fast (*e.g.*, in the order of milliseconds) to meet the stringent timing requirement usually imposed by Service Level Agreements (SLAs). To meet such requirements, we consider the re-embedding of only the failed nodes and links of a batch of VNs without disrupting their unaffected parts. While re-embedding the failed part as well as the unaffected part of an active VN may incur lower costs of re-embedding [8], [15], it will require additional time for virtual node migration and virtual link re-configuration, which may increase VN downtime. Further, the reactive approaches in [9], [16] opt for the recovery of all the failed nodes and links of a VN. If the SN does not have adequate resources to recover all the failed nodes and links, those approaches re-embed the complete VN, turning it inactive for a while. Unlike those

978-3-901882-85-2

approaches, our model allows partial recovery of an affected VN while maximizing the number of recoveries across all the affected VNs. This can be the sought-after choice of an InP who wants to treat all the affected VNs fairly in a resource constrained SN. Other design alternatives such as prioritizing the affected VNs based on SLA requirements, impacts of failure or profits can be accommodated by slightly modifying the proposed model.

In this paper, we focus on the problem of **Re**covering from a **No**de failure in **V**irtu**a**l Ne**t**work **E**mbedding (*ReNoVatE*). It accepts a batch of VN failures resulting from a single substrate node failure, and produces alternate embeddings for the failed virtual nodes and links. The objective is to maximize the number of recovered virtual links across all the affected VNs while minimizing total bandwidth required for recovery. We formulate *ReNoVatE* as an Integer Linear Programming (ILP) based optimization model, namely *Opt-ReNoVatE*. Since *Opt-ReNoVatE* cannot scale to large instances of the problem, we devise an efficient heuristic algorithm, called *Fast-ReNoVatE*, to find satisfactory solutions within prescribed time limits. We evaluate *Fast-ReNoVatE* by extensive simulations and compare it with *Opt-ReNoVatE*, as well as with the most related state-of-the-art proposal in the literature [9]. Evaluation results demonstrate that *Fast-ReNoVatE* performs close to *Opt-ReNoVatE* and outperforms the state-of-the-art solution in terms of i) number of recovered virtual links, ii) cost of recovery, and iii) execution time.

The rest of the paper is organized as follows. In Section III, we present the system model and formally introduce the problem. Section IV presents *Opt-ReNoVatE* for solving the problem optimally. The heuristic solution, *Fast-ReNoVatE*, is presented in Section V. Evaluation results are presented in Section VI. Section II presents the related literature. Finally, we summarize our findings and conclude in Section VII.

II. Related Work

Various works in the literature investigated different failure scenarios in NV including single link [7], [17], [18], multiple link [19], single node [13], [20], and single regional failure [21]. We discuss the related approaches for single node failure, and contrast them with our solutions for *ReNoVatE*.

Most of the approaches focusing on substrate node (SNode) failure are pro-active [10], [11], [13]. Yu *et al.*, [13] proposed a two-step method to recover a VN. The first step enhances the VN with backup virtual nodes (VNodes) and virtual links (VLinks), and the second step maps this enhanced VN on the SN. This approach, in the worst case, has to reserve a backup VNode for each VNode. In contrast, [10] designed the enhanced VN with a failure-dependent strategy to reduce backup resources. Despite the resource efficiency of this approach, it is not practical due to the large number of migrations of working VNodes. Unlike these methods, [11] presented a joint optimization strategy for allocating primary and backup resources altogether. The location constrained SVNE, to address geographically-correlated SNode failures, has been studied in [12], [22], [23]. While [22] adopts sequential embedding of working and backup VNodes, [12] embeds them jointly to minimize total bandwidth. Recently, [23], [24] proposed embedding primary and dedicated backup resources for each VNode and VLink in a VN simultaneously.

There is a growing trend towards designing survivable resource allocation schemes for embedding virtual data centers (VDCs) in cloud [25], [26], [27]. For instance, [28] proposed a scheme for provisioning VDCs with backup virtual machines and links. Bodik *et al.*, [29] proposed an optimization framework for improving survivability, while reducing total bandwidth consumption. Yeow *et al.*, [27] defines the reliability level as a function of backup resources that are shared between VNs through opportunistic pooling. Finally, [30] provided a VDC embedding framework for achieving high VDC availability by considering heterogeneous failure rates.

Reactive approaches, on the other hand, do not preallocate any backup resources. Chang *et al.*, [8] proposed a migration aware VN re-embedding algorithm to recover from an SNode failure. The algorithm allows the migration of some active VNodes and VLinks to free up some substrate resources, thus facilitating the re-mapping of the failed VNodes and VLinks. Cai *et al.*, [15] addressed the problem of optimally upgrading the existing VN in a highly evolving SN. Their goal is to minimize the upgrading cost, in addition to migration and remapping costs, while satisfying QoS constraints. Both of these approaches may need a chain of migrations to converge, thus disrupting ongoing communication in the VN. Another line of works [31], [32], originated in optical networks, proposed a hybrid mechanism to select from a set of precomputed detours for recovery during failure. However, in a highly saturated SN, this mechanism may not find adequate resources left for the recovery.

Distributed reactive approaches including, [16], [33], proposed multi-agent based algorithms to dynamically adapt the VNs in response to SN failures. When an agent detects a failure of another agent in the same cluster, the agents within the same cluster collaborate with each other to reprovision the failed VNodes and VLinks. These approaches may generate sub-optimal solutions due to the lack of global knowledge of the SN. Finally, [9] proposed a greedy algorithm to find alternate substrate resources for the affected part of a VN. In case of resource inadequacy, this approach requires remapping the entire VN resulting in prolonged service unavailability. Nonetheless, existing reactive algorithms focus on re-embedding only a single VN, whereas *ReNoVatE*, for the first time, considers partial re-embedding of a set of affected VNs to improve recovery performance.

III. System Model and Problem Statement

We first present a mathematical representation of a substrate network, virtual network, and types of failure (Section III-A). Then, we formally define the problem (Section III-B).

A. System Model

1) Substrate Network (SN): We represent the SN as an undirected graph, $G = (V, E)$, where V and E denote the set of SNodes and Substrate Links (SLinks), respectively. The set of neighbors of an SNode $u \in V$ is denoted by $\mathcal{N}(u)$. Bandwidth capacity and residual bandwidth of an SLink $(u,v) \in E$ are represented by b_{uv} and r_{uv}, respectively, while the cost of allocating one unit of bandwidth in (u,v) is C_{uv}. V^f and E^f represent the set of failed SNodes and SLinks, respectively. P_{uv} represents a path between SNodes u and v

2) Virtual Network (VN): We denote the set of VNs embedded on the SN G as $\bar{G} = \{\bar{G}_1, \bar{G}_2, \dots \bar{G}_{|\bar{G}|}\}$. Each VN $\bar{G}_i \in \bar{G}$ is represented as an undirected graph $\bar{G}_i = (\bar{V}_i, \bar{E}_i)$, where $\bar{V}_i$ and $\bar{E}_i$ are the sets of VNodes and VLinks of $\bar{G}_i$, respectively. The set of neighbors of a VNode $\bar{u} \in \bar{V}_i$ is denoted by $\mathcal{N}(\bar{u})$. Each VLink $(\bar{u},\bar{v}) \in \bar{E}_i$ has a bandwidth demand $b_{i\bar{u}\bar{v}}$. Each VN $\bar{G}_i$ has a set of location constraints, $L_i = \{L_i(\bar{u}) | L_i(\bar{u}) \subseteq V, \forall \bar{u} \in \bar{V}_i\}$, such that a VNode $\bar{u} \in \bar{V}_i$ can only be mapped to an SNode $u \in L_i(\bar{u})$. We represent this location constraint with a binary variable $\ell_{i\bar{u}u}$, defined as:

$$\ell_{i\bar{u}u} = \begin{cases} 1 & \textit{iff } \bar{u} \in \bar{V}_i \text{ can be provisioned on } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

3) Types of failure: Let, $f(\bar{u})$ and $g(\bar{u}\bar{v})$ denote the SNode and substrate path where $\bar{u}$ and $(\bar{u},\bar{v})$ have been embedded, respectively. An SNode failure results in a set of VNode and VLink failures of a VN $\bar{G}_i$ defined as $\bar{V}_i^f = \{\bar{u} \in \bar{V}_i | f(\bar{u}) \subseteq V^f\}$ and $\bar{E}_i^f = \{(\bar{u},\bar{v}) \in \bar{E}_i | (u,v) \in g(\bar{u}\bar{v}) \wedge (u,v) \in E^f\}$, respectively. There are two types of VLinks in $\bar{E}_i^f$:

Adjacent VLinks: The set of VLinks adjacent to the failed VNode $\bar{u} \in \bar{V}_i^f$ is represented by $\bar{\mathcal{E}}_i^f = \{(\bar{u},\bar{v}) | \bar{u} \in \bar{V}_i^f \wedge \bar{v} \in \mathcal{N}(\bar{u})\}$.

Independent VLinks: The set of VLinks that have failed due to the failure of some SLinks on their mapped substrate paths is denoted by $\bar{\mathtt{E}}_i^f = \{(\bar{u},\bar{v}) | (u,v) \in g(\bar{u}\bar{v}) \wedge (u,v) \in E^f \wedge \bar{u} \notin \bar{V}_i^f \wedge \bar{v} \notin \bar{V}_i^f\}$.

Finally, $\bar{V}^f = \{\bigcup \bar{V}_i^f\}$, $\bar{E}^f = \{\bigcup \bar{E}_i^f\}$, and $\bar{\mathtt{E}}^f = \{\bigcup \bar{\mathtt{E}}_i^f\}$ represent the set of failed VNodes, VLinks, and Independent VLinks of all the VNs in $\bar{G}$, respectively. Fig. 1 illustrates the embedding of two VNs, $\bar{G}_1$ with $\bar{V}_1 = \{a,b,c\}$ and $\bar{G}_2$ with $\bar{V}_2 = \{d,e\}$ on the SN G shown in the bottom. The numbers next to a VLink and an SLink represent the VLink demand and SLink residual bandwidth, respectively. The VNode mapping *i.e.*, $f(.)$ is shown by placing a VNode beside its mapped SNode and the VLink mapping *i.e.*, $g(.)$ is depicted by dashed paths between SNodes. For instance, $f(a) = \{D\}$, $f(b) = \{C\}$, $f(c) = \{G\}$, $f(d) = \{B\}$, $f(e) = \{H\}$ and $g(ab) = \{DB, BC\}$, $g(ac) = \{DH, HG\}$, $g(bc) = \{CF, FG\}$, and $g(de) = \{BD, DH\}$. We now show the impact of an SNode failure with $V^f = \{D\}$ and $E^f = \{DB, DE, DH\}$. VN $\bar{G}_1$ experiences a VNode failure with $\bar{V}_1^f = \{a\}$. Consequently, the VLinks adjacent to a fail leading to $\bar{\mathcal{E}}_1^f = \{ab, ac\}$. Note there is no Independent VLink failures in $\bar{G}_1$, and so $\bar{E}_1^f = \bar{\mathcal{E}}_1^f$. On the other hand, $\bar{V}_2^f = \phi$ since no VNode of $\bar{G}_2$ is mapped on D. However, the failure of D results in an independent VLink failure yielding $\bar{E}_2^f = \bar{\mathtt{E}}_2^f = \{de\}$. Hence, any recovery algorithm should re-embed all the affected VNodes and VLinks from $\bar{G}_1$ and $\bar{G}_2$ leaving unaffected part such as VLink bc undisrupted.

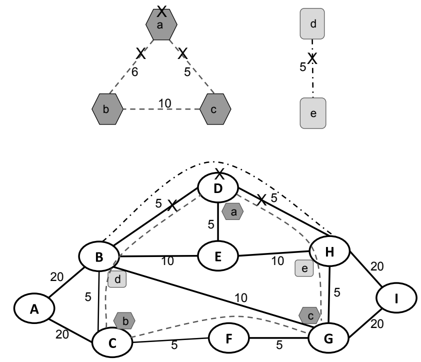


Fig. 1: VN embedding and impact of failure

B. Problem Statement

Given an SN $G = (V,E)$, a failed SNode implying $|V_f| = 1$, and a set of affected VNs $\bar{G}$ embedded on G, re-embed the failed VNodes in $\bar{V}^f$ and the failed VLinks in $\bar{E}^f$ on G such that the re-embedding achieves the following objectives:

- Primary objective: maximize the total number of recovered VLinks across all the affected VNs.
- Secondary objective: minimize the total cost of re-embedding in terms of SLink bandwidth consumption.

subject to the following constraints:

- a failed VNode $\bar{u} \in \bar{V}_i^f$ is re-embedded on exactly one SNode, $v \in L_i(\bar{u})$. In addition, multiple VNodes of the same VN cannot be mapped to an SNode. However, multiple VNodes from different VNs can share an SNode.
- a failed VLink $(\bar{u},\bar{v}) \in \bar{E}_i^f$ is re-embedded on a substrate path $P_{f(\bar{u})f(\bar{v})}$ having sufficient bandwidth to accommodate the demand of the VLink. The re-embedding cannot use a substrate path containing the failed SNode.
- VNodes and VLinks not affected by the SNode failure are not re-embedded.

IV. ILP Formulation: *Opt-ReNoVatE*

We provide an ILP formulation, *Opt-ReNoVatE*, based on the *Multi-commodity Flow Problem* formulation of *ReNoVatE*. We first present the decision variables (Section IV-A). Then, we introduce the constraints (Section IV-B) followed by the objective function of *Opt-ReNoVatE* (Section IV-C).

A. Decision Variables

The following decision variables indicate VNode and VLink embedding of a VN $\bar{G}_i \in \bar{G}$ on an SN G.

$$y_{i\bar{u}u} = \begin{cases} 1 & \textit{iff } \bar{u} \in \bar{V}_i \text{ is mapped to } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

$$x_{uv}^{i\bar{u}\bar{v}} = \begin{cases} 1 & \textit{iff } (\bar{u},\bar{v}) \in \bar{E}_i \text{ is mapped to } (u,v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

The objective of *Opt-ReNoVatE* is to recover as many failed VLinks in $\bar{E}^f$ as possible to mitigate the impact of failure. It may be possible that not all VLinks in $\bar{E}^f$ can be re-embedded due to substrate resource limitation. The following decision variable defines which VLinks are re-embedded:

$$z_{i\bar{u}\bar{v}} = \begin{cases} 1 & \textit{iff } (\bar{u},\bar{v}) \in \bar{E}_i^f \text{ is mapped to any substrate path} \\ 0 & \text{otherwise.} \end{cases}$$

B. Constraints

1) Intactness of Unaffected VNodes and VLinks: The mapping of VNodes and VLinks that are not affected by the substrate failure remains unchanged. Constraints (1) and (2) ensure that unaffected VNodes and VLinks are not re-embedded.

$$\forall \bar{G}_i \in \bar{G}, \forall \bar{u} \in \bar{V}_i \setminus \bar{V}_i^f : y_{i\bar{u}f(\bar{u})} = 1 \quad (1)$$

$$\forall \bar{G}_i \in \bar{G}, \forall (\bar{u},\bar{v}) \in \bar{E}_i \setminus \bar{E}_i^f, \forall (u,v) \in g(\bar{u}\bar{v}) : x_{uv}^{i\bar{u}\bar{v}} = 1 \quad (2)$$

2) Exclusion of Failed SNodes and SLinks from re-embedding: The failed VNodes or VLinks cannot use any of the failed SNodes or SLinks during re-embedding. Constraint (3) ensures that the failed VNodes are not re-embedded on the failed SNodes, and (4) ensures that the failed VLinks are not re-embedded on substrate paths containing a failed SLink.

$$\forall \bar{G}_i \in \bar{G}, \forall \bar{u} \in \bar{V}_i^f, \forall u \in V_f : y_{i\bar{u}u} = 0 \quad (3)$$

$$\forall \bar{G}_i \in \bar{G}, \forall (\bar{u},\bar{v}) \in \bar{E}_i^f, \forall (u,v) \in E_f : x_{uv}^{i\bar{u}\bar{v}} = 0 \quad (4)$$

3) Link Mapping Constraints: Constraint (5) prevents over-commitment of SLink bandwidth. Constraint (6) ensures that the in-flow and out-flow of each SNode is equal except at the SNodes where the endpoints of a failed VLink are embedded. Finally, constraint (7) ensures that if a VLink $(\bar{u},\bar{v})$ is selected to be re-embedded due to the failure of $\bar{u}$, there is some flow from the SNode u where $\bar{v}$ is embedded already.

$$\forall (u,v) \in E : \sum_{\forall \bar{G}_i \in \bar{G}} \sum_{\forall (\bar{u},\bar{v}) \in \bar{E}_i} x_{uv}^{i\bar{u}\bar{v}} \times b_{i\bar{u}\bar{v}} \leq b_{uv} \quad (5)$$

$$\forall \bar{G}_i \in \bar{G}, \forall (\bar{u},\bar{v}) \in \bar{E}_i^f, \forall u \in V \setminus f(\bar{v}) : \sum_{\forall v \in \mathcal{N}(u)} (x_{uv}^{i\bar{u}\bar{v}} - x_{vu}^{i\bar{u}\bar{v}}) \leq y_{i\bar{u}u} - y_{i\bar{v}u} \quad (6)$$

$$\forall \bar{G}_i \in \bar{G}, \forall (\bar{u},\bar{v}) \in \bar{E}_i^f, \forall u \in f(\bar{v}) : \sum_{\forall v \in \mathcal{N}(u)} (x_{uv}^{i\bar{u}\bar{v}} - x_{vu}^{i\bar{u}\bar{v}}) = z_{i\bar{u}\bar{v}} \quad (7)$$

4) Node Mapping Constraints: First, constraint (8) ensures that re-embedding of a failed VNode should be done according to the provided location constraint set. Second, constraint (9) makes sure that a VNode should be mapped to at most an SNode in the SN. Third, constraint (10) enforces that an SNode will not host more than one VNodes from the same VN. Finally, constraint (11) ensures that if a VLink $(\bar{u},\bar{v}) \in \bar{\mathcal{E}}_i^f$ is selected to be re-embedded due to the failure of $\bar{u}$, the VNode $\bar{u}$ must be re-embedded on an SNode according to the location constraint. Here, λ is a very large integer that turns the left side of (11) into a fraction between 0 and 1 when any of the $z_{i\bar{u}\bar{v}}$ is 1. This enforces the right side of (11) to become 1, thus ensuring the failed VNode to be re-embedded.

$$\forall \bar{G}_i \in \bar{G}, \forall \bar{u} \in \bar{V}_i^f, \forall u \in V : y_{i\bar{u}u} \leq \ell_{i\bar{u}u} \quad (8)$$

$$\forall \bar{G}_i \in \bar{G}, \forall \bar{u} \in \bar{V}_i^f, : \sum_{u \in V} y_{i\bar{u}u} \leq 1 \quad (9)$$

$$\forall \bar{G}_i \in \bar{G}, \forall u \in V : \sum_{\bar{u} \in \bar{V}_i} y_{i\bar{u}u} \leq 1 \quad (10)$$

$$\forall \bar{G}_i \in \bar{G}, \forall \bar{u} \in \bar{V}_i^f : \frac{1}{\lambda} \sum_{\bar{v} \in \mathcal{N}(\bar{u})} z_{i\bar{u}\bar{v}} \leq \sum_{\forall u \in V} y_{i\bar{u}u} \quad (11)$$

C. Objective Function

Following the problem statement, the objective function (12) has two components. The first component maximizes the number of re-embedded failed VLinks. The second one minimizes the total cost of provisioning bandwidth for re-embedding the failed VLinks on substrate paths. A weight factor w is multiplied to the second component to impose the necessary priority to the components of (12). The value of w is chosen to be a very small fraction so that it comes into effect only to break ties among multiple solutions that have the same value for the primary objective. In this way, w prioritizes the number of recovered VLinks over the cost of re-embedding.

$$\text{minimize} \left(|\bar{E}^f| - \sum_{\forall \bar{G}_i \in \bar{G}} \sum_{\forall (\bar{u},\bar{v}) \in \bar{E}_i^f} z_{i\bar{u}\bar{v}} \right) + w \left(\sum_{\forall \bar{G}_i \in \bar{G}} \sum_{\forall (\bar{u},\bar{v}) \in \bar{E}_i^f} \sum_{\forall (u,v) \in E} x_{uv}^{i\bar{u}\bar{v}} \times C_{uv} \times b_{i\bar{u}\bar{v}} \right) \quad (12)$$

D. Hardness of the Problem

The binary nature of the decision variables and the flow constraints of *Opt-ReNoVatE* prevent any VLink from being mapped to multiple substrate paths. This restricts the re-embedding of Independent VLinks to the $\mathcal{NP}$-hard *Multi-commodity Unsplittable Flow Problem* [34]. The VNode re-embedding follows from VLink re-embedding since there are no costs associated with VNodes. Therefore, the re-embedding of a VNode and its adjacent VLinks of a VN becomes the $\mathcal{NP}$-hard *Single-source Unsplittable Flow Problem* with unknown source [35]. When there are a batch of affected VNs, computing the best sequence of VNs from a combinatorial number of sequences to maximize the number of recovered VLinks makes the problem computationally intractable.

V. Heuristic Solution: *Fast-ReNoVatE*

Due to the intractability of *Opt-ReNoVatE*, we resort to a heuristic algorithm, *Fast-ReNoVatE*, to find feasible solutions in reasonable time. *Fast-ReNoVatE* re-embeds the failed VNodes and their Adjacent VLinks (Section V-A) and Independent VLinks (Section V-B) of the affected VNs efficiently.

A. Recovery of VNodes and Adjacent VLinks

The inputs to the VNode Recovery algorithm (Algorithm 1) comprise an SN G, a set of affected VNs $\bar{G}$ that are embedded on G, and a set of failed SNodes V^f and failed SLinks E^f. The algorithm initially sorts the affected VNs in $\bar{G}$ in increasing order of the total lost bandwidth in their Adjacent VLinks, and $\bar{\mathcal{G}}$ represents this sorted order. The total lost bandwidth of a VN, $\bar{G}_i \in \bar{\mathcal{G}}$ is computed as the summation of the bandwidth demands for all the failed VLinks in $\bar{\mathcal{E}}_i^f$ of $\bar{G}_i$. Intuitively, the algorithm proceeds to recover the VNs in $\bar{G}$ in the sorted order of $\bar{\mathcal{G}}$ to increase the number of recovered VLinks. For each VN $\bar{G}_i$, the algorithm tries to re-embed the failed VNode $\bar{u} \in \bar{V}_i^f$ and the VLinks adjacent to $\bar{u}$ *i.e.*, $\bar{\mathcal{E}}_i^f$ to an SNode present in the location constraint set of $\bar{u}$ *i.e.*, $L_i(\bar{u})$ and to substrate paths, respectively. To accomplish this, it iterates over all the SNodes $l \in L_i(\bar{u}) \setminus V^f$, and selects the SNode, $best_{\bar{u}}$ that maximizes the cardinality of the set of substrate paths, $\mathcal{P}_l$, computed for the VLinks in $\bar{\mathcal{E}}_i^f$. In case of a tie, the SNode with the lower cost of the paths in $\mathcal{P}_l$, denoted by M, is selected. Finally, the algorithm re-embeds $\bar{u}$ and the VLinks in $\bar{\mathcal{E}}_i^f$ to $best_{\bar{u}}$ and to the paths in M, respectively.

As discussed in Section IV-D, optimally computing the set of substrate paths $\mathcal{P}_l$ from an SNode $l \in L_i(\bar{u}) \setminus V^f$ for the VLinks in $\bar{\mathcal{E}}_i^f$ of a VN is $\mathcal{NP}$-hard. Majority of the VN embedding proposals aims to minimize the cost of embedding [2]. They would embed each VLink in $\bar{\mathcal{E}}_i^f$ one-by-one by adopting a minimum cost substrate path finding approach. However, in a bandwidth constrained scenario, a minimum cost path may contain some bottleneck SLinks. Allocating the bandwidth of these SLinks to a VLink may leave later VLinks unrecoverable. The objective of *ReNoVatE* is to maximize the number of recovered VLinks irrespective of the cost of recovery. Hence, our heuristic (Algorithm 2) simultaneously computes $\mathcal{P}_l$ for all the VLinks in $\bar{\mathcal{E}}_i^f$ to maximize the cardinality of $\mathcal{P}_l$. Algorithm 2 works by finding maximum flow from a source to a sink in a graph avoiding any bottleneck SLink. If we always send unit flow from a source to a sink in a graph, the paths carrying the maximum amount of flow will correspond to the maximum number of paths between the source and the sink without exceeding the capacity of the SLinks.

To implement this idea, Algorithm 2 first augments the SN graph G with a pseudo sink SNode, namely S. It then adds a pseudo SLink from each SNode that hosts a VNode, $\bar{v} \in \mathcal{N}(\bar{u})$ to S, where $\bar{u} \in \bar{V}_i^f$. The capacity of each augmented SLink is set to 1 to ensure the un-splittability of the substrate paths. Further, each bidirectional SLink in $E \setminus E^f$ is replaced with two unidirectional SLinks, and the capacity of these new SLinks are discretized according to an estimation function, $\frac{r_{uv}}{\max\limits_{\forall\{(\bar{u},\bar{v}) \in \bar{\mathcal{E}}_i^f\}} \{b_{i\bar{u}\bar{v}}\}}$. This stringent estimation ensures that each selected SLink can provide the bandwidth even for the VLink with the maximum demand among all the VLinks in $\bar{\mathcal{E}}_i^f$. Other estimation functions such as min or $average$ could be used allowing over-subscription of bandwidth for some of the VLinks. Fig. 2 illustrates the maximum flow realization for the recovery of VN $\bar{G}_1$ embedded on SN G according to Fig. 1. Upon the failure of SNode D, VNode a and VLinks ab and ac of $\bar{G}_1$ fail. Since $\mathcal{N}(a) = \{b, c\}$, $f(b) = C$, and $f(c) = G$, we add SLinks CS and GS to sink S with capacity 1, as shown by the dashed arrows in Fig. 2. This figure depicts the transformed SN after removing the failed SNode and SLinks, replacing each bidirectional SLink with two unidirectional SLinks, and estimating the capacity of these SLinks using $\frac{r_{uv}}{\max\{6,5\}}$.

After augmenting and initializing the capacity and flow of each SLink in G, Algorithm 2 proceeds with the steps of the *Edmonds-Karp* algorithm [36] for computing the maximum flow from each $l \in L_i(\bar{u}) \setminus V^f$ to S. We modify the *Edmonds-Karp* algorithm to compute the set of augmenting paths $\mathcal{P}$ from each l to S so that the sum of flows carried along these paths is the maximum. Each augmenting path $P \in \mathcal{P}$ will consume one

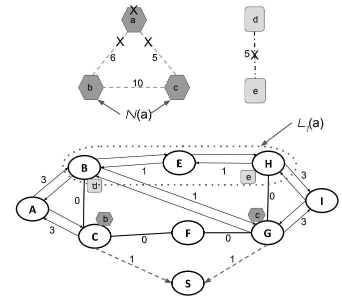


Fig. 2: Maxflow Realization

unit of bandwidth since we deliberately assign unit capacity to the pseudo SLinks incident to S. In addition, a path $P \in \mathcal{P}$ will contain an SNode from the set $\{f(\bar{v}) | \bar{v} \in \mathcal{N}(\bar{u})\}$, since S can only be reached through these SNodes. In the event that a new path P cancels the flow on any bottleneck SLink (u, v) of any existing path $P_i \in \mathcal{P}$, Algorithm 2 updates both P and P_i to exclude (u, v). Following these steps, the set of paths $\mathcal{P}$ from l to S is computed. However, the VLink re-mapping requires substrate paths from l to any SNode in $\{f(\bar{v}) | \bar{v} \in \mathcal{N}(\bar{u})\}$. Hence, the algorithm identifies the path P_i that contains any SNode in $\{f(\bar{v}) | \bar{v} \in \mathcal{N}(\bar{u})\}$, removes the SLink $(f(\bar{v}), S)$ from P_i, and indexes the modified P_i with the corresponding VLink $(\bar{u}, \bar{v})$. After modifying and indexing all the paths in $\mathcal{P}_l$, the algorithm returns $\mathcal{P}_l$.

To illustrate, we assume $L_1(a) = \{B, E, H\}$ in the example of Fig. 2. When $l = E$, Algorithm 2 computes the paths carrying the maximum flow from E to S through C and G in the transformed SN of Fig. 2. If the augmenting path finding step in the algorithm picks the shortest path $\{EB, BG, GS\}$ as P_1, the maximum flow is restricted to 1 by the bottleneck SLink BG. Hence, the algorithm computes a new path $P = \{EH, HI, IG, GB, BA, AC, CS\}$ in the residual network as defined in the *Edmonds-Karp* algorithm. Since P cancels the flow along the bottleneck SLink $BG \in P_1$, Algorithm 2 reorganizes the segments of P with P_1 to yield $\mathcal{P} = \{\{EB, BA, AC, CS\}, \{EH, HI, IG, GS\}\}$. Finally, the algorithm excludes CS and GS from the two paths and returns $\mathcal{P}_l[(a, b)] = \{EB, BA, AC\}$ and $\mathcal{P}_l[(a, c)] = \{EH, HI, IG\}$.

B. Recovery of Independent VLinks

As presented in Section IV-D, the re-embedding of Independent VLinks in $\bar{\mathrm{E}}^f$ is a variant of the $\mathcal{NP}$-hard *Multi-commodity Unsplittable Flow Problem*. The heuristic described in Section V-A is not applicable to this problem since both the endpoints of a VLink in $\bar{\mathrm{E}}^f$ are mapped to some SNodes, and finding maximum paths may yield invalid paths between the wrong pair of SNodes. Hence, we propose a greedy strategy (Algorithm 3) based on computing the minimum cost path. Algorithm 3 first sorts the VLinks in $\bar{\mathrm{E}}^f$ in increasing order of their bandwidth demand, and $\bar{\mathscr{E}}^f$ represents this order. According to this order, the algorithm computes alternate substrate path for each VLink in $(\bar{x}, \bar{y}) \in \bar{\mathscr{E}}^f$ in the subgraph induced by excluding the failed SNode and SLinks from G. For a particular VLink $(\bar{x}, \bar{y})$, the algorithm finds the minimum cost path from $f(\bar{x})$ to $f(\bar{y})$ using the procedure

Algorithm 1 VNodes Recovery

function VNODES RECOVERY($G, \bar{G}, V^f, E^f$)
  $\bar{\mathcal{G}} \leftarrow$ Sort $\bar{G}_i \in \bar{G}$ in increasing order of $\sum_{\forall(\bar{u},\bar{v}) \in \bar{\mathcal{E}}_i^f} b_{i\bar{u}\bar{v}}$
  $max_{\mathcal{R}} \leftarrow 0$
  for all $\bar{G}_i \in \bar{\mathcal{G}}$ **do**
    $\bar{u} \leftarrow$ failed virtual node in $\bar{G}_i$, $best_{\bar{u}} \leftarrow NIL$
    for all $l \in L_i(\bar{u}) \setminus V^f$ **do**
      $\mathcal{P}_l \leftarrow Max-Paths(G, \bar{G}_i, \bar{u}, l, \bar{\mathcal{E}}_i^f, V^f, E^f)$
      if $\mathcal{R}(\mathcal{P}_l) > max_{\mathcal{R}}$ **or** ($\mathcal{R}(\mathcal{P}_l) = max_{\mathcal{R}}$ **and** $\mathcal{C}(\mathcal{P}_l) \leq \mathcal{C}(M)$) **then**
        $M \leftarrow \mathcal{P}_l$, $best_{\bar{u}} \leftarrow l$
      end if
    end for
    if $best_{\bar{u}} \neq NIL$ **then**
      map $\bar{u}$ to $best_{\bar{u}}$
      $\forall(\bar{u},\bar{v}) \in \mathcal{E}_i^f$: map $(\bar{u},\bar{v})$ to $M[(\bar{u},\bar{v})]$
    end if
  end for
end function

Algorithm 2 Max-Paths

function MAX-PATHS($G, \bar{G}_i, \bar{u}, l, \bar{\mathcal{E}}_i^f, V^f, E^f$)
  $V \leftarrow V \setminus V^f \cup \{S\}$, $E \leftarrow E \setminus E^f$
  $\forall(\bar{u},\bar{v}) \in \bar{\mathcal{E}}_i^f$: $E \leftarrow E \setminus E^f \cup \{(f(\bar{v}), S)\}$
  $max\_bw \leftarrow \max_{\forall(\bar{u},\bar{v}) \in \bar{\mathcal{E}}_i^f} \{b_{i\bar{u}\bar{v}}\}$
  $\forall(u,v) \in E$ s.t. $u = S$ or $v = S$:
    $flow_{uv} \leftarrow 0$, $c_{uv} \leftarrow 1$
  $\forall(u,v) \in E$ s.t. $r_{uv} > 0$ and $u \neq S$ and $v \neq S$:
    $flow_{uv} \leftarrow flow_{vu} \leftarrow 0$
    $c_{uv} \leftarrow c_{vu} \leftarrow \lfloor \frac{r_{uv}}{max\_bw} \rfloor$
  $G_r \leftarrow G$, $\mathcal{P} \leftarrow \phi$, $\mathcal{P}_l \leftarrow \phi$
  while $\exists$ augmenting path P from l to S in G_r **do**
    $C_f(P) \leftarrow \min_{\forall(u,v) \in P} \{c_{uv}\}$
    Augment $C_f(P)$ units flow in G_r along P
    Update residual capacity in G_r along P
    Remove links with residual capacity ≤ 0
    for all $P_i \in \mathcal{P}$ **do**
      if P cancels the flow along $(u,v) \in P_i$ **then**
        $\tau \leftarrow$ Sub paths in $P_i \setminus (u,v)$
        $\nu \leftarrow$ Sub paths in $P \setminus (v,u)$
        $P \leftarrow$ Path formed by segments in τ and ν
        $P_i \leftarrow$ Path formed by segments in τ and ν
      end if
    end for
    $\mathcal{P} \leftarrow \mathcal{P} \cup P$
  end while
  $\forall P_i \in \mathcal{P}$, $\forall(\bar{u},\bar{v}) \in \bar{\mathcal{E}}_i^f$:
    $P_i \leftarrow$ Path containing $(f(\bar{v}), S)$
    $\mathcal{P}_l[(\bar{u},\bar{v})] \leftarrow P_i \setminus (f(\bar{v}), S)$
return $\mathcal{P}_l$
end function

Algorithm 3 VLinks-Recovery

function VLINKS-RECOVERY($G, \bar{\mathtt{E}}^f, V^f, E^f$)
  $\mathcal{P} \leftarrow \phi$
  $V \leftarrow V \setminus V^f$, $E \leftarrow E \setminus E^f$
  $\bar{\mathscr{E}}^f \leftarrow$ Sort $(\bar{x},\bar{y}) \in \bar{\mathtt{E}}^f$ in increasing order of $b_{i\bar{x}\bar{y}}$
  for all $(\bar{x},\bar{y}) \in \bar{\mathscr{E}}^f$ **do**
    $\mathcal{P}[(\bar{x},\bar{y})] \leftarrow$ MCP($G, f(\bar{x}), f(\bar{y}), b_{i\bar{x}\bar{y}}$)
    map $(\bar{x},\bar{y})$ to $\mathcal{P}[(\bar{x},\bar{y})]$
  end for
end function

MCP, and adds it to the set $\mathcal{P}$. The procedure MCP uses a modified version of *Dijkstra's shortest path* algorithm [37] to take into account SLink residual capacity and VLink demand while computing the minimum cost path.

To illustrate, Fig. 2 depicts the recovery of Independent VLink de of VN $\bar{G}_2$, embedded on SN G according to Fig. 1. Algorithm 3 should find alternate substrate paths between B and H for VLink de. The max flow based heuristic may return substrate path between B and G (or, between H and C), and is thus inappropriate. Hence, Algorithm 3 recovers de through the minimum cost path $\{BG, GH\}$ that has sufficient bandwidth to satisfy the demand of de.

C. Running Time Analysis

The most expensive step in Algorithm 1 is the computation of the maximum paths using Algorithm 2. The core part of Algorithm 2 follows the steps of *Edmonds-Karp* Algorithm. If *Edmonds-Karp* Algorithm computes augmenting paths using Breadth-first Search, it runs in $O(|V||E|^2)$ time. Since Algorithm 2 is invoked $|L(\bar{u})|$ times for recovering a VN, and there are $|\bar{V}^f|$ number of affected VNs, total running time yields $O(|L(\bar{u})||\bar{V}^f||V||E|^2)$. In contrast, Algorithm 3 invokes *Dijkstra's shortest path* algorithm for each VLink in $\bar{\mathtt{E}}^f$. Since *Dijkstra's shortest path* algorithm runs in $O(|E|+|V|\log|V|)$ time, Algorithm 3 requires $O(|\bar{\mathtt{E}}^f|(|E|+|V|\log|V|))$ time.

VI. EVALUATION

In this section, we present our evaluation results found through extensive simulation.

A. Compared Algorithms

We compare *Opt-ReNoVatE* and *Fast-ReNoVatE* with an implementation of dynamic recovery approach in [9], called *Dyn-Recovery*. For fair comparison with our algorithms, we exclude the last step of re-embedding the entire VN from the implementation of *Dyn-Recovery* in the event of resource inadequacy. Although we evaluate all three algorithms in small size networks, we cannot evaluate *Opt-ReNoVatE* for large networks because of the inherent complexity of ILP-solvers. Alternatively, we present *Fast-ReNoVatE*'s baseline performance assuming the SLinks of an SN have infinite bandwidth, and refer to it as *Fast-ReNoVatE-INF*.

B. Performance Metrics

1) Recovery Efficiency: The fraction of successfully recovered VLinks over all failed VLinks expressed in percentage.

Scenario	Figure	SNodes	SLinks	VNodes/VN	VLinks/VN	VNs	SN Utilization	VLink BW	Total Failed VLinks
Small Scale	Fig. 3, Fig. 4(a)	50	90	5	8	10-32	$\sim$20% - $\sim$75%	$\sim$10% of SLink	250 - 866
	Fig. 4(b)	20-65	37-118	5	8	8-19	$\sim$53%	$\sim$15% of SLink	178 - 519
Large Scale	Fig. 5	1000	1798	3-15	2-30	93-563	$\sim$20% - $\sim$80%	$\sim$10% of SLink	4712 - 13546

TABLE I: Summary of Simulation Parameters

2) Recovery Cost: The average cost of provisioning bandwidth along a substrate path times the cost of allocating one unit bandwidth for re-embedding each failed VLink.

3) Execution Time: The time required for an algorithm to find the solution for all the VNs affected by an SNode failure.

C. Simulation Setup

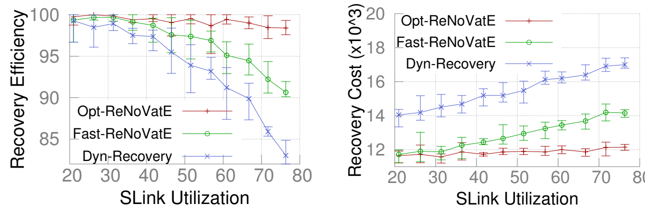
We implement *Opt-ReNoVatE* using IBM ILOG CPLEX C++ library; and *Fast-ReNoVatE* and *Dyn-Recovery* [9] using C++. We evaluate the algorithms on both small and large scale networks as summarized in Table I. For each problem instance in this table, we generate 5 random SNs by taking the number of SNodes and link-to-node ratio as inputs, and randomly creating SLinks between SNodes. Then, we generate a number of VNs for each SN, and embed the VNs on the SN following the procedure described in Section VI-D. We select a random SNode and its one hop neighbor SNodes as the location constraint set of a VNode. Afterwards, we simulate an SNode failure by removing each SNode in the SN one-by-one and execute the recovery algorithms being evaluated on the affected VNs. Finally, we measure the performance metrics of the compared algorithms by taking averages across all SNode failures for all 5 similar problem instances. The simulations are performed on a server with 2 Intel Xeon E5-2650 (8 cores @ 2.0GHz, each) processors and 256GB of RAM.

D. VN Embedding Method

VN generation and embedding are done simultaneously to overcome the issue of creating VNs that do not have feasible embeddings. The embedding of VNs on an SN is done in a random but load-balanced manner to achieve uniform distribution of VNs across the SN. Achieving higher utilization (beyond 70%) of SN requires a denser embedding which is done by relaxing the load-balancing criteria. The embedding of a VN starts by randomly selecting a source SNode that has free capacity for a VNode. This forms the first VNode (source VNode) of the VN currently being embedded. For each randomly assigned neighbor of this VNode, a random destination SNode within several hop distances from the source SNode is selected as a candidate for embedding the subsequent VNode (destination VNode). The source and destination VNodes are joined using a VLink embedded on the shortest path between source and destination SNodes. If the path is not found, the destination VNode is moved to a different destination SNode. These steps are repeated until all VNodes are embedded, generating both the VN and its embedding on the SN.

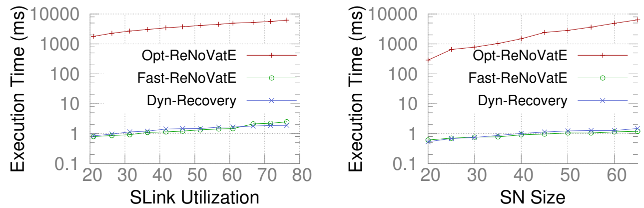
E. Small Scale Evaluations

In small scale settings, we first evaluate *Opt-ReNoVatE*, *Fast-ReNoVatE*, and *Dyn-Recovery* by varying number of embedded VNs to achieve different SLink utilizations. Fig. 3(a) shows that the recovery efficiencies of all three approaches decrease with the increase in SLink utilization. As the utilization increases, more VNs are affected by the SNode failure, and less bandwidth is left for recovery resulting in the gradual decrease in the number of recovered VLinks. Further, the impact of utilization is more profound in the higher utilization cases due to the lack of load balanced embedding. In the higher utilized cases, the recovery efficiencies of *Fast-ReNoVatE* is $\sim$ 6% better than those of *Dyn-Recovery* and $\sim$ 3% worse than those of *Opt-ReNoVatE*.



(a) Recovery Performance (b) Cost Analysis

Fig. 3: Performance of small scale testcases



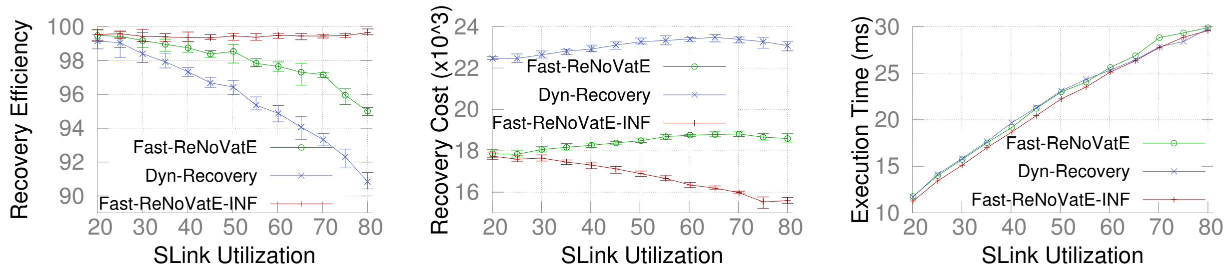
(a) Fixed SN (b) Fixed Utilization

Fig. 4: Scalability Analysis

We show the recovery cost in Fig. 3(b) against different SLink utilizations. In higher utilization cases, more SLinks become saturated in terms of bandwidth, and all the algorithms have to select longer substrate paths to recover resulting in increased costs. Further, the costs of *Opt-ReNoVatE* are the least among the three, since it selects the most suitable paths through exhaustive search. In contrast, *Fast-ReNoVatE* iterates over all the affected VNs and the independent VLinks in a greedy manner, sometimes preferring less suitable paths resulting in more costs than *Opt-ReNoVatE*. Finally, *Dyn-Recovery* does not consider the cost of a path while recovering a VLink. It may select a longer path than that selected by *Fast-ReNoVatE* leading to a much larger cost than *Fast-ReNoVatE*. On average, *Fast-ReNoVatE* incurs $\sim$ 7% more cost than *Opt-ReNoVatE* and $\sim$ 20% less cost than *Dyn-Recovery*.

F. Scalability Analysis

To demonstrate the scalability of the compared algorithms, Fig. 4 shows their execution times on the same problem instances. Fig. 4(a) presents the execution times by varying utilization on a fixed SN, whereas Fig. 4(b) presents the



(a) Recovery Performance (b) Cost Analysis (c) Timing Performance

Fig. 5: Performance of large scale testcases

execution times in SNs with varying sizes while keeping the utilization fixed at $\sim 53\%$. According to Table I, both problem size and the total number of failed VLinks increase with the increase in utilization and SN size. Consequently, the execution times grow for all the approaches, however, the increase is exponential for *Opt-ReNoVatE*. As it turns out, *Fast-ReNoVatE* and *Dyn-Recovery* take less than $3ms$ and $2ms$, respectively, in the highest utilized SN, whereas *Opt-ReNoVatE* takes $\sim 6s$ on the same problem instance. The slightly higher execution times of *Fast-ReNoVatE* compared to *Dyn-Recovery* are due to the extra iterations of *Fast-ReNoVatE* to avoid bottleneck SLinks to achieve higher recovery efficiencies. Despite that, *Fast-ReNoVatE* is 400x$-$2000x faster than *Opt-ReNoVatE* depending on problem instance. Finally, *Opt-ReNoVatE* could not scale to more than 65 SNode SNs.

G. Large Scale Evaluations

Fig. 5 shows the performance of the compared algorithms at large scale by varying SLink utilization. Fig. 5(a) shows that the recovery efficiencies of *Fast-ReNoVatE* are $\sim 6\%$ better than those of *Dyn-Recovery* and $\sim 2.5\%$ worse than those of *Fast-ReNoVatE-INF*. Similar to the small scale results, recovery efficiencies of *Fast-ReNoVatE* and *Dyn-Recovery* decrease with the increase in SLink utilization whereas recovery efficiencies remain almost the same for *Fast-ReNoVatE-INF*. The near constant recovery efficiencies of *Fast-ReNoVatE-INF* confirms that the reason of failing to recover is the insufficiency of bandwidth in SLinks. In other words, if there were adequate bandwidth in the SLinks, *Fast-ReNoVatE* could recover $\sim 99\%$ of the failed VLinks. The very small percentage of un-recovered VLinks of *Fast-ReNoVatE-INF* is due to the partitioning in the SN caused by the SNode failure. In these cases, it is not possible to recover a failed VLink even if there is sufficient bandwidth.

Fig. 5(b) shows the recovery cost in large networks. For the same reasons discussed in the case of small scale networks, *Dyn-Recovery* incurs the largest amount of cost, and the costs of *Fast-ReNoVatE* and *Dyn-Recovery* rise with the increase in SLink utilization. In contrast, there is no effect of residual bandwidth in finding an alternate path in *Fast-ReNoVatE-INF*, and it can select the minimum cost path resulting in the least costs. This is true for *Fast-ReNoVatE* in very low utilized SNs. Further, the two end nodes of the failed VLink are embedded closely to each other in a highly utilized SN. Hence, *Fast-ReNoVatE-INF* recovers the VLink with shorter path leading to the decrease of costs with the increase in SLink utilization. The counterintuitive behavior of Fig. 5(b) from SLink utilization of 70 to 80 is due to relaxing the load-balancing criteria as explained in Section VI-D. The denser embedding to achieve higher utilization maps the VNodes of a VN closer to one another, thus requiring lower recovery cost than a sparser one. Finally, Fig. 5(c) shows that *Fast-ReNoVatE* has similar timing performance to *Dyn-Recovery*, and both are able to find a solution in less than 30ms even in the highest utilized SN.

VII. Conclusion

In this paper, we have addressed the problem of reactive recovery of a batch of affected VNs, resulting from a single substrate node failure. We have formulated the problem as an ILP model, *Opt-ReNoVatE* and presented an efficient heuristic algorithm, *Fast-ReNoVatE*, to tackle the computational complexity. We have evaluated *Fast-ReNoVatE*, *Opt-ReNoVatE*, and a state-of-the-art solution, *Dyn-Recovery* in both small and large scale networks. Evaluation results demonstrate that *Fast-ReNoVatE* can recover $\sim 6\%$ more VLinks than *Dyn-Recovery* and $\sim 3\%$ less VLinks than *Opt-ReNoVatE* in high utilization scenarios. In terms of scalability, *Fast-ReNoVatE* is several orders of magnitude faster than *Opt-ReNoVatE*, and has comparable performance with *Dyn-Recovery*. In large scale networks, we have compared *Fast-ReNoVatE* with *Dyn-Recovery* and a baseline case of infinite bandwidth SN. These results demonstrate that *Fast-ReNoVatE* is able to recover $\sim 99\%$ of the failed VNs if the SN has adequate residual capacity, and has similar timing performance to *Dyn-Recovery*.

In the future, we plan to extend this work to accommodate recovery with over-subscribed bandwidth allocation for the failed VLinks. Another possible research direction is to prioritize the affected VNs based on SLA requirements, impacts of failure or profits, and adhere to that priority during recovery.

VIII. Acknowledgment

We would like to thank our shepherd Luciano Paschoal Gaspary and the anonymous CNSM 2016 reviewers for their valuable feedback and Nicole Keshav for writing suggestions that helped improve the quality of this paper. This work was supported in part by Huawei Technologies and in part by an NSERC Collaborative Research and Development Grant. Additionally, this work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

References

[1] N. M. M. K. Chowdhury and R. Boutaba, "A Survey of Network Virtualization," *Computer Networks*, vol. 54, pp. 862–876, Apr 2010.

[2] A. Fischer, J. F. Botero, M. T. Beck, H. D. Meer, and X. Hesselbach, "Virtual Network Embedding: A Survey," *IEEE Communications Surveys and Tutorials*, vol. 15, pp. 1888–1906, Feb 2013.

[3] P. Gill, N. Jain, and N. Nagappan, "Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications," in *ACM SIGCOMM*, vol. 41, Aug 2011, pp. 350–361.

[4] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C. Chuah, and C. Diot, "Characterization of Failures in an IP Backbone," in *INFOCOM*, vol. 4, Mar 2004, pp. 123–133.

[5] S. Herker, A. Khan, and X. An, "Survey on Survivable Virtual Network Embedding Problem and Solutions," in *ICNS*, Portugal, 2013, pp. 99–104.

[6] T. Guo, N. Wang, K. Moessner, and R. Tafazolli, "Shared Backup Network Provision for Virtual Network Embedding," in *IEEE ICC*, Kyoto, Japan, Jun 2011, pp. 1–5.

[7] M. R. Rahman and R. Boutaba, "SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization," *IEEE Transactions on Network and Service Management*, vol. 10, pp. 105–118, Feb 2013.

[8] X. Chang, J. Muppala, B. Wang, J. Liu, and L. Sun, "Migration cost aware virtual network re-embedding in presence of resource failures," in *Networks (ICON), 2012 18th IEEE International Conference on*, Dec 2012, pp. 24–29.

[9] B. LU, H. Tao, S. Xiao-chuan, C. Jian-ya, and L. Yun-jie, "Dynamic Recovery for Survivable Virtual Network Embedding," *The Journal of China Universities of Posts and Telecommunications*, vol. 21, pp. 77–84, Jun 2014.

[10] B. Guo, C. Qiao, J. Wang, H. Yu, Y. Zuo, J. Li, Z. Chen, and Y. He, "Survivable virtual network design and embedding to survive a facility node failure," *Lightwave Technology, Journal of*, vol. 32, no. 3, pp. 483–493, 2014.

[11] Q. Hu, Y. Wang, and X. Cao, "Survivable network virtualization for single facility node failure: A network flow perspective," *Optical Switching and Networking*, vol. 10, no. 4, pp. 406–415, 2013.

[12] H. Jiang, L. Gong, and Z. Zuqing, "Efficient joint approaches for location-constrained survivable virtual network embedding," in *Global Communications Conference (GLOBECOM), 2014 IEEE*. IEEE, 2014, pp. 1810–1815.

[13] H. Yu, V. Anand, C. Qiao, and G. Sun, "Cost efficient design of survivable virtual infrastructure to recover from facility node failures," in *Communications (ICC), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1–6.

[14] O. Soualah, I. Fajjari, N. Aitsaadi, and A. Mellouk, "A batch approach for a survivable virtual network embedding based on monte-carlo tree search," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*. IEEE, 2015, pp. 36–43.

[15] Z. Cai, F. Liu, N. Xiao, Q. Liu, and Z. Wang, "Virtual network embedding for evolving networks," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–5.

[16] I. Houidi, W. Louati, D. Zeghlache, P. Papadimitriou, and L. Mathy, "Adaptive virtual network provisioning," in *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*. ACM, 2010, pp. 41–48.

[17] M. M. A. Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "Simple: Survivability in multi-path link embedding," in *Network and Service Management (CNSM), 2015 11th International Conference on*. IEEE, 2015, pp. 210–218.

[18] M. M. A. Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "Multi-path link embedding for survivability in virtual networks," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 253–266, June 2016.

[19] N. Shahriar, R. Ahmed, S. R. Chowdhury, M. M. A. Khan, R. Boutaba, J. Mitra, and F. Zeng, "Connectivity-aware virtual network embedding," in *The 15th IFIP Networking Conference (NETWORKING 2016)*, Vienna, Austria, 2016, pp. 46–54.

[20] Y. Chen, J. Li, T. Wo, C. Hu, and W. Liu, "Resilient virtual network service provision in network virtualization environments," in *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*. IEEE, 2010, pp. 51–58.

[21] H. Yu, C. Qiao, V. Anand, X. Liu, H. Di, and G. Sun, "Survivable virtual infrastructure mapping in a federated computing and networking system under single regional failures," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*. IEEE, 2010, pp. 1–6.

[22] Q. Hu, Y. Wang, and X. Cao, "Location-constrained survivable network virtualization," in *Sarnoff Symposium (SARNOFF), 2012 35th IEEE*. IEEE, 2012, pp. 1–5.

[23] S. R. Chowdhury, R. Ahmed, M. M. A. Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng, "Protecting virtual networks with drone," in *IEEE/IFIP Network Operations and Management Symposium 2016*, Istanbul, Turkey, April 2016.

[24] S. R. Chowdhury, R. Ahmed, M. M. A. KHAN, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng, "Dedicated protection for survivable virtual network embedding," *IEEE Transactions on Network and Service Management*, vol. PP, no. 99, pp. 1–1, 2016.

[25] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," in *NSDI*, 2010, pp. 17–32.

[26] N. Bansal, R. Bhagwan, N. Jain, Y. Park, D. Turaga, and C. Venkatramani, "Towards optimal resource allocation in partial fault-tolerant applications," in *IEEE INFOCOM*, 2008.

[27] W.-L. Yeow, C. Westphal, and U. C. Kozat, "Designing and embedding reliable virtual infrastructures," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 57–64, 2011.

[28] J. Xu, J. Tang, K. Kwiat, W. Zhang, and G. Xue, "Survivable virtual infrastructure mapping in virtualized data centers," in *IEEE CLOUD*, 2012, pp. 196–203.

[29] P. Bodik, I. Menache, M. Chowdhury, P. Mani, D. A. Maltz, and I. Stoica, "Surviving failures in bandwidth-constrained datacenters," in *ACM SIGCOMM*, 2012, pp. 431–442.

[30] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 289–297.

[31] M. R. Rahman, I. Aib, and R. Boutaba, "Survivable virtual network embedding," in *NETWORKING 2010*. Springer, 2010, pp. 40–52.

[32] R. R. Oliveira, D. S. Marcon, L. R. Bays, M. C. Neves, L. S. Buriol, L. P. Gaspary, and M. P. Barcellos, "No more backups: Toward efficient embedding of survivable virtual networks," in *Communications (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 2128–2132.

[33] M. Soares and E. Madeira, "A multi-agent architecture for autonomic management of virtual networks," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, April 2012, pp. 1183–1186.

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

[35] Y. Dinitz, N. Garg, and M. X. Goemans, "On the single-source unsplittable flow problem," *Combinatorica*, vol. 19, no. 1, pp. 17–41, 1999.

[36] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *J. ACM*, vol. 19, no. 2, pp. 248–264, Apr. 1972. [Online]. Available: http://doi.acm.org/10.1145/321694.321699

[37] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.