Connectivity-aware Virtual Network Embedding

Nashid Shahriar*, Reaz Ahmed*, Shihabur Rahman Chowdhury*, Md. Mashrur Alam Khan*, Raouf Boutaba*, Jeebak Mitra†, and Feng Zeng†
*David R. Cheriton School of Computer Science, University of Waterloo
{nshahria | r5ahmed | sr2chowdhury | mmalamkh | rboutaba}@uwaterloo.ca
†Huawei Technologies
{jeebak.mitra | zengfeng137140}@huawei.com

***Abstract*—The problem of ensuring virtual network (VN) connectivity in presence of multiple link failures in the substrate network (SN) is not well investigated in Network Virtualization (NV) literature. We name this problem as Co*nnectivity-aware* Vi*rtual* N*etwork* E*mbedding* (*CoViNE*). Solving *CoViNE* will enable a VN operator to perform failure recovery without depending on the SN provider, similar to the IP restoration mechanisms in IP-over-WDM networks. There are two steps in solving *CoViNE*: i) finding the virtual links that should be embedded disjointly, and ii) finding a substrate resource efficient embedding that ensures the virtual link disjointness constraint. We present two solutions to the *CoViNE* problem. The first solution uses a heuristic to compute the disjointness constraint, while an optimization model is used for VN embedding. The second solution, in contrast, uses heuristic for both the steps, and thus can solve larger instances of the problem. We compare our solutions with a cut set based approach that ensures VN connectivity for a single substrate link failure. Evaluation results show that our heuristics allocate $\sim$15% extra resources on average compared to the cut set based optimal solution, and executes two to three orders of magnitude faster on the same problem instances.**

I. Introduction

Perceived as a key enabling technology for the future Internet, Network Virtualization (NV) offers efficient resource sharing by embedding multiple Virtual Networks (VNs) on a single Substrate Network (SN). One of the major challenges in NV is VN embedding (VNE) [1], *i.e.*, to find a mapping of the virtual nodes and links onto substrate nodes and paths, respectively without violating physical resource constraints.

Substrate resources may fail. Surviving failures is of paramount importance, since a single failure in SN may result into multiple failures in the embedded VNs. Finding a VN embedding that can survive failures in SN is known as the Survivable Virtual Network Embedding (SVNE) problem [2]. Majority of the works on SVNE focus on link failures, as they occur more frequently than node failures [3]. SVNE approaches, in general, allocate redundant bandwidth for each (or selected) virtual link(s), either proactively while computing the embedding or reactively after a failure occurs [4].

In this paper, we focus on a different form of survivability than traditional SVNE, which we call Co*nnectivty-aware* Vi*rtual* N*etwork* E*mbedding* (*CoViNE*). Our goal is to find a VN embedding that can ensure connectivity in a VN topology in presence of multiple substrate link failures. In contrast, SVNE approaches focus on guaranteeing virtual link demand in presence of failure(s). SVNE approaches assume that SN-providers hide physical failures by over-provisioning a fraction of each virtual link's bandwidth, which in turn incurs additional cost to VN-operators. In contrast, we ensure VN connectivity, which is a weaker form of survivability incurring lesser resource overhead and reduced cost of leasing. We intend to empower a VN-operator to handle link failures according to its internal policy, e.g., customer priority. A VN-operator can plan and over-provision different amounts of bandwidth in each virtual link to handle failures according to its needs, instead of blindly relying on the SN-provider that over provisions fixed bandwidth for each virtual link as in SVNE approaches. Connectivity aware embedding will enable a VN-operator to reroute traffic on failed virtual links, which can be done using any IP link restoration protocol.

Although our focus is NV, the *CoViNE* problem is equally applicable in IP-over-Wavelength-division multiplexing (WDM) domain. The problem of ensuring IP layer connectivity in presence of a single WDM link failure is known as *link survivable mapping*. Two variations of the problem have been studied in IP-over-WDM literature [5]: i) *weakly link survivable mapping* (WLSM) ensures IP-layer connectivity; ii) *strong link survivable* mapping guarantees both connectivity and bandwidth of the failed IP link(s) in presence of a single WDM link failure. WLSM, which considers single link failure, is merely a special case of *CoViNE*.

Despite being neglected in the literature, we focus on multiple (more specifically up to double) link failures, since it is not a rare event in large transport networks. First, repairing a failed link (e.g., due to fiber cut) can take long time [3]. Chances of a second link failure is not negligible given the high Mean-Time-to-Repair (MTTR). Second, some inter-datacenter links destined to different places may be physically routed together for some distance, and a backhaul failure may cause multiple physical links to fail [6]. It can be derived from the statistics in [7] that $\sim$12% of the failures in inter-datacenter transport networks are double link failures in SN.

There are two conditions for surviving multiple (say, k) substrate link failures: i) the VN topology must be $k+1$ edge connected, and ii) the embedding algorithm must ensure at least $k+1$ edge-disjoint paths in SN between every pair of virtual nodes. The first condition can be satisfied by augmenting the VN with new links [8]. A naive way to satisfy the second condition is to embed all the virtual links of a $k+1$ edge connected VN onto disjoint paths in the SN. However, this is an NP-complete problem [9], and imposes an

ISBN 978-3-901882-83-8

unsatisfiable number of disjointness constraints. Existing cut set based approaches suffer from poor scalability [10], [11]. Furthermore, most of the heuristic schemes either focus on single link failure [12], [13], or fail to deal with arbitrary VN topologies [14]. The approach in [8] proposes a generalized solution for multiple link failures. This solution requires a large number of virtual links to be embedded disjointly, hence, is not resource efficient. Therefore, we propose novel solutions that embed arbitrary VN topologies with near-optimal disjointness constraints to survive in presence of multiple link failures. Our solutions augment a VN with minimal number of virtual links while preserving the topological structure of the VN. The major contributions of this paper are:

1) We explore an alternate survivability model, *CoViNE*, requiring significantly less backup resources than traditional survivability approaches in SVNE literature.
2) We present two generalized solutions to the *CoViNE* problem dealing with multiple substrate link failures. The first solution builds upon heuristic for augmenting the VN and computing the virtual links that should be embedded disjointly, and an optimization model for VN embedding adhering to the disjointness requirement. The second solution, in contrast, uses heuristic for both the steps and thus can solve larger instances of the problem.
3) Through extensive simulations, we evaluate the optimality and the time-complexity of the proposed solutions. In addition, we show how *CoViNE* can reduce the impact of failure in single and double link failure scenarios.

The rest of this paper is organized as follows. We present the related literature in Section II. In Section III, we present the system model and problem statement. A theoretical foundation of *CoViNE* is laid in Section IV. A heuristic algorithm for computing disjointness constraint is presented in Section V. An optimization model and a heuristic algorithm for VN embedding are presented in Section VI and Section VII, respectively. We present the simulation setup and evaluation results in Section VIII. Finally, we conclude with future research directions in Section IX.

II. Related Works

A number of approaches exist in the literature for survivable VN Embedding. However, these approaches mostly focus on ensuring the same end-to-end QoS guarantee after single SLink failure [2], [15], [16], [17]. A number of research works in IP-over-WDM literature focus on ensuring connectivity of IP links under WDM link failures [10], [13], [14]. In this section, we briefly describe the most prominent approaches in literature, and contrast them with our solutions for *CoViNE*.

Modiano *et al.* [10] presented an ILP formulation for survivable VLink routing on WDM SN in presence of single SLink failure. Their formulation explores exponential number of cut sets in the VN and routes all the VLinks of a cut set on disjoint WDM paths. Todimala *et al.* [11] improved the ILP formulation by identifying polynomial number of primary cuts in a VN. The authors in [18] extended the Max-flow min-cut theorem for multi-layer networks and proposed approximation algorithms for VLink routing, while maximizing the minimum cross layer cut. A major drawback of these approaches is that they do not scale well with network size, because of the inherent complexity of LP-solvers.

Several heuristic based approaches have been proposed for survivable VN embedding in large networks. Kurant *et al.* [14] proposed SMART, a framework for finding survivable mapping of a VN by repeatedly picking cycles of a VN and finding survivable mappings for the cycles. SMART can ensure connectivity under double SLink failures for VNs having a few special structures, hence, has limited applicability. An extension to SMART has been proposed by [12] that exploits the duality between circuits and cuts in the VN. Zhou *et al.* [13] proposed an algorithm that identifies a set of spanning trees of the VN and computes a shortest-path based routing of the VLinks such that at least one of the spanning trees survives after an SLink failure. In contrast, our solution is generic, *i.e.*, does not assume any specific property of the VN and SN, and can ensure connectivity in presence of multiple SLink failures.

Several research works from IP-over-WDM literature ensure survivability through IP link augmentation [12], [13], [19], [20]. However, all of these works focus on the single failure resiliency and cannot be generalized to multiple failures. In contrast, Thulasiraman *et al.* propose an augmentation strategy for ensuring survivability under k SLink failures [8]. They propose to augment VLinks until a complete subgraph of $k+1$ VNodes is constructed and the remaining VNodes are $k+1$ edge connected to the subgraph. Their solution maps any k of the VLinks incident to a VNode onto disjoint paths. This approach requires higher number of VLinks to be augmented and more disjointness constraints on the SN than our approach.

III. Preliminaries

The subsequent sections build upon the background, definitions, and assumptions presented in this section.

A. System Model

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

2) Virtual Network: A VN is represented as an undirected graph $\bar{G} = (\bar{V}, \bar{E})$, where $\bar{V}$ and $\bar{E}$ denote the set of Virtual Nodes (VNodes) and Virtual Links (VLinks), respectively. The neighbors of a VNode $\bar{v} \in \bar{V}$ is denoted by $\mathcal{N}(\bar{v})$. Each VLink $(\bar{u}, \bar{v}) \in \bar{E}$ has bandwidth requirement $b_{\bar{u}\bar{v}}$. Each VNode $\bar{u} \in \bar{V}$ has a location constraint, $L(\bar{u}) \subseteq V$, that denotes the set of SNodes where $\bar{u}$ can be embedded.

B. Design Choices

The first condition for surviving k SLink failures is that a VN must be $k+1$ edge connected. However, if the input VN $\bar{G}$ does not have such connectivity, we will need to augment

TABLE I
NOTATION TABLE

$G = (V, E)$	Substrate Network (SN)
$\bar{G} = (\bar{V}, \bar{E})$	Virtual Network (VN)
$\hat{G} = (\hat{V}, \hat{E})$	k-protected VN
$\hat{G}_k = (\hat{V}_k, \hat{E}_k)$	k-protected component of a VN $\bar{G}$
$\hat{G}_k \odot \hat{v}$	An expansion of $\hat{G}_k$ towards $\hat{v}$
$\chi^{\hat{u}\hat{v}}$	Conflicting set of a VLink $(\hat{u}, \hat{v})$
$\chi^{\hat{u}\hat{v}}_{\odot}$	Conflicting set of a VLink $(\hat{u}, \hat{v})$ during expansion
$\chi^{\hat{G}}$	Conflicting set of a VN $\hat{G}$
Q^{uv}	A path between SNodes u and v in G
$P^{\hat{u}\hat{v}}$	A path between VNodes $\hat{u}$ and $\hat{v}$ in $\hat{G}$
$\mathscr{P}^{\hat{u}\hat{v}}$	Set of edge-disjoint paths between $\hat{u}$ and $\hat{v}$ in $\hat{G}$
$\mathrm{p}_i^{\hat{u}\hat{v}}$	ith edge-disjoint shortest path from $\hat{u}$ to $\hat{v}$ in $\hat{G}$
$\mathcal{P}^{\hat{G}_k\hat{v}}$	Set of edge-disjoint shortest paths from $\hat{v}$ to $\hat{G}_k$

$\bar{G}$ with additional VLinks. This augmentation can be done in two ways: i) VLinks can be augmented between arbitrary pair of VNodes to ensure $k+1$ edge connectivity, which is a well studied problem [19], [20]; ii) the other way is to augment only parallel VLinks between adjacent VNodes in $\bar{G}$ [12], [13]. Arbitrary augmentation can ensure $k+1$ edge connectivity by introducing minimal number of VLinks, but this approach will change the input VN topology. Although parallel VLink augmentation may not be minimal in terms of resource usage, it does not change the input VN topology. From VN user perspective, it is very important to preserve the input VN topology. Hence, we opt for the second alternative, *i.e.*, to augment parallel VLinks only.

We use the term k-protected VN, $\hat{G} = (\hat{V}, \hat{E})$, to represent a VN that is made $k+1$ edge connected by adding parallel VLinks to an input VN, $\bar{G} = (\bar{V}, \bar{E})$. Here, $\hat{V} = \bar{V}$ and $\hat{E} = \bar{E} \cup \tilde{E}$ where $\tilde{E}$ is the set of parallel VLinks to be added. Determining the capacity of the parallel VLinks (in $\tilde{E}$) as well as the amount of spare capacity to be reserved for the input VLinks (in $\bar{E}$) in order to guarantee full bandwidth of a failed VLink is a separate problem of its own [21]. In this work, we assume that the capacity of a parallel VLink will be the same as the capacity of the input VLink it augments.

C. Definitions

Definition 1. ***k-protected component:*** *A k-protected component of a graph $\bar{G}$ is a multi-graph $\hat{G}_k = (\hat{V}_k, \hat{E}_k)$, where $\hat{V}_k \subseteq \bar{V}$, $\hat{E}_k = \bar{E}_k \cup \tilde{E}_k$, $\bar{E}_k \subseteq \bar{E}$, $\tilde{E}_k \subseteq \tilde{E}$ and $\tilde{E}_k$ is a set of parallel VLinks augmented in such a way that simultaneous removal of k arbitrary VLinks in $\hat{G}_k$ will not partition $\hat{G}_k$.*

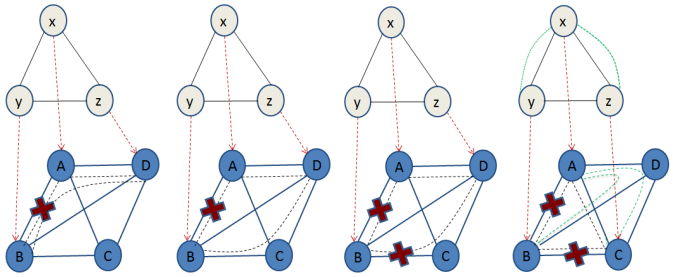
Definition 2. ***Conflicting VLinks:*** *Two VLinks are considered as conflicting if they must be embedded on edge-disjoint substrate paths in order to ensure $k+1$ edge connectivity.*

Definition 3. ***Conflicting set:*** *A conflicting set of a VLink $(\hat{u}, \hat{v})$, denoted by $\chi^{\hat{u}\hat{v}}$, is the set of VLinks in $\hat{E}$ those are conflicting with $(\hat{u}, \hat{v})$. A Conflicting set of a VN $\hat{G} = (\hat{V}, \hat{E})$, denoted by $\chi^{\hat{G}}$, is defined as* $\chi^{\hat{G}} = \bigcup_{\forall(\hat{u},\hat{v})\in\hat{E}} \chi^{\hat{u}\hat{v}}$.

D. CoViNE Problem Statement

Given an SN $G = (V, E)$, a VN $\bar{G} = (\bar{V}, \bar{E})$, and location constraints $L(\bar{u})$ for all $\bar{u} \in \bar{V}$ find an embedding that

- provides a function $f : \bar{V} \rightarrow V$ to map every VNode $\bar{u} \in \bar{V}$ to exactly one SNode $u \in V$ while satisfying the location constraint and without any overlap, i.e., $\forall \bar{u}, \bar{v} \in \bar{V} \wedge \bar{u} \neq \bar{v} \implies f(\bar{u}) \neq f(\bar{v})$ and $\forall \bar{u} \in \bar{V}\ \ f(\bar{u}) \in L(\bar{u})$,
- provides a function $g : \bar{E} \rightarrow 2^E$ to map each VLink $(\bar{u}, \bar{v}) \in \bar{E}$ to a substrate path $Q^{f(\bar{u})f(\bar{v})}$ with sufficient bandwidth to satisfy the VLink demand $b_{\bar{u}\bar{v}}$,
- ensures the connectivity in $\bar{G}$ in presence of up to k SLink failures in G,
- minimizes the total cost of embedding in terms of substrate bandwidth consumption.



(a) Single Failure (b) Double Failure

Fig. 1. CoViNE examples

We illustrate CoViNE examples for different failure scenarios in Fig. 1. In these examples, xyz is the VN and $ABCD$ is the SN. The arrow from a VNode to an SNode defines node mapping and the dotted lines between SNodes define link mapping. To survive single $(k = 1)$ failure, the VN must be 2 edge-connected. Since xyz VN is already 2 edge-connected, no augmentation is required. Fig. 1(a) shows an un-survivable embedding (on the left) and a survivable embedding (on the right) of the xyz VN. They differ in satisfying disjointness constraints. The embedding on the left satisfies no disjointness constraint, hence VLinks (x, y) and (y, z) share an SLink (A, B). Upon the failure of (A, B), both VLinks fail, and VNode y is disconnected from the rest of the VN. The embedding on the right adheres to the disjointness constraints, hence no sharing of SLinks is possible. Even though SLink (A, B) and correspondingly VLink (x, y) fail, the VN remains connected.

Fig. 1(b) exhibits the double $(k = 2)$ failure scenario for the same VN and SN topology. To survive double failures, the VN must have 3 edge-connectivity which is absent in the xyz VN. The embedding on the left demonstrates that even an edge-disjoint embedding of the VN results in an un-survivable embedding for double link failures due to the lack of necessary edge-connectivity in the VN. For the embedding on the right, the VN has necessary edge-connectivity through augmentation of green colored VLinks, and embedding is done adhering to the disjointness constraints resulting in a survivable embedding. It is to be noted that for the augmented VN on the right, not all the VLinks need to be disjoint with each other, hence there are some sharing of SLinks.

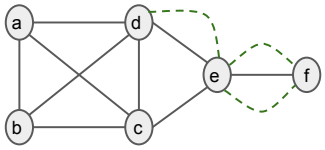


Fig. 2. The VN with only solid edges is the input VN, $\bar{G}$. The VN with both solid edges ($\bar{E}$) and dashed edges ($\tilde{E}$) is the 2-protected VN, $\hat{G}$. Any subgraph of $\hat{G}$ having 3 edge connectivity is $\hat{G}_2$.

IV. Problem Formulation

In this section, we propose a simple but efficient mechanism for computing *Conflicting set* of a VN (§ IV-A). We also show that this mechanism helps to transform a given VN $\bar{G}$ to a k-protected VN $\hat{G}$ by augmenting parallel VLinks only (§ IV-B).

A. Conflicting Set Computation

In this subsection, we assume that the VNs are k-protected, while the mechanism for transforming an arbitrary VN to a k-protected VN is discussed in the next subsection. In order to remain connected in presence of k SLink failures, the embedding algorithm must ensure $k+1$ edge connectivity between SNodes $f(\bar{u})$ and $f(\bar{v})$ for every pair of VNodes $\hat{u} \in \hat{V}$ and $\hat{v} \in \hat{V}$ of the k-protected VN $\hat{G}$. This can be achieved if the VLinks of every edge-cut in $\hat{G}$ are embedded on at least $k+1$ edge-disjoint paths in G. Since there are exponential number of edge-cuts in $\hat{G}$ and there are combinatorial number of ways of choosing $k+1$ conflicting VLinks from an edge-cut in $\hat{G}$, the number of possibilities for computing a conflicting set of $\hat{G}$ is enormous. However, an *optimal conflicting set* of $\hat{G}$ is one that ensures $k+1$ edge connectivity of the embedding of $\hat{G}$ while minimizing disjoint path requirement in the embedding. This can be achieved by finding the minimum number of partitions of the VLinks of $\hat{E}$ such that the VLinks in a partition are not conflicting with each other. Since the VLinks in a partition do not impose any disjointness constraint, minimizing the number of partitions will yield optimal conflicting set. Computing the optimal conflicting set of a VN is NP-complete since it can be reduced to the the well-known *Minimum vertex coloring* problem.[1] The following results provide a basis of our heuristic algorithm for computing the conflicting set of a VN $\hat{G}$.

Theorem 1. *([22]) The size of the minimum edge-cut for two distinct VNodes $\hat{u}, \hat{v} \in \hat{G}$ is equal to the maximum number of edge-disjoint paths between $\hat{u}$ and $\hat{v}$ in $\hat{G}$.*

According to Theorem 1, also known as Menger's Theorem, any pair of VNodes $\hat{u}$ and $\hat{v}$ in $\hat{G}$ will remain connected in presence of k SLink failures, if at least one of the edge-disjoint paths $P_i^{\hat{u}\hat{v}} \in \mathscr{P}^{\hat{u}\hat{v}}$ remains intact. This can be achieved by mapping any $k+1$ paths in $\mathscr{P}^{\hat{u}\hat{v}}$ into $k+1$ edge-disjoint paths in the SN. There are a combinatorial number of ways of choosing these $k+1$ edge-disjoint paths between $\hat{u}$ and $\hat{v}$. If $\mathscr{P}_1^{\hat{u}\hat{v}} = P_1^{\hat{u}\hat{v}}, P_2^{\hat{u}\hat{v}}, ..., P_{k+1}^{\hat{u}\hat{v}}$ is one possible combination chosen to have edge-disjoint mapping, two VLinks $(\hat{x}, \hat{y}) \in P_i^{\hat{u}\hat{v}}$ and $(\hat{w}, \hat{z}) \in P_j^{\hat{u}\hat{v}}$, s.t. $x \neq w$ and $y \neq z$, cannot share an SLink in their mappings. Therefore, a VLink $(\hat{x}, \hat{y}) \in P_i^{\hat{u}\hat{v}}$ is conflicting with all other VLinks present in the paths in $\mathscr{P}_1^{\hat{u}\hat{v}} \setminus P_i^{\hat{u}\hat{v}}$, leading to $|\chi^{\hat{x}\hat{y}}| = \sum_{P_i^{\hat{u}\hat{v}} \in \mathscr{P}_1^{\hat{u}\hat{v}} \wedge (\hat{x},\hat{y}) \notin P_i^{\hat{u}\hat{v}}} |P_i^{\hat{u}\hat{v}}|$. For example, in Fig. 2, VNodes a and b will remain connected in presence of 2 SLink failures if the VLinks on paths $P_1^{ab} = (a,b)$, $P_2^{ab} = \{(a,d),(d,c),(c,b)\}$, and $P_3^{ab} = \{(a,c),(c,e),(e,d),(d,b)\}$ are mapped to disjoint SN paths. Hence, $\chi^{ab} = P_2^{ab} \cup P_3^{ab}$.

We now discuss some heuristics to reduce the above computation. First, we can ensure connectivity in $\hat{G}$ by ensuring connectivity in a minimum spanning tree $\hat{T}$ of $\hat{G}$. In this case, we need to compute $k+1$ edge-disjoint paths only for the $|\hat{V}|-1$ VLinks in $\hat{T}$, as opposed to considering all the VLinks in $\hat{G}$. For the VN in Fig. 2, $k+1$ edge-disjoint path computations are required for the VLinks in $\hat{T} = \{(a,b),(a,c),(c,d),(d,e),(e,f)\}$ instead of all the 12 VLinks in $\hat{G}$. Second, instead of arbitrarily picking $k+1$ edge-disjoint paths from $\mathscr{P}^{\hat{u}\hat{v}}$, we can pick the first $k+1$ edge-disjoint shortest paths between $\hat{u}$ and $\hat{v}$. Thus, the size of the conflicting set of a VLink $(\hat{u},\hat{v})$ in $\hat{T}$ becomes $|\chi^{\hat{u}\hat{v}}| = \sum_{\mathtt{p}_i^{\hat{u}\hat{v}} \in \mathscr{P}^{\hat{u}\hat{v}} \wedge (\hat{u},\hat{v}) \notin \mathtt{p}_i^{\hat{u}\hat{v}}}^{i \leq k+1} |\mathtt{p}_i^{\hat{u}\hat{v}}|$, where $\mathtt{p}_i^{\hat{u}\hat{v}}$ is the ith edge-disjoint shortest path between two adjacent VNodes $\hat{u}$ and $\hat{v}$. This method yields smaller conflicting set $\chi^{ab} = \mathtt{p}_2^{ab} \cup \mathtt{p}_3^{ab}$, where $\mathtt{p}_2^{ab} = \{(a,c),(c,b)\}$, and $\mathtt{p}_3^{ab} = \{(a,d),(d,b)\}$ in Fig. 2.

Definition 4. ***Expansion Operator*** $\odot$*: Given a k-protected component $\hat{G}_k$ of a VN $\hat{G}$ and a VNode $\hat{v}$ s.t., $\hat{v} \in \hat{V} \setminus \hat{V}_k$ and $\exists \hat{u} \in \hat{V}_k, \hat{v} \in \mathcal{N}(\hat{u})$, we define $\hat{G}_k \odot \hat{v}$ as an expansion of $\hat{G}_k$ generated by adding $\hat{v}$ and all the incident VLinks on $\hat{v}$ from any VNode in $\hat{G}_k$. Mathematically,*

$$\hat{G}_k \odot \hat{v} = (\hat{V}_k \cup \{\hat{v}\}, \hat{E}_k \cup \{(\hat{u},\hat{v}) | \hat{u} \in \hat{V}_k, \hat{u} \in \mathcal{N}(\hat{v})\})$$

Definition 5. ***EDSP*** $\mathcal{P}^{\hat{G}_k\hat{v}}$***:*** *We define EDSP as a set of Edge-Disjoint Shortest Paths $\mathcal{P}^{\hat{G}_k\hat{v}} = \{\mathtt{p}_i^{\hat{x}\hat{v}}\}$ between $\hat{G}_k$ and a VNode $\hat{v} \in \hat{V} \setminus \hat{V}_k$ s.t. $\hat{x} \in \hat{V}_k$ and all $\mathtt{p}_i^{\hat{x}\hat{v}}$ terminate as the first VNode $\hat{x}$ in $\hat{V}_k$ is encountered, i.e., the only VNode from $\hat{V}_k$ that is on $\mathtt{p}_i^{\hat{x}\hat{v}}$ is $\hat{x}$.*

Observation 1: Using the expansion lemma, it can be shown that $\hat{G}_k \odot \hat{v}$ is a k-protected component if and only if there exists $k+1$ edge-disjoint paths from $\hat{G}_k$ to $\hat{v}$ in $\hat{G}$ [8]. Furthermore, it can be intuitively observed that in $\hat{G}_k \odot \hat{v}$, any $k+1$ EDSPs in $\mathcal{P}^{\hat{G}_k\hat{v}}$ will not contain a VLink from $\hat{E}_k$.

Lemma 1. *In an expansion $\hat{G}_k \odot \hat{v}$, the size of the conflicting set of a VLink $(\hat{u},\hat{v}) \in \hat{E} \setminus \hat{E}_k$ is $|\chi_{\odot}^{\hat{u}\hat{v}}| = \sum_{\mathtt{p}_i^{\hat{u}\hat{v}} \in \mathcal{P}^{\hat{G}_k\hat{v}} \wedge (\hat{u},\hat{v}) \notin \mathtt{p}_i^{\hat{u}\hat{v}}}^{i \leq k+1} |\mathtt{p}_i^{\hat{u}\hat{v}}|$, where $\hat{u} \in \hat{V}_k$ and $\hat{v} \in \mathcal{N}(\hat{u})$.*

Proof. For the embedding of $\hat{G}_k \odot \hat{v}$ on G to remain connected in presence of k SLink failures, we need to satisfy two conditions: i) at least $k+1$ edge-disjoint paths from $\hat{v}$ to $\hat{G}_k$ exist (*i.e.*, $|\mathcal{P}^{\hat{G}_k\hat{v}}| \geq k+1$), and ii) all of these paths are embedded on $k+1$ edge-disjoint paths in G. Therefore, the VLink $(\hat{u},\hat{v})$ is conflicting with all the VLinks in the first $k+1$ edge-disjoint shortest paths in $\mathcal{P}^{\hat{G}_k\hat{v}} \setminus \mathtt{p}_i^{\hat{u}\hat{v}}$, where $(\hat{u},\hat{v}) \in \mathtt{p}_i^{\hat{u}\hat{v}}$. This leads to $|\chi_{\odot}^{\hat{u}\hat{v}}| = \sum_{\mathtt{p}_i^{\hat{u}\hat{v}} \in \mathcal{P}^{\hat{G}_k\hat{v}} \wedge (\hat{u},\hat{v}) \notin \mathtt{p}_i^{\hat{u}\hat{v}}}^{i \leq k+1} |\mathtt{p}_i^{\hat{u}\hat{v}}|$. □

[1] *Minimum vertex coloring* is to color the vertices of a graph with a minimum number of colors so that adjacent vertices are of different colors.

Theorem 2. *In comparison to computing conflicting set $\chi^{\hat{u}\hat{v}}$ independently for a VLink $(\hat{u},\hat{v}) \in \hat{E}$, computing conflicting set for $(\hat{u},\hat{v})$ through the expansion $\hat{G}_k \odot \hat{v}$ will generate conflicting sets of lesser or equal size.*

Proof. Let's consider two VNodes $\hat{u} \in \hat{V}_k$ and $\hat{v} \in \hat{V} \setminus \hat{V}_k$ s.t. $\hat{v} \in \mathcal{N}(\hat{u})$. When computed independently, the size of the conflicting set of $(\hat{u},\hat{v})$ is $|\chi^{\hat{u}\hat{v}}| = \sum_{\mathtt{p}_i^{\hat{u}\hat{v}} \in \mathscr{P}^{\hat{u}\hat{v}} \wedge (\hat{u},\hat{v}) \notin \mathtt{p}_i^{\hat{u}\hat{v}}}^{i \leq k+1} |\mathtt{p}_i^{\hat{u}\hat{v}}|$. On other hand, when we construct conflicting set through the expansion, $\hat{G}_k \odot \hat{v}$, the size of the conflicting set of the VLink $(\hat{u},\hat{v}) \in \hat{E} \setminus \hat{E}_k$ is $|\chi_{\odot}^{\hat{u}\hat{v}}| = \sum_{\mathtt{p}_i^{\hat{u}\hat{v}} \in \mathcal{P}^{\hat{G}_k \hat{v}} \wedge (\hat{u},\hat{v}) \notin \mathtt{p}_i^{\hat{u}\hat{v}}}^{i \leq k+1} |\mathtt{p}_i^{\hat{u}\hat{v}}|$ (as proven in Lemma 1). In the beginning, when $\hat{G}_k$ contains only one VNode i.e. $|\hat{V}_k| = 1$, it is obvious that $|\chi^{\hat{u}\hat{v}}| = |\chi_{\odot}^{\hat{u}\hat{v}}|$. For $|\hat{V}_k| > 1$, consider $\hat{x} \in \hat{V}_k$ s.t. $\exists \mathtt{p}_i^{\hat{u}\hat{v}} \in \mathscr{P}^{\hat{u}\hat{v}}$ contains $\hat{x}$ and $\mathtt{p}_j^{\hat{x}\hat{v}} \in \mathcal{P}^{\hat{G}_k \hat{v}}$. Since $\mathtt{p}_i^{\hat{u}\hat{v}}$ contains $\hat{x}$, according to the optimal substructure property of shortest path, we get $\mathtt{p}_i^{\hat{u}\hat{v}} = \mathtt{p}_i^{\hat{u}\hat{x}} || \mathtt{p}_j^{\hat{x}\hat{v}}$, assuming $||$ is the path concatenation operator. Thus, $|\mathtt{p}_j^{\hat{x}\hat{v}}| < |\mathtt{p}_i^{\hat{u}\hat{v}}|$ resulting into $|\chi_{\odot}^{\hat{u}\hat{v}}| < |\chi^{\hat{u}\hat{v}}|$. If such an $\hat{x}$ is not found, we can assume $\hat{x} = \hat{u}$ and in that case $\mathtt{p}_j^{\hat{x}\hat{v}} = \mathtt{p}_i^{\hat{u}\hat{v}}$ yielding $|\chi_{\odot}^{\hat{u}\hat{v}}| = |\chi^{\hat{u}\hat{v}}|$. Hence, $|\chi_{\odot}^{\hat{u}\hat{v}}| \leq |\chi^{\hat{u}\hat{v}}|$. □

As an example of Theorem 2, let us consider the VLink (d,e) in Fig. 2 and the VN needs to survive against single SLink failure. If we compute independently, we get $\chi^{de} = \{(d,c),(c,e)\}$. When we compute through expansion $\hat{G}_1 \odot e$ where $\hat{V}_1 = \{a,b,c,d\}$, we get $\chi^{de} = \{(c,e)\}$.

B. VLink Augmentation

As described in § III-B, we may need to augment a given VN $\bar{G}$ with parallel VLinks in order to make it a k-protected VN $\hat{G} = (\hat{V},\hat{E})$. Now, the challenge here is to minimize the number of augmented parallel VLinks. We use Menger's Theorem [22] to find the pair of VNodes with less than $k+1$ edge connectivity and add parallel VLinks as needed. Assume that for each pair of adjacent VNodes $\bar{u},\bar{v} \in \bar{V}$ there are at least m edge-disjoint paths in $\bar{G}$. If $m \geq k+1$, $\bar{G}$ is at least $k+1$ edge-connected, hence no augmentation is needed. If $m < k+1$, we need to add $k+1-m$ parallel VLinks between $\bar{u}$ and $\bar{v}$. In general, $max(0, k+1-m)$ parallel VLinks are needed for each pair of adjacent VNodes. For instance, a VN should be 3 edge connected to survive 2 SLink failures. Since there are 2 edge-disjoint paths between d and e in Fig. 2, we add a parallel VLink. Similarly, we add two parallel VLinks between e and f to make the VN 3 edge connected. No augmentation is required for the rest of the adjacent pair of VNodes. It can be easily shown that the number of parallel VLinks to be augmented remains the same during the expansion, $\hat{G}_k \odot \bar{v}$. In other words, if there are $\hat{m}$ edge-disjoint paths from $\hat{G}_k$ to $\bar{v}$ in $\bar{G}$, augmentation of $max(0, k+1-\hat{m})$ parallel VLinks is needed to ensure the $k+1$ edge connectivity between $\hat{u}$ and $\bar{v}$, where $\hat{u} \in \hat{V}_k$ and $\bar{v} \in \mathcal{N}(\hat{u})$.

V. Heuristic Algorithm for Conflicting Set

Given the NP-complete nature of computing optimal conflicting set (§ IV), we propose a heuristic algorithm (Algorithm 1) for computing conflicting sets within a reasonable time. Algorithm 1 starts with a k-protected component, $\hat{G}_k$, containing an arbitrary VNode $\bar{u} \in \bar{V}$. The algorithm then includes all of $\bar{u}$'s neighbors $\bar{v} \in \mathcal{N}(\bar{u})$ to $\hat{V}_k$. This process is repeated until all the VNodes of $\bar{G}$ are added to $\hat{G}_k$. For each $\bar{v}$, the algorithm computes $k+1$ EDSPs, $\mathcal{P}^{\hat{G}_k \bar{v}}$ between $\hat{G}_k$ and $\bar{v}$ using the procedure *COMPUTE-EDSP* (Line 8). This procedure initially includes the VLink, $(\bar{u},\bar{v})$ as the first shortest path $\mathtt{p}_1^{\hat{G}_k \bar{v}}$ to $\mathcal{P}^{\hat{G}_k \bar{v}}$. It then invokes *Dijkstra's shortest path* algorithm k times to compute $\mathtt{p}_i^{\hat{G}_k \bar{v}}$, the ith EDSP between $\hat{G}_k$ and $\bar{v}$. After computing each $\mathtt{p}_i^{\hat{G}_k \bar{v}}$, all the VLinks present in $\mathtt{p}_i^{\hat{G}_k \bar{v}}$ are removed from $\bar{G}$ in order to ensure the edge-disjointness of the later paths. Although we use *Dijkstra's shortest path* algorithm repeatedly to compute $k+1$ EDSPs, other algorithms from the literature can be used for this purpose [9]. If the number of computed EDSPs is less than $k+1$, Algorithm 1 adds $k+1-|\mathcal{P}^{\hat{G}_k \bar{v}}|$ parallel VLinks between $\bar{u}$ and $\bar{v}$ (Line 10). The ith parallel VLink is denoted by $(\bar{u},\bar{v})^i$, and constitutes the $(|\mathcal{P}^{\hat{G}_k \bar{v}}|+i)$th EDSP between $\hat{G}_k$ and $\bar{v}$. Finally, Algorithm 1 updates the conflicting sets of the corresponding VLinks as described in Lemma 1 (Line 13).

Algorithm 1 Compute Conflicting Sets

```
1: function COMPUTE-CONFLICTING-SETS(Ḡ)
2:     ∀(ū, v̄) ∈ Ē: χ^{ūv̄} ← φ, Q ← φ
3:     ∃v̄ ∈ V̄ : Ĝ_k ← ({v̄}, φ) // v̄ is an arbitrary VNode
4:     ENQUEUE(Q, v̄)
5:     while Q is not empty do
6:         ū ← DEQUEUE(Q)
7:         for all v̄ ∈ N(ū) and v̄ ∉ Ĝ_k do
8:             P^{Ĝ_k v̄} ← COMPUTE-EDSP(Ḡ, Ĝ_k, ū, v̄, k+1)
9:             for i = 1 → (k+1−|P^{Ĝ_k v̄}|) do
10:                Ē ← Ē ∪ (ū, v̄)^i, P^{Ĝ_k v̄} ← P^{Ĝ_k v̄} ∪ (ū, v̄)^i
11:            end for
12:            ∀(x̄, ȳ) ∈ p_i^{Ĝ_k v̄}, p_i^{Ĝ_k v̄} ∈ P^{Ĝ_k v̄} :
13:              χ^{x̄ȳ} ← χ^{x̄ȳ} ∪ {(s̄, t̄) ∈ p_j^{Ĝ_k v̄} | p_j^{Ĝ_k v̄} ∈ P^{Ĝ_k v̄} ∧ i ≠ j}
14:            Ĝ_k ← Ĝ_k ⊙ v̄
15:            ENQUEUE(Q, v̄)
16:        end for
17:    end while
18: return χ^Ḡ
19: end function
```

The time complexity of the heuristic algorithm is dominated by the *COMPUTE-EDSP* procedure, which invokes *Dijkstra's shortest path* algorithm k times. The time complexity of *Dijkstra's shortest path* algorithm based on a min-priority queue is $O(|\bar{E}| + |\bar{V}| \log |\bar{V}|)$. Since *COMPUTE-EDSP* is invoked $O(|\bar{V}||\mathcal{N}(\bar{u})|)$ times, the running time of Algorithm 1 becomes $O(k|\bar{V}||\mathcal{N}(\bar{u})|(|\bar{E}| + |\bar{V}| \log |\bar{V}|))$.

VI. ILP Formulation for CoViNE

In this section, we present an Integer Linear Programming (ILP) formulation for CoViNE. The ILP minimizes the total cost of provisioning bandwidth for the VLinks of a VN, $\hat{G}$.

We represent the location constraint $L(\hat{u}) \subseteq V$ of $\hat{u} \in \hat{V}$ with the binary variable $\ell_{\hat{u}u}$ defined as follows:

$$\ell_{\hat{u}u} = \begin{cases} 1 & \text{if } \hat{u} \in \hat{V} \text{ can be mapped to } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

A. Decision Variables

A VLink is mapped to a path in SN. The following decision variable indicates the mapping between a VLink $(\hat{u}, \hat{v}) \in \hat{E}$ and an SLink $(u, v) \in E$.

$$x_{uv}^{\hat{u}\hat{v}} = \begin{cases} 1 & \text{if } (\hat{u}, \hat{v}) \in \hat{E} \text{ is mapped to } (u, v) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

The following variable represents VNode mapping:

$$y_{\hat{u}u} = \begin{cases} 1 & \text{if } \hat{u} \in \hat{V} \text{ is mapped to } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

B. Constraints

1) VLink Mapping Constraints: Our VLink mapping constraints are as follows:

$$\forall (\hat{u}, \hat{v}) \in \hat{E} : \sum_{\forall (u,v) \in E} x_{uv}^{\hat{u}\hat{v}} \geq 1 \tag{1}$$

$$\forall (u, v) \in E : \sum_{\forall (\hat{u},\hat{v}) \in \hat{E}} x_{uv}^{\hat{u}\hat{v}} \times b_{\hat{u}\hat{v}} \leq b_{uv} \tag{2}$$

$$\forall \hat{u}, \hat{v} \in \hat{V}, \forall u \in V : \sum_{\forall v \in \mathcal{N}(u)} (x_{uv}^{\hat{u}\hat{v}} - x_{vu}^{\hat{u}\hat{v}}) = y_{\hat{u}u} - y_{\hat{v}u} \tag{3}$$

(1) ensures that each VLink is mapped to a non-empty set of SLinks and no VLink is left unmapped. (2) ensures that an SLink is not assigned VLink demands that exceeds the SLink's capacity. Finally, (3) ensures that the in-flow and out-flow of each SNode is equal except at the SNodes where the endpoints of a VLink are mapped following the constraint in [23].

2) VNode Mapping Constraints: We map the VNodes according to the location constraint as described in (4). We also ensure that a VNode is mapped to exactly one SNode by (5). Finally, (6) ensures that an SNode does not host more than one VNode from the same VN. The VNode mapping follows from the VLink mapping since we do not have any VNode mapping cost.

$$\forall \hat{u} \in \hat{V}, \forall u \in V : y_{\hat{u}u} \leq \ell_{\hat{u}u} \tag{4}$$

$$\forall \hat{u} \in \hat{V} : \sum_{u \in V} y_{\hat{u}u} = 1 \tag{5}$$

$$\forall u \in V : \sum_{\hat{u} \in \hat{V}} y_{\hat{u}u} \leq 1 \tag{6}$$

3) Disjointness Constraints: To ensure the desired survivability of CoViNE, a VLink $(\hat{u}, \hat{v}) \in \hat{E}$ should never share an SLink with it's conflicting VLinks in $\chi^{\hat{u}\hat{v}}$ in their mappings. This disjointness requirement is ensured with the following constraint $\forall (u, v) \in E, \forall (\hat{u}, \hat{v}) \in \hat{E}, \forall (\hat{a}, \hat{b}) \in \chi^{\hat{u}\hat{v}}$:

$$x_{uv}^{\hat{u}\hat{v}} + x_{vu}^{\hat{u}\hat{v}} + x_{uv}^{\hat{a}\hat{b}} + x_{vu}^{\hat{a}\hat{b}} \leq 1 \tag{7}$$

C. Objective Function

Our objective is to minimize the bandwidth provisioning cost over all the SLinks used by the mappings of all the VLinks of a VN, $\hat{G}$. Given that C_{uv} is the cost of allocating unit bandwidth on SLink $(u, v) \in E$, we have the following objective function for our ILP:

$$\text{minimize} \left(\sum_{\forall (\hat{u},\hat{v}) \in \hat{E}} \sum_{\forall (u,v) \in E} x_{uv}^{\hat{u}\hat{v}} \times C_{uv} \times b_{\hat{u}\hat{v}} \right)$$

VII. Heuristic Algorithm for CoViNE

The ILP formulation presented in § VI cannot solve larger instances of the problem due to the limitation of LP solvers. Hence, we propose a heuristic algorithm (Algorithm 2) to produce near-optimal solutions within reasonable time limit. Algorithm 2 embeds $\hat{G}$ while ensuring the disjointness constraint imposed by $\chi^{\hat{G}}$ and minimizing the total cost of embedding according to the objective function in § VI-C.

Algorithm 2 computes two functions, $nmap$ and $emap$, which represent the VNode and VLink mapping of $\hat{G}$ on G, respectively. Since there is no cost associated with VNode mapping, a VLink mapping that minimizes total cost determines the VNode mapping. Algorithm 2 first sorts the VNodes $\hat{u} \in \hat{V}$ in decreasing order of the sum of conflicting set sizes of incident VLinks. This sorted list of VNodes is represented by $\hat{\mathcal{V}}$. A VNode with VLinks having larger conflicting sets becomes too constrained to be mapped to a suitable SNode, hence, Algorithm 2 tries to map VNodes in the order of $\hat{\mathcal{V}}$.

For each VNode $\hat{u} \in \hat{\mathcal{V}}$, Algorithm 2 searches for an unallocated SNode in $\hat{u}$'s location constraint set, $L(\hat{u})$, which yields a feasible mapping while minimizing the cost. To embed $\hat{u}$, Algorithm 2 loops through each candidate SNode $l \in L(\hat{u})$ (Line $5-14$), to first temporarily map $\hat{u}$ to l (Line 6). Then

Algorithm 2 VN-Embedding

```
1: function VN-EMBEDDING(G, Ĝ)
2:     V̂ ← Sort û ∈ V̂ in decreasing order of Σ_{∀v̂∈N(û)} |χ^{ûv̂}|
3:     for all û ∈ V̂ do
4:         Candidate ← φ
5:         for all l ∈ L(û) do
6:             Add mapping û → l to nmap
7:             E ← Sort (û, v̂) ∈ Ê in decreasing order of |χ^{ûv̂}|
8:             ∀(û, v̂) ∈ E such that emap(û, v̂) = φ
9:                 P[(û, v̂)] ← VLINK-MAP(G, Ĝ, (û, v̂))
10:            if Σ_{∀(u,v)∈E} cost(P[(û, v̂)]) is minimum then
11:                M ← P, Candidate ← l
12:            end if
13:            nmap(û) ← φ, ∀(û, v̂) ∈ E: emap(û, v̂) ← φ
14:        end for
15:        if Candidate ≠ φ then
16:            Add mapping û → Candidate to nmap
17:            ∀(û, v̂) ∈ E and nmap(û) ≠ φ and nmap(v̂) ≠ φ:
18:                Add mapping (û, v̂) → M[(û, v̂)] to emap
19:        else
20:            return No Solution Found
21:        end if
22:    end for
23: return {nmap, emap}
24: end function
```

Algorithm 3 VLink-Map

1: **function** VLINK-MAP($G, \hat{G}, (\hat{u}, \hat{v})$)
2: $p^{\hat{u}\hat{v}} \leftarrow \phi$
3: $\forall (\hat{s}, \hat{t}) \in \chi^{\hat{u}\hat{v}}$:
4: $E \leftarrow E - \{(a, b) \in E | (\hat{s}, \hat{t})$ is mapped to $(a, b)\}$
5: **if** $nmap(\hat{u}) \neq \phi \wedge nmap(\hat{v}) \neq \phi$ **then**
6: $Q^{nmap(\hat{u})nmap(\hat{v})} \leftarrow$ MCP$(G, nmap(\hat{u}), nmap(\hat{v}), b_{\hat{u}\hat{v}})$
7: **else if** $nmap(\hat{x}) = \phi \wedge nmap(\hat{y}) \neq \phi$ s.t. $\hat{x}, \hat{y} \in \{\hat{u}, \hat{v}\}$, $\hat{y} \neq \hat{x}$ **then**
8: $Q^{nmap(\hat{u})nmap(\hat{v})} \leftarrow$
9: $\min_{\forall l \in L(\hat{x})} \{MCP(G, nmap(\hat{y}), l, b_{\hat{y}l})\}$
10: **end if**
11: **if** $Q^{nmap(\hat{u})nmap(\hat{v})} \neq \phi$ **then**
12: Add mapping $(\hat{u}, \hat{v}) \rightarrow Q^{nmap(\hat{u})nmap(\hat{v})}$ to $emap$
13: **end if**
14: **return** $Q^{nmap(\hat{u})nmap(\hat{v})}$
15: **end function**

the algorithm tries to embed all the VLinks incident to $\hat{u}$. The *VLINK-MAP* (Algorithm 3) procedure is invoked to find the mapping for each such VLink (Line 9). VLinks incident to $\hat{u}$ are processed in the decreasing order of their conflicting set sizes to maximize the chances of finding substrate paths that can satisfy the disjointness constraint enforced by the conflicting sets. Algorithm 2 finally embeds $\hat{u}$ to the l that leads to a feasible mapping for all the VLinks incident to $\hat{u}$ and yields the minimum embedding cost. The algorithm fails, if no such feasible l is found. Once a VNode $\bar{u}$ has been finally mapped, Algorithm 2 creates the final mapping for only those VLinks incident to $\hat{u}$ whose both endpoints are already finally mapped (Line $17-18$). The mappings of other VLinks incident on $\hat{u}$ are finalized when their unmapped endpoints are mapped. We now describe the *VLINK-MAP* (Algorithm 3) procedure for finding the mapping of a VLink, $(\hat{u}, \hat{v})$. First we remove all the SLinks used by the mapping of all the VLinks in $\chi^{\hat{u}\hat{v}}$ to satisfy the disjointness constraint (Line $3-4$). Then, we compute mapping for $(\hat{u}, \hat{v})$ by considering the following two cases: (i) both endpoints of $(\hat{u}, \hat{v})$ have already been mapped to some SNodes (Line 5). In this case, we find a minimum cost path between $nmap(\hat{u})$ and $nmap(\hat{v})$ with capacity at least $b_{\hat{u}\hat{v}}$ in G; (ii) only $\hat{u}$ (or $\hat{v}$) is mapped and the other endpoint $\hat{v}$ (or $\hat{u}$) has not been mapped (Line 7). In this case, we compute the minimum cost path between $nmap(\hat{u})$ (or $nmap(\hat{v})$) and all possible locations for the unmapped VNode $\hat{v}$ (or $\hat{u}$), $l \in L(\hat{v})$ (or $L(\hat{u})$) with at least $b_{\hat{u}\hat{v}}$ capacity. $(\hat{u}, \hat{v})$ is temporarily mapped to this path and the mapping is added to $emap$ (Line 12). We modified *Dijkstra's shortest path* algorithm to consider link capacities while computing the minimum cost path (MCP procedure call in Algorithm 3). The cost for each SLink is set $(u, v) \in E$ to $C_{uv} \times b_{\hat{u}\hat{v}}$, where $b_{\hat{u}\hat{v}}$ is the bandwidth requirement of the VLink to be embedded.

The most expensive step of Algorithm 2 is the *VLINK-MAP* function, which invokes *Dijkstra's shortest path* algorithm on the SN requiring $O(|E|+|V|\log|V|)$ time. Since *VLINK-MAP* is invoked $O(|\hat{V}||L(\hat{u})||\mathcal{N}(\hat{u})|)$ times, the running time of Algorithm 2 becomes $O(|\hat{V}||L(\hat{u})||\mathcal{N}(\hat{u})|(|E|+|V|\log|V|))$.

VIII. Evaluation

A. Compared Approaches

We compare six approaches (Table II) that combine different strategies for computing disjointness constraint and VN embedding. We have chosen single failure ($k = 1$) and double failure ($k = 2$) scenarios, since the possibility of more than two simultaneous link failures is very low [7], [3]. The first four approaches in Table II are from our contributions, while the last two are based on [10] and [24].

TABLE II
Compared Algorithms

Notation	Failures	Disjointness	Embedding
S-CoViNE	Single	Algorithm 1	Algorithm 2
D-CoViNE	Double	Algorithm 1	Algorithm 2
S-CoViNE-ILP	Single	Algorithm 1	§ VI
D-CoViNE-ILP	Double	Algorithm 1	§ VI
S-Cutset-ILP [10]	Single	Optimal Cut-set	ILP
ViNE-ILP [24]	None	None	MCUF ILP[1]

1. Multi-commodity Unsplittable Flow

B. Simulation Setup

We implement the ILP formulations using IBM ILOG CPLEX C++ library. The simulations were performed on a server with quad-core 3.4GHz processor and 8GB of RAM. To demonstrate the scalability of our solutions, we consider both small and large network topologies as summarized in Table III. For each problem instance in this table, we perform 3 simulation runs and take the average. VNs for the small scale scenario are 2-edge connected, since it is required by the cut-set based approach [10]. We also vary the Link-to-Node Ratio (LNR) to assess the robustness of our solution for different VN connectivity levels. In addition to scalability and robustness, we analyze the behavior of our approach under different failure scenarios. Since our focus is VN connectivity, we use enough bandwidth capacity in SN topologies.

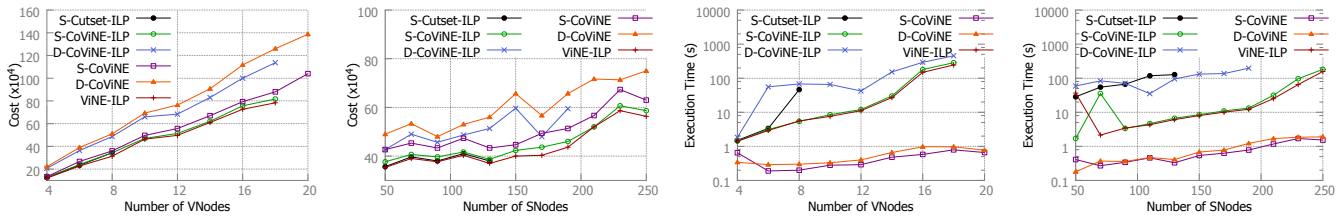
TABLE III
Summary of Simulation Parameters

Scenario	Figure	SNodes	SLinks	VNodes	VLinks
Small Scale	Fig. 3(a) Fig. 3(c)	150	310	4-20	5-37
	Fig. 3(b) Fig. 3(d)	50-250	105-494	10	17-24
Large Scale	Fig. 4(a)	500	2017	10-100	21-285
	Fig. 4(d)	1000	4023	10-100	21-285
	Fig. 4(b)	500	2017	10	11-31
		1000	4023	10	11-31
	Fig. 4(c)	500	1000-2000	10	21
		1000	2000-4000	10	21
Failure	Fig. 5	150	310	10	11-31

C. Results

1) Small Scale Scenarios:

a) Embedding Cost: Fig. 3(a) and Fig. 3(b) depict embedding cost for different VN and SN sizes, respectively. As expected, *ViNE-ILP* produces the lowest cost embedding, since it neither augments any parallel VLinks nor satisfies any disjointness constraint. The costs of embedding produced by *S-Cutset-ILP* and *S-CoViNE-ILP* lie very close to that of *ViNE-ILP*. Since the VNs are 2-edge connected in this experiment,



(a) Cost Vs. VN Size (b) Cost Vs. SN Size (c) Time Vs. VN Size (d) Time Vs. SN Size

Fig. 3. Small Scale Performance

no parallel augmentation is performed. The difference in *S-Cutset-ILP* and *S-CoViNE-ILP* is only due the variation of the disjointness computation methods. In contrast, *S-CoViNE* employs heuristic algorithms for disjointness computation and embedding, resulting in $\sim$10% more cost than *S-CoViNE-ILP* and $\sim$15% more cost than the cut set based optimal solution (*S-Cutset-ILP*). Both *D-CoViNE-ILP* and *D-CoViNE* incur $\sim$30% more cost than single failure approaches, since they augment parallel VLinks to ensure 3-edge connectivity. In general, cost increases almost linearly with increase in SN size for a fixed VN, and vice versa.

b) Execution Time: Fig. 3(c) and Fig. 3(d) present execution times in logarithmic scale by varying VN and SN sizes, respectively. Execution times of *S-CoViNE* and *D-CoViNE* vary almost linearly with VN or SN sizes. When embedding a VN of 18 VNodes on a 150 node SN, *S-CoViNE-ILP* and *D-CoViNE-ILP* take $\sim$ 285s and $\sim$ 456s, respectively, which is significantly slower compared to less than a second execution time for *S-CoViNE* and *D-CoViNE*. *ViNE-ILP* runs faster than *S-CoViNE-ILP* and *D-CoViNE-ILP*, since it does not satisfy any disjointness constraint while embedding. *S-Cutset-ILP* is the slowest since it computes an optimal solution.

c) Scalability: S-CoViNE and *D-CoViNE* can scale with arbitrary VN and SN sizes, whereas the ILP-based approaches can only scale up to 18 node VNs on 150 node SNs. Scalability of *S-Cutset-ILP* is the worst as it cannot scale beyond 10 node VNs on the same 150 node SNs. In summary, the higher costs of *S-CoViNE* and *D-CoViNE*, compared to the corresponding ILP-based approaches, are compensated by their higher scalability and faster execution time.

2) Large Scale Scenarios:

a) Embedding Cost: Fig. 4(a) shows embedding cost by varying VN sizes on SNs of 500 and 1000 nodes. On the other hand, Fig. 4(b) and Fig. 4(c) show embedding cost for VN and SN topologies with different LNRs, respectively. In this scenario, embedding cost is mostly influenced by disjointness constraint and parallel VLink augmentation. For double failure scenarios, augmentation cost dominates for VN LNR ≤ 2.4, hence the initial decrease in embedding cost. However for VN LNR > 2.4, cost for ensuring disjointness constraint dominates, which justifies the corresponding increase in Fig. 4(b). On the other hand for *S-CoViNE*, disjointness constraint dominates and embedding cost increases as higher number of VLinks are embedded on the same SN for larger LNR. An increase in SN LNR results into higher path diversity in SN. *S-CoViNE* and *D-CoViNE* exploit this path diversity by finding shorter paths while embedding a VLink. This accounts for the decrease in cost with an increase in SN LNR.

b) Execution Time: Conforming to the running time analysis in § V and § VII, the execution times for *S-CoViNE* and *D-CoViNE* increase with both VN and SN sizes (Fig. 4(d)).

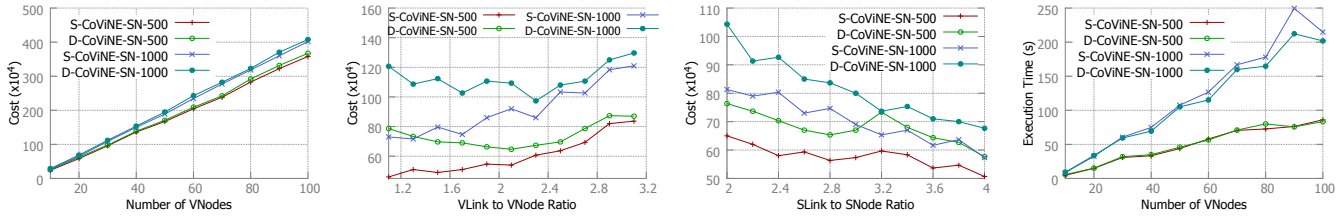
3) Impact of Failure: In this scenario, we assume three traffic classes in VN, labeled as 1 (highest priority), 2 and 3 (lowest priority) demanding 20%, 30%, and 50% of each VLink's bandwidth, respectively. We handle failures by rerouting traffic in the affected VLinks along alternate shortest paths in VN. Bandwidth sharing along these paths follow fair sharing policy between traffic from the same class and weighted fair sharing across different traffic classes.

Fig. 5(a) and Fig. 5(b) present the percentage of restored bandwidth for single and double failure scenarios, respectively. On the other hand, Fig. 5(c) and Fig. 5(d) present the overhead for ensuring connectivity in terms of embedding cost and number of augmented VLinks, respectively. Fig. 5(a) and Fig. 5(b) depict that performance of our embedding heuristics (*S-CoViNE* and *D-CoViNE*) is very close to the optimal embedding (*S-CoViNE-ILP* and *D-CoViNE-ILP*) for all three traffic classes. The percentage of restored bandwidth by ViNE-ILP is very poor at low VN LNR and increases with the increase in VN LNR. A higher LNR induces higher path diversity in VN. This has twofold impact. First, it reduces the chances of VN partitioning. Second, there are more options for steering traffic in the affected VLinks. Both of these reasons contribute to the increase in restored bandwidth for VN with higher LNR.

As envisioned at the beginning of this paper, our approach is able to successfully restore almost the full bandwidth for the highest priority traffic in presence of single and double failures as shown in Fig. 5(a) and Fig. 5(b). However, this successful restoration is at the expense of penalizing the lower priority traffic classes. The overall decrease in restored bandwidth for all variants of *CoViNE* with increasing VN LNR is counter-intuitive. This can be explained by observing the overheads in Fig. 5(c) and Fig. 5(d). As VN LNR increases, the number of augmented VLinks decreases. This results into lower spare bandwidth in VNs with higher LNR, and consequently reducing the percentage of restored bandwidth.

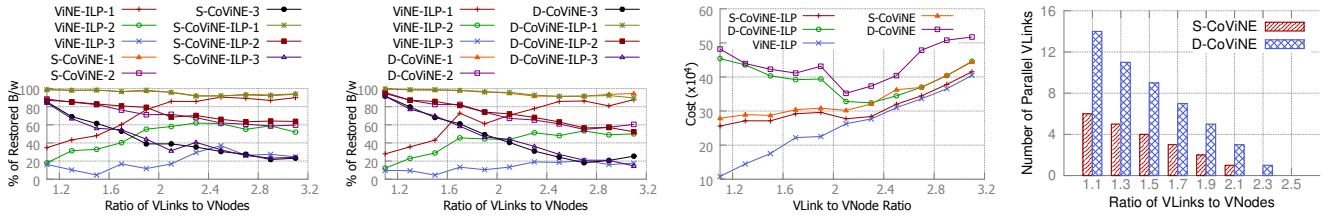
IX. Conclusion

In this paper, we have investigated the **Co***nnectivty-aware* **Vi***rtual* **N***etwork* **E***mbedding* (CoViNE) problem that ensures VN connectivity in presence of multiple SLink failures. We



(a) Cost Vs. VN Size (b) Cost Vs. VN LNR (c) Cost Vs. SN LNR (d) Time Vs. VN Size

Fig. 4. Large Scale Performance



(a) Single Failure (b) Double Failure (c) Overhead (d) Parallel VLink Augmention

Fig. 5. Impact of Failure

have addressed the two major challenges in solving *CoViNE*: i) finding the conflicting VLinks that should be embedded disjointly, and ii) computing a resource efficient embedding that adheres to disjointness requirement. For the first challenge, we have coined the concept of conflicting set, and have proven that computing optimal conflicting set is NP-complete. We also provided a heuristic algorithm for finding conflicting set efficiently. For the second challenge, we provided an ILP formulation and a heuristic to tackle its computational complexity. All of our solutions are generalized to handle multiple SLink failures for any VN and SN topology. Evaluation results show that solutions from our heuristics use around 15% extra resources on average compared to the optimal solution, whereas the execution time of our heuristic is two to three orders of magnitude faster on the same problem instances. We have also demonstrated that VN connectivity can be successfully applied to restore high priority traffic in presence of multiple SLink failures.

We believe that *CoViNE* can set the stage for further research investigations. Among the possibilities, we want to investigate the problem of ensuring different connectivity levels for each VLink in a VN, which can empower a VN-operator to offer a wide variety of Service Level Agreements (SLAs) to its customers. We also want to extend our current solutions by considering SLinks' spare bandwidth allocation, SNodes' throughput, and substrate path length constraints in a coordinated manner.

Acknowledgment

This work was supported in part by Huawei Technologies and in part by an NSERC Collaborative Research and Development Grant.

References

[1] N. M. M. K. Chowdhury *et al.*, "A Survey of Network Virtualization," *Computer Networks*, Apr 2010.

[2] M. R. Rahman *et al.*, "SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization," *IEEE TNSM*, 2013.

[3] A. Markopoulou *et al.*, "Characterization of Failures in an IP Backbone," in *INFOCOM*, Mar 2004.

[4] S. Herker *et al.*, "Survey on Survivable Virtual Network Embedding Problem and Solutions," in *ICNS*, 2013.

[5] Z. Zhou *et al.*, "Cross-layer network survivability under multiple cross-layer metrics," *IEEE/OSA J. of Optical Comm. & Net.*, 2015.

[6] H. Choi *et al.*, "Loopback recovery from double-link failures in optical mesh networks," *IEEE/ACM TON*, Dec 2004.

[7] P. Gill *et al.*, "Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications," in *ACM SIGCOMM*, Aug 2011.

[8] K. Thulasiraman *et al.*, "Logical topology augmentation for guaranteed survivability under multiple failures in ip-over-wdm optical networks," *Optical Switching and Networking*, vol. 7, no. 4, pp. 206–214, 2010.

[9] J. M. Kleinberg, "Approximation algorithms for disjoint paths problems," Ph.D. dissertation, Citeseer, 1996.

[10] E. Modiano *et al.*, "Survivable lightpath routing: a new approach to the design of wdm-based networks," *IEEE JSAC*, 2002.

[11] A. Todimala *et al.*, "A scalable approach for survivable virtual topology routing in optical wdm networks," *IEEE JSAC*, 2007.

[12] K. Thulasiraman *et al.*, "Circuits/cutsets duality and a unified algorithmic framework for survivable logical topology design in ip-over-wdm optical networks," in *IEEE INFOCOM*, Apr 2009.

[13] Z. Zhou *et al.*, "Novel survivable logical topology routing in ip-over-wdm networks by logical protecting spanning tree set," in *ICUMT*, 2012.

[14] M. Kurant *et al.*, "Survivable mapping algorithm by ring trimming (smart) for large ip-over-wdm networks," in *BroadNets*, Oct 2004.

[15] T. Guo *et al.*, "Shared backup network provision for virtual network embedding," in *IEEE ICC*, 2011.

[16] J. Xu *et al.*, "Survivable virtual infrastructure mapping in virtualized data centers," in *IEEE CLOUD*, 2012.

[17] M. M. A. Khan *et al.*, "Simple: Survivability in multi-path link embedding," in *IEEE CNSM*, 2015, pp. 210–218.

[18] K. Lee *et al.*, "Cross-layer survivability in wdm-based networks," *IEEE/ACM TON*, Aug 2011.

[19] C. Liu *et al.*, "A new survivable mapping problem in ip-over-wdm networks," *IEEE JSAC*, Apr 2007.

[20] M. Kurant *et al.*, "Survivable routing of mesh topologies in ip-over-wdm networks by recursive graph contraction," *JSAC*, 2007.

[21] D. D.-J. Kan *et al.*, "Lightpath routing and capacity assignment for survivable ip-over-wdm networks," in *IEEE DRCN*, 2009.

[22] Menger's theorem [online] http://math.fau.edu/locke/menger.htm.

[23] M. Melo *et al.*, "Virtual network mapping–an optimization problem," in *Mobile Networks and Management*. Springer, 2012, pp. 187–200.

[24] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components," in *IEEE INFOCOM*, 2006.