Protecting Virtual Networks with DRONE

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

***Abstract*—Network virtualization is enabling infrastructure providers (InPs) to offer new services to higher level service providers (SPs). InPs are usually bound by Service Level Agreements (SLAs) to ensure various levels of resource availability for different SPs' virtual networks (VNs). They provision redundant backup resources while embedding an SP's VN request to conform to the SLAs during physical failures in the infrastructure. An extreme of this backup resource provisioning is to reserve a dedicated backup of each element in an SP's VN request. Such dedicated protection scheme can enable an InP to ensure fast VN recovery, thus, providing high uptime guarantee to the SPs. In this paper, we study the $1+1$-Protected Virtual Network Embedding ($1+1$-*ProViNE*) problem. We propose Dedicated Protection for Virtual Network Embedding (*DRONE*), a suite of solutions to the $1+1$-*ProViNE*. *DRONE* includes an Integer Linear Programming (ILP) formulation for optimal solution (*OPT-DRONE*) and a heuristic (*FAST-DRONE*) to tackle the computational complexity in computing the optimal solution. Trace driven simulations show that *FAST-DRONE* allocates only 14.3% extra backup resources on average compared to the optimal solution, while executing 200x – 12000x faster.**

I. Introduction

Network virtualization has evolved as a key enabler for next generation of network services. Infrastructure providers (InPs) such as Data Center Network (DCN) operators and Internet Service Providers (ISPs) are rolling out network virtualization technologies to offer virtualized slices of their networking infrastructure to higher level Service Providers (SPs) [1]. Even the long haul connectivity providers, *i.e.*, the transport network operators are working towards leveraging Software Defined Networking (SDN) to offer full fledged virtual networks (VNs) to their customers [2], [3]. This next generation of transport network, also known as Transport SDN (T-SDN), gives customers more flexibility and control over their virtual slice and deploy their own routing and traffic engineering.

The benefits from network virtualization come at the cost of additional resource management challenges for the InP. A fundamental and well studied problem in this area is to efficiently embed a VN request from an SP on the physical network (PN), also known as the Virtual Network Embedding (VNE) problem [4]. Typical objectives for VNE include maximizing the fraction of embedded VNs [5], minimizing the resource provisioning cost on the PN [6], [7], *etc*. One particular aspect of VNE is to take the possibility of PN failures into account, known as the Survivable VNE (SVNE) [8] problem. Protection and restoration mechanisms exist in the literature for SVNE. Restoration approaches reactively take action after a failure has occurred, while protection approaches pro-actively provision backup resources when a VN is embedded.

One extreme case for the VN protection approach is to provision dedicated backup resource for each virtual node and virtual link in a VN request, also known as the $1+1$-protection scheme. $1+1$-protection has its roots back to Wavelength Division Multiplexing (WDM) optical networks where light paths are established with a dedicated backup path for recovering fiber cuts within tens of milliseconds [9], [10]. In network virtualization context, $1+1$-protection for VN is motivated by use cases from T-SDN. T-SDN leverages SDN technology to separate the control and optical switching planes of the Optical Transport Networks (OTNs) for flexible management and better automation. T-SDN envisions the coexistence of multiple customer VNs, each having full control over its virtual slice. These customer VNs carry high volume and high speed traffic, and usually have Service Level Agreements (SLAs) with the InP for recovery from physical failures within tens of milliseconds. To satisfy such tight SLAs, the InP needs to provision dedicated backup resources for the VN request, which can be used for immediate recovery from a physical failure. Otherwise, a prolonged recovery time can lead to service disruption for the SP, leading to revenue and reputation loss for the InP. However, such fast recovery with dedicated backup comes at the expense of provisioning idle backup resources in the network. Therefore, the InP should carefully provision VN requests to minimize resource provisioning cost.

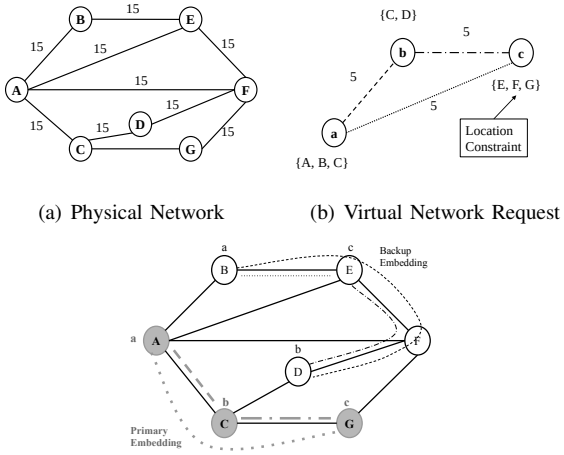
In this paper, we study the problem of $1+1$-**Pro**tected **Vi**rtual **N**etwork **E**mbedding ($1+1$-*ProViNE*) with the objective of minimizing resource provisioning cost in the PN, while protecting each node and link in a VN request with dedicated backup resource in PN. A major challenge in solving $1+1$-*ProViNE* is to find the primary and backup embedding at the same time. Relevant literature [11] shows that sequentially embedding the primary and backup can lead to failure in embedding even though a feasible embedding exists. In this regard, we propose **D**edicated P**ro**tection for Virtual **N**etwork **E**mbedding (*DRONE*), a suite of solutions for $1+1$-*ProViNE*. *DRONE* guarantees a VN to survive under a single physical node failure. We focus on single node failure scenario since it is the most probable case [12], [13], and leave the multiple failure scenario for future investigation. Specifically, we make the following contributions in this paper:

***OPT-DRONE*:** An Integer Linear Program (ILP) formulation to find the optimal solution for $1+1$-*ProViNE*, improving on the quadratic formulation from previous work [11]. We also show that $1+1$-*ProViNE* is at least as hard as jointly solving balanced graph partitioning and minimum unsplittable flow problems, both of which are NP-Hard [14], [15].

FAST-DRONE: A heuristic to tackle the computational complexity of *OPT-DRONE* and to find solution in a reasonable time frame. Trace driven simulations show that *FAST-DRONE* uses about $\mathbf{14.3}\%$ extra resources on average compared to *OPT-DRONE*, while executing **200x – 12000x** faster.

The rest of the paper is organized as follows. We begin with introducing the mathematical notations and a formal definition of $1+1$-*ProViNE* in Section II. Then we present the ILP formulation of $1+1$-*ProViNE*, *i.e.*, *OPT-DRONE* in Section III followed by the details of *FAST-DRONE* in Section IV. Section V presents our evaluation of *DRONE*. Finally, we conclude with some future research directions in Section VII.

II. Mathematical Model: $1+1$-*ProViNE*



(a) Physical Network (b) Virtual Network Request

(c) Embedding by *DRONE*

Fig. 1. Example embedding with *DRONE*

In this section, we first present a mathematical representation of the inputs: the PN and the VN request. Then we formally define $1+1$-*ProViNE*.

A. Physical Network

We represent the PN as an undirected graph, $G=(V,E)$, where V and E are the set of physical nodes and links, respectively. The set of neighbors of each physical node $u \in V$ is represented by $\mathcal{N}(u)$. Each physical link $(u,v) \in E$ has the following attributes: i) b_{uv} : bandwidth capacity of the link (u,v), ii) C_{uv} : cost of allocating unit bandwidth on (u,v) for provisioning a virtual link.

B. Virtual Network

We represent a VN as an undirected graph $\bar{G}=(\bar{V},\bar{E})$, where $\bar{V}$ and $\bar{E}$ are the set of virtual nodes and virtual links, respectively. Each virtual link $(\bar{u},\bar{v}) \in \bar{E}$ has bandwidth requirement $b_{\bar{u}\bar{v}}$. We also have a set of location constraints, $\mathcal{L}=\{L(\bar{u})|L(\bar{u}) \subseteq V, \forall \bar{u} \in \bar{V}\}$, such that a virtual node $\bar{u} \in \bar{V}$ can only be provisioned on a physical node $u \in L(\bar{u})$. The location constraint set for $\bar{u} \in \bar{V}$ contains all physical nodes when there is no location constraint for $\bar{u}$. The binary variable $\ell_{\bar{u}u}$ represents the location constraint as follows:

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

C. $1+1$-*ProViNE* Problem Statement

Given a PN $G=(V,E)$, VN request $\bar{G}=(\bar{V},\bar{E})$, and a set of location constraints $\mathcal{L}$, embed $\bar{G}$ on G such that:

- Each virtual node $\bar{u} \in \bar{G}$ has a primary and a backup embedding in the PN, satisfying the location constraint.
- For each virtual node $\bar{u} \in \bar{G}$, the physical nodes used for the primary embedding are disjoint from the physical nodes used for the backup embedding.
- Each virtual link $(\bar{u},\bar{v}) \in \bar{E}$ has a primary and a backup embedding in the PN. A primary or backup embedding of a virtual link on the PN corresponds to a single path in the PN having at least $b_{\bar{u}\bar{v}}$ available bandwidth. The physical paths corresponding to the primary and backup embedding of a virtual link $(\bar{u},\bar{v}) \in \bar{E}$ are represented by $P_{\bar{u}\bar{v}}$ and $P'_{\bar{u}\bar{v}}$, respectively.
- Backup embedding of a virtual link is disjoint from the set of physical paths used for primary embedding of the virtual links. The same disjointness principle applies for the primary embedding.
- The total cost of provisioning bandwidth in PN is minimum according to the following cost function:

$$\sum_{\forall(\bar{u},\bar{v})\in\bar{E}} \sum_{\forall(u,v)\in P_{\bar{u}\bar{v}}\cup P'_{\bar{u}\bar{v}}} C_{uv} \times b_{\bar{u}\bar{v}} \qquad (1)$$

Therefore, a solution of $1+1$-*ProViNE* will yield two disjoint embeddings of a VN request on the PN while minimizing the given cost function. Fig. 1 shows such an example with filled nodes and thickened lines marking the primary, and hollow nodes and thinner lines marking the backup embedding of a VN request on a PN.

III. ILP Formulation: *OPT-DRONE*

$1+1$-*ProViNE*'s objective is to ensure fault tolerance of a VN by providing dedicated protection to each VN element with minimal resource overhead. This ensures that a *single physical element failure* does not bring down both the primary and backup embedding of the same VN element. To find an optimal solution, we first transform the input VN (Section III-A), which ensures that the primary and the backup embedding are computed simultaneously, and then provide an ILP formulation for the optimal embedding (Section III-B).

A. Virtual Network Transformation

We formulate $1+1$-*ProViNE* as simultaneously embedding two copies of the same VN disjointly on the PN. To accomplish this goal, we first replicate the input VN $\bar{G}$ to obtain a

shadow VN, $\tilde{G} = (\tilde{V}, \tilde{E})$. $\tilde{G}$ has the same number of nodes and links as $\bar{G}$ and each shadow virtual link $(\tilde{u}, \tilde{v}) \in \tilde{E}$ has the same bandwidth requirement as the original virtual link $(\bar{u}, \bar{v}) \in \bar{E}$. We enumerate the nodes in the shadow VN $\tilde{G}$ by using the following transformation function: $\tau(\bar{u}) = \tilde{u}$.

Our transformed input now contains the graph $\hat{G} = (\hat{V}, \hat{E})$, *s.t.* $\hat{V} = \bar{V} \cup \tilde{V}$ and $\hat{E} = \bar{E} \cup \tilde{E}$. We now embed $\hat{G}$ on G in such a way that any node $u \in \bar{V}$ and any node $\tilde{u} \in \tilde{V}$ are not provisioned on the same physical node. Similar constraints apply on the virtual links as well.

B. ILP Formulation

We begin by introducing the decision variables (Section III-B1). Then we present the constraints (Section III-B2) followed by the objective function (Section III-B3).

1) Decision Variables: A virtual link is mapped to a physical path. The following decision variable indicates the mapping between a virtual link and a physical link.

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

The following decision variable represents the virtual node mapping:

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

2) Constraints:

a) Link Mapping Constraints: We ensure that every virtual link is mapped to a non-zero length physical path by (2). It also ensures that no virtual link is left unmapped. Physical link resource constraint is expressed using (3). Finally, (4) makes sure that the in-flow and out-flow of each physical node is equal except at the nodes where the endpoints of a virtual link are mapped. (4) ensures a continuous path between the mapped endpoints of a virtual link [16].

$$\forall (\hat{u}, \hat{v}) \in \hat{E} : \sum_{\forall (u,v) \in E} x_{uv}^{\hat{u}\hat{v}} \geq 1 \tag{2}$$

$$\forall (u, v) \in E : \sum_{\forall (\hat{u},\hat{v}) \in \hat{E}} x_{uv}^{\hat{u}\hat{v}} \times b_{\hat{u}\hat{v}} \leq b_{uv} \tag{3}$$

$$\forall \hat{u}, \hat{v} \in \hat{V}, \forall u \in V : \sum_{\forall v \in \mathcal{N}(u)} (x_{uv}^{\hat{u}\hat{v}} - x_{vu}^{\hat{u}\hat{v}}) = y_{\hat{u}u} - y_{\hat{v}u} \tag{4}$$

The binary nature of the virtual link mapping decision variable along with the flow constraint prevents any virtual link being mapped to more than one physical path. This restricts the link mapping to the *Multi-Commodity Unsplittable Flow Problem* [15].

b) Node Mapping Constraints: (5) and (6) ensures that a virtual node is mapped to exactly one physical node according to the given location constraints. Then (7) ensures that a physical node does not host more than one virtual node from the same virtual network request.

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

$$\forall \hat{u} \in \hat{V}, \forall u \in V : y_{\hat{u}u} \leq \ell_{\hat{u}u} \tag{6}$$

$$\forall u \in V : \sum_{\hat{u} \in \hat{V}} y_{\hat{u}u} \leq 1 \tag{7}$$

The virtual node embedding follows from the virtual link embedding since we do not have any cost associated with virtual node embedding. Therefore, the problem of coordinated node and link embedding is at least as hard as the *Multi-commodity Unsplittable Flow Problem with Unknown Sources and Destinations* [17].

c) Disjointness Constraints: We need to ensure that every virtual link in $\bar{G}$ and its corresponding virtual link in $\tilde{G}$ is embedded on node and link disjoint paths in PN. To ensure this disjointness property, we first constrain the virtual links in $\bar{G}$ and $\tilde{G}$ to be mapped on disjoint set of physical links using (8) and (9).

$$\forall (u, v) \in E : \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} x_{uv}^{\tilde{u}\tilde{v}} = 0 \text{ if } x_{uv}^{\bar{u}\bar{v}} = 1, \forall (\bar{u}, \bar{v}) \in \bar{E} \tag{8}$$

$$\forall (u, v) \in E : x_{uv}^{\bar{u}\bar{v}} = 0 \text{ if } \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} x_{uv}^{\tilde{u}\tilde{v}} > 0, \forall (\bar{u}, \bar{v}) \in \bar{E} \tag{9}$$

Then we forbid the virtual link endpoints of the primary embedding to be intermediate nodes on the path of backup embedding and vice versa using (10) and (11).

$$\forall u \in V : y_{\bar{u}u} = 0, \text{ if } \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} \sum_{\forall v \in \mathcal{N}(u)} x_{uv}^{\tilde{u}\tilde{v}} > 0 \tag{10}$$

$$\forall u \in V : \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} \sum_{\forall v \in \mathcal{N}(u)} x_{uv}^{\tilde{u}\tilde{v}} = 0, \text{ if } y_{\bar{u}u} = 1 \tag{11}$$

We also ensure that the physical paths corresponding to the virtual links in $\bar{G}$ and $\tilde{G}$ do not share any intermediate nodes. This constraint is necessary to ensure that a physical failure does not affect a primary resource and its corresponding backup resource at the same time.

$$\forall u \in V : \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} \sum_{\forall v \in \mathcal{N}(u)} x_{uv}^{\tilde{u}\tilde{v}} = 0, \text{ if } \sum_{\forall (\bar{u},\bar{v}) \in \bar{E}} \sum_{\forall v \in \mathcal{N}(u)} x_{uv}^{\bar{u}\bar{v}} > 0 \tag{12}$$

$$\forall u \in V : \sum_{\forall (\bar{u},\bar{v}) \in \bar{E}} \sum_{\forall v \in \mathcal{N}(u)} x_{uv}^{\bar{u}\bar{v}} = 0, \text{ if } \sum_{\forall (\tilde{u},\tilde{v}) \in \tilde{E}} \sum_{\forall v \in \mathcal{N}(u)} x_{uv}^{\tilde{u}\tilde{v}} > 0 \tag{13}$$

3) Objective Function: Our objective is to minimize the cost of provisioning bandwidth on the physical links. Therefore, we have the following objective function:

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

C. Hardness of $1 + 1$*-ProViNE*

As discussed earlier in Section III-B2, the coordinated node and link mapping without the disjointness constraints is at least as hard as solving the NP-Hard *Multi-commodity Unsplittable Flow Problem with Unknown Source and Destinations*. State-of-the art literature reveals that this problem is also very hard to approximate even when the source and destination of the flows are known. Recent research works have found $(7 + \epsilon)$ and $(8+\epsilon)$ approximation algorithms for line and cycle graphs, respectively [18]. However, finding constant factor approximation algorithms for general graphs still remains open [18]. With

the added disjointness constraints, the embedding problem becomes at least as hard as partitioning the PN while minimizing the cost of multi-commodity unsplittable flow with unknown sources and destinations in each of the partition. Even an easier version of this problem, *balanced graph partitioning*, is NP-hard [19] and does not have a constant factor approximation algorithm [19], [14]. This makes it challenging to devise a constant factor approximation algorithm for $1+1$-*ProViNE*.

IV. Heuristic Solution: *FAST-DRONE*

Given the NP-hard nature of the $1+1$-*ProViNE* problem, we resort to a heuristic for finding solutions within reasonable time frame. First, we restructure $1+1$-*ProViNE* for the ease of designing a heuristic, while keeping the original problem intact in its meaning (Section IV-A). Then we present our heuristic algorithm (Section IV-B) to solve the restructured problem.

A. Problem Restructuring

We reformulate $1+1$-*ProViNE* as a variant of graph partitioning problem as follows:

Given a PN $G=(V,E)$, a VN request $\bar{G}=(\bar{V},\bar{E})$, and a set of location constraints, $\mathcal{L}=\{L(\bar{u})|L(\bar{u})\subseteq V, \forall \bar{u}\in\bar{V}\}$ (Section II-B), $1+1$-*ProViNE* requires to partition the graph G into two disjoint partitions $\mathcal{P}$ and $\mathcal{Q}$ such that:

- $\forall \bar{u}\in\bar{V}$, $\mathcal{P}$ has at least one element from each $L(\bar{u})$.
- $\forall \bar{u}\in\bar{V}$, $\mathcal{Q}$ has at least one element from each $L(\bar{u})$.
- The sub-graph induced by the elements of each set $L(\bar{u})$ in $\mathcal{P}$ (and $\mathcal{Q}$) is connected.
- The sum of costs of embedding $\bar{G}$ on $\mathcal{P}$ and $\mathcal{Q}$ is minimum according to the given cost function.

The sets $\mathcal{P}$ and $\mathcal{Q}$ are disjoint partitions of G where the primary and backup resources for $\bar{G}$ can be provisioned without violating the disjointness constraint of $1+1$-*ProViNE*. An optimal $\mathcal{P}$ and $\mathcal{Q}$ will minimize the total cost of primary and backup link embedding. Such optimal $\mathcal{P},\mathcal{Q}$ will yield the optimal solution to $1+1$-*ProViNE*.

Graph partitioning, which is an NP-hard problem [19], can be reduced to the aforementioned partitioning problem by relaxing the location constraint, *i.e.,* setting each set $L(\bar{u})$, $\forall \bar{u}\in\bar{V}$, equal to V. Once we have the two partitions, embedding the virtual links inside one partition is at least as hard as solving the NP-Hard *Multi-commodity Unsplittable Flow* problem [15], since we are not allowed to embed a virtual link over multiple physical paths. In the next section, we present our heuristic algorithm based on this reformulation.

B. Heuristic Algorithm

In order to find a solution to $1+1$-*ProViNE* we need to partition the PN *s.t.* the total cost of embedding the virtual links in the partitions are minimized (Section IV-A). Our heuristic starts with a seed mapping set containing the primary and backup mapping of one virtual node and goes through the following three phases to partition the PN and embed the VN:

Node Mapping Phase: Use the seed mapping and location constraint set to find a primary and backup node embedding for the other virtual nodes. This phase yields a partial partitioning of the PN. This partial partition acts as a seed that we grow to a complete partition of the PN into two disjoint subgraphs.

Partitioning Phase: Once we have a seed primary and backup partition from the node mapping phase, we grow the seed partition to include the rest of the physical nodes into either of the partitions. At the end of this phase, all of the physical nodes are either assigned to the primary or to the backup partition.

Link Mapping Phase: In this phase, we have the virtual node mapping and the primary and backup partition of the PN as input. We embed the virtual links in these partitions separately by using the Constrained Shortest Path First algorithm.

We run this three phase algorithm for different initial seed node mapping and retain the solution with the minimum cost. To generate different seed node mappings we identify the virtual nodes that have the minimum number of elements in their location constraint set. We call these virtual nodes the *most constrained virtual nodes*. Such virtual nodes may lead to infeasible embedding if they are not embedded first, since they have the fewest options for embedding. For each of these most constrained virtual nodes $\bar{u}_c$, we take every pair of physical nodes from $\bar{u}_c$'s location constraint set $L(\bar{u}_c)$ and consider that pair as a primary and backup node embedding for $\bar{u}_c$. In this way, we generate a number of seed node mappings and execute the above-described three phase algorithm. In the rest of this section, we describe the individual phases in detail.

C. Node Mapping Phase

The node mapping phase follows a greedy approach to map the virtual nodes to it's primary and backup physical nodes, while satisfying the location constraint. In this phase, we map the virtual nodes one at a time and select them in the increasing order of their location constraint set size. The rationale for following this order is that a virtual node with fewer possible locations for mapping is more constrained. Mapping a less constrained virtual node first might lead to infeasible mapping of the more constrained virtual node(s). Node mapping is performed by the `MapVNodes` procedure (Algorithm 1). We first initialize the primary and backup node mapping sets $nmap_p$ and $nmap_s$, respectively, with the provided $seed$. Then we take one virtual node at a time according to the aforementioned order (Line 8) and iterate over its location constraint set to find the best physical node for primary mapping (Line 10 – 20) . After finding a primary mapping, we determine the corresponding backup mapping (Line 22 – 33). While considering a physical node $u\in V$ as primary mapping of a virtual node, we try to determine if u is a better choice compared to $best_u$, the best choice of physical node that we have seen so far considering the node mappings we already have. This is evaluated using the `BetterAssignment` procedure. This procedure performs the following tests in the order they are listed. We choose this order to minimize the chances of not finding a solution and to create a partition that yields a close to optimal embedding.

Infeasibility Test: Does adding u to the primary mapping makes the backup mapping impossible to be connected and

Algorithm 1 MapVNodes

1: **function** MAPVNODES($G, \bar{G}, location, seed$)
2: $nmap_p(seed.node) \leftarrow seed.primary$
3: $nmap_s(seed.node) \leftarrow seed.backup$
4: $taken(seed.primary), taken(seed.backup) \leftarrow$ **false**
5: $\mathcal{P} \leftarrow \phi$, $\mathcal{Q} \leftarrow \phi$
6: // Sequence $\bar{\mathcal{V}}$ represents virtual nodes sorted in
7: // decreasing order of location constraint set size
8: **for all** $\bar{u} \in \bar{\mathcal{V}}$ **do**
9: $best \leftarrow$ NIL
10: **for all** $c \in location(\bar{u})$ **do**
11: **if** $taken(c) =$ **false then**
12: **if** BetterAssignment($G, \mathcal{P}, \mathcal{Q}, c, best$) **then**
13: $best \leftarrow c$
14: **end if**
15: **end if**
16: **end for**
17: **if** $best \neq$ NIL **then**
18: $taken(best) \leftarrow$ **true**
19: $nmap_p(\bar{u}) \leftarrow best$, $\mathcal{P} \leftarrow \mathcal{P} \cup \{best\}$
20: **end if**
21: $best \leftarrow$ NIL
22: **for all** $c \in location(\bar{u})$ **do**
23: **if** $taken(c) =$ **false then**
24: **if** BetterAssignment($G, \mathcal{Q}, \mathcal{P}, c, best$) **then**
25: $best \leftarrow c$
26: **end if**
27: **end if**
28: **end for**
29: **if** $best \neq$ NIL **then**
30: $taken(best) \leftarrow$ **true**
31: $nmap_s(\bar{u}) \leftarrow best$, $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{best\}$
32: **end if**
33: **end for**
34: **return** $\{nmap_p, nmap_s\}$
35: **end function**

vice versa ? If the answer is *yes*, then we do not consider u for primary node mapping of the virtual node. Otherwise, we perform the next test.

Compact Mapping Test: Does considering u instead of $best_u$ in the primary (or backup) mapping decreases the mean shortest path length among the nodes currently present in the primary (or backup) mapping set ? If the answer is *yes* then u is considered to be better than $best_u$. Otherwise, we perform the next test.

Connectivity Contribution Test: Does u contribute more *connectivity* to the mapping set (primary or backup) compared to $best_u$? If the answer is *yes*, then $best_u$ is updated with u. We measure *connectivity contribution* using the following:

- Number of connected components decreased in the current mapping set if u is considered instead of $best_u$ in the mapping set.
- Number of links incident from u to the current mapping set compared to $best_u$.

We iterate over all possible physical nodes u in the location constraint set of a virtual node and find the best among them for the mapping. We do the same iteration and tests again (line 24 of Algorithm 1) to find a backup mapping for that virtual node. We repeat this procedure for all the virtual nodes and we finally obtain a primary and backup mapping of the virtual nodes, $nmap_p$ and $nmap_s$, respectively. This primary and backup mapping sets acts as seed primary and backup partitions ($\mathcal{P}_0$ and $\mathcal{Q}_0$, respectively) that we grow to full partitions in the Partitioning phase.

D. Partitioning Phase

Algorithm 2 PartitionGraph

1: **function** PARTITIONGRAPH($G, nmap_p, nmap_s$)
2: $\mathcal{P} \leftarrow \bigcup_{\forall n_p \in nmap_p} n_p$, $\mathcal{Q} \leftarrow \bigcup_{\forall n_s \in nmap_s} n_s$
3: $taken \leftarrow$ Array of size $|V|$, initialized with **false**
4: **for all** $v \in V$ **do**
5: **if** $taken(v)$ = **false then**
6: **if** IsFeasiblePartition($G, \mathcal{P}, \mathcal{Q}, v$) = **false then**
7: $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{v\}$
8: **else**
9: 1. Add u to $\mathcal{P}$ (or $\mathcal{Q}$) if u decreases more
10: shortest path cost in $\mathcal{P}$ (or $\mathcal{Q}$) than in $\mathcal{Q}$
11: (or $\mathcal{P}$)
12: 2. Break ties by adding u to $\mathcal{P}$ (or $\mathcal{Q}$) if
13: u decreases more components in $\mathcal{P}$ (or $\mathcal{Q}$)
14: than in $\mathcal{Q}$ (or $\mathcal{P}$)
15: 3. In case of another tie, assign u to $\mathcal{P}$
16: (or $\mathcal{Q}$) if u has more edges incident to $\mathcal{P}$
17: (or $\mathcal{Q}$)
18: 4. Break any remaining tie by assigning u
19: to the smaller of $\mathcal{P}$ and $\mathcal{Q}$
20: **end if**
21: **end if**
22: **end for**
23: **return** $\{\mathcal{P}, \mathcal{Q}\}$
24: **end function**

Given two seed primary ($\mathcal{P}_0$) and backup ($\mathcal{Q}_0$) partitions obtained from the node mapping phase, we partition the physical network G into two disjoint partitions $\mathcal{P}$ and $\mathcal{Q}$ for the primary and backup embeddings of the virtual network, respectively. The partitioning process is performed using the `PartitionGraph` procedure (Algorithm 2). We consider the physical nodes that are not already assigned to any of the partitions (line 4 – 5) one at a time, and perform the following tests in the order they are listed. Such order is chosen for similar reasons as discussed in the node mapping phase.

Infeasibility Test: Does adding u to $\mathcal{P}$ makes the partition $\mathcal{Q}$ impossible to be connected (line 6)? If the answer is *yes*, then we do not consider u for $\mathcal{P}$, rather we add u to $\mathcal{Q}$.

Compact Partition Test: Does including u to $\mathcal{P}$ reduce shortest path length more than that reduced when u is added

to $\mathcal{Q}$ (9 – 11) ? If the answer is *yes*, then add u to $\mathcal{P}$, otherwise evaluate the next test.

Connectivity Contribution Test: We determine whether a candidate node $u \in V$ contributes more connectivity to $\mathcal{P}$ or to $\mathcal{Q}$ by evaluating the following:

- Does including u in $\mathcal{P}$ reduces more the number of components compared to adding u to $\mathcal{Q}$ (line 12 – 14)? If the answer is *yes*, then u is added to $\mathcal{P}$, otherwise we evaluate the next criterion.
- Does the candidate node $u \in V$ has more physical links going to $\mathcal{P}$ compared to $\mathcal{Q}$ (line 15 – 17) ? If the answer is *yes* then u is added to $\mathcal{P}$, otherwise u is added to $\mathcal{Q}$.

Load Balancing Test: If all the previous tests fail to assign a $u \in V$ to either $\mathcal{P}$ or $\mathcal{Q}$, then we assign u to $\mathcal{P}$ if $|\mathcal{P}| < |\mathcal{Q}|$, otherwise u is assigned to $\mathcal{Q}$.

`PartitionGraph` procedure iterates over all the unassigned physical nodes $u \in V$ and assigns u to either $\mathcal{P}$ or to $\mathcal{Q}$. At the end of this phase, we have two disjoint partitions, each of them has at least one node from each of the location constraint sets. Therefore, this partitioning conforms to the conditions as described in Section IV-A.

E. Link Mapping Phase

Given the two disjoint partitions, $\mathcal{P}$ and $\mathcal{Q}$, and the node mappings for the virtual nodes in each partition, we use constrained shortest path first algorithm to map a virtual link to a physical path inside a partition. Application of shortest path based algorithms are common practice in cases when virtual links cannot be split and embedded on multiple physical paths [20]. We also have this constraint in $1 + 1$-*ProViNE*.

All of the three phases are combined and presented in the `FAST-DRONE` procedure (Algorithm 3). Line 2 corresponds to the node mapping phase, Line 3 represents the partition growing phase, and finally lines 4 and 5 gives us the link mappings.

Algorithm 3 FAST-DRONE

1: **function** FAST-DRONE($G, \bar{G}, location$)
2: $\{nmap_p, nmap_s\} \leftarrow$ MapVNodes($G, \bar{G}, location$)
3: $\{\mathcal{P}, \mathcal{Q}\} \leftarrow$ PartitionGraph($G, nmap_p, nmap_s$)
4: $emap_p \leftarrow$ EmbedAllVLinks($G, \bar{G}, \mathcal{P}, nmap_p$)
5: $emap_s \leftarrow$ EmbedAllVLinks($G, \bar{G}, \mathcal{Q}, nmap_s$)
6: Compute $embedding\_cost$ from $emap_p$ and $emap_s$
7: **return** $\{nmap_p, emap_p, nmap_s, emap_s, cost\}$
8: **end function**

F. Running Time Analysis

Before going to the analysis we first introduce the following notations:

- n = Number of vertices in the physical network
- n' = Number of vertices in the virtual network
- m = Number of edges in the physical network
- m' = Number of edges in the virtual network
- σ = Maximum size of a location constraints set for any virtual node
- δ = Maximum degree of a physical node

We analyze the running time of `FAST-DRONE` procedure by analyzing the running time for each of the phases as follows:

Node Mapping Phase: Sorting the virtual nodes requires $O(n' \log n')$ time. Then for each of these n' virtual nodes, we traverse its location constraint set, which can have $\leq \sigma$ elements. For each of these $O(\sigma)$ nodes, we perform: (i) feasibility check (ii) compute the reduction in shortest path length (iii) compute the decrease in number of components and (iv) compute the number of edges incident form the candidate physical node to the current set of mappings. (i) can be accomplished in $O(n + m)$ time by simply keeping a disjoint set data structure with $O(n)$ elements, and perform union operation on the data structure. (ii) can take up to $O(n'^3)$ time. (iii) can be performed in $O(n + m)$ time in the worst case with a disjoint set data structure. Finally for step (iv), the number of edges incident from a candidate physical node to a mapping set can be computed in $O(\delta)$ time. Therefore, the mapping phase runs in $O(n'\sigma(n + m + \delta + n'^3))$ time.

Graph Partitioning Phase We iterate over $O(n)$ unassigned physical nodes and perform similar steps as in the node mapping phase. Therefore, the time complexity of each iteration is the same as the four tasks described for the node mapping phase. Hence, partitioning the graph requires $O(n(n + m + \delta + n'^3))$ time.

Link Mapping Phase For the link mapping phase, we compute shortest path between the mapped nodes for each of the m' virtual links using Dijkstra's shortest path algorithm. This requires $O(m'm \log n)$ time in total.

Overall, the running time of the proposed heuristic is: $O((n'\sigma + n)(n + m + \delta + n'^3) + m'm \log n)$.

V. Performance Evaluation

We evaluate the proposed solutions for $1 + 1$-*ProViNE* through extensive simulation. We perform simulations using randomly generated network topologies with various connectivity levels. Section V-A discusses the simulation setup in detail and Section V-B defines the performance metrics. Our evaluation is focused on the following aspects: i) comparing the performance of the proposed heuristic (*FAST-DRONE*) with the optimal (*OPT-DRONE*) (Section V-C), ii) analyze the impact of physical network connectivity levels (Section V-D), iii) scalability of *FAST-DRONE* (Section V-E), and iv) comparing *DRONE* with PAR from [11] (Section V-F).

A. Simulation Setup

1) Testbed: We have implemented *OPT-DRONE*, the ILP based optimal solution for $1 + 1$-*ProViNE* using IBM ILOG CPLEX 12.5 C++ libraries. The heuristic is also implemented in C++. We exploited inherent parallelism in the heuristic and implemented it as a multi-threaded program. Both the hueristic and CPLEX solution were run on a machine with hyper-threaded 8×10 core Intel Xeon E7-8870 CPU and 1TB of memory. Both CPLEX and heuristic implementations spawn upto 160 threads to saturate all the processing cores during their executions.

2) Physical Network Topology: We have generated random topologies by varying the number of nodes from 50 to 200 in increments of 25, and varying the Link-to-Node Ratio (LNR) from 1.2 to 2.2 in steps of 0.1. We used PNs with different LNR to study the impact of physical network's connectivity on *FAST-DRONE*'s performance.

3) Virtual Network Topology: We generated three types of VN requests: ring, star and randomly connected graphs with ≤ 16 nodes evaluate *FAST-DRONE*'s performance. Since our focus is to study the resource efficiency given that a feasible embedding exists, we set the bandwidth requirement in VN and bandwidth capacity in PN to make sure at least one feasible embedding exists.

B. Performance Metrics

1) Embedding Cost: The embedding cost is the cost of provisioning bandwidth for the virtual links and their backups, computed using (1).

2) Execution Time: The time required for an algorithm to find the solution to $1+1$-*ProViNE*.

C. Comparison between OPT-DRONE and FAST-DRONE

In this section, we present the results on how much extra resource is provisioned by *FAST-DRONE* compared to *OPT-DRONE*. This extra resource usage is measured as the ratio of *FAST-DRONE*'s cost to *OPT-DRONE*'s cost since our cost function is proportional to the total bandwidth allocated for the VN. Fig. 2 shows the CDF of cost ratio for different types of VN requests as well as the CDF for all types combined. This plot shows that about 70% VN requests are embedded by *FAST-DRONE* with at most 15% extra resources, while 90% VN requests are embedded with at most 23% extra resources. On average, *FAST-DRONE* provisions 14.3% extra resource compared to *OPT-DRONE* over all VN request types.

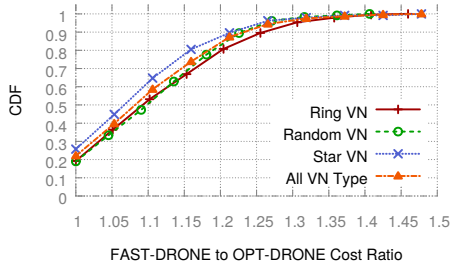


Fig. 2. Comparison between *OPT-DRONE* and *FAST-DRONE*

D. Impact of Physical Network Connectivity

In this section we present results on how the connectivity level of the underlying PN impacts the performance of *FAST-DRONE*. We varied the LNR of the generated physical networks from 1.2 to 2.2 in increments of 0.1 and measured the mean *FAST-DRONE* to *OPT-DRONE* cost ratio for each case. Fig. 3 shows the mean cost ratio with 5-th and 95-th percentile error bars against different LNRs. This plot gives us an idea about a good operating region for *FAST-DRONE*. As we can see, *FAST-DRONE* allocates about 15% extra resources compared to the optimal solution for PNs having an LNR ≤ 1.8. For higher LNR, the increased path diversity may lead to more sub-optimal solution since the heuristic does not explore all paths to keep the running time fast.

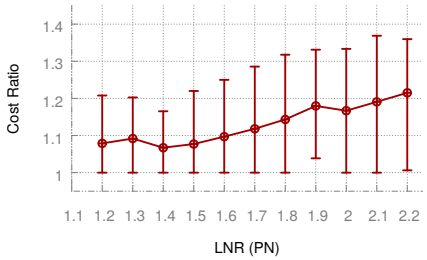


Fig. 3. Impact of PN Connectivity

E. Scalability of Heuristic

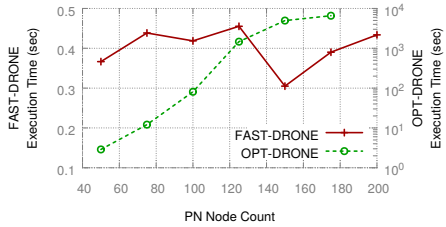


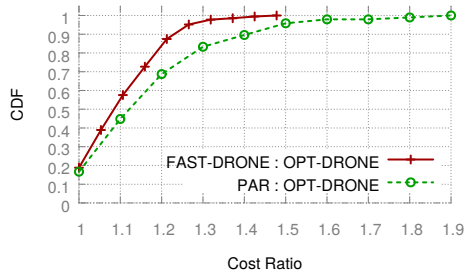
Fig. 4. Comparison of Execution time

To demonstrate the scalability of *FAST-DRONE* we show the execution times of *FAST-DRONE* and *OPT-DRONE* on same problem instances in Fig. 4. As it turns out, *FAST-DRONE* takes less than 500ms on average over all test cases, whereas *OPT-DRONE* takes more than 2-minutes to run on average on the smallest PN instance. For larger instances, the execution time exponentially increases for *OPT-DRONE* and becomes in the order of hours (*e.g.*, about 110 minutes on average for a 175 node PN). We found *FAST-DRONE* to be 200x – 1200x faster than *OPT-DRONE* depending on the problem instance. With our current setup *OPT-DRONE* did not scale beyond PNs with more than 175 nodes.

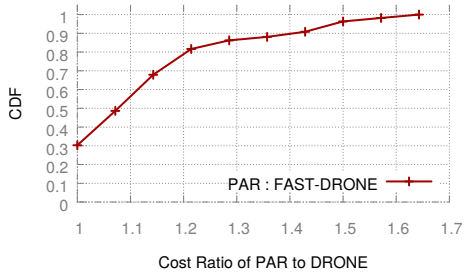
F. Comparison with PAR [11]

PAR [11] is a greedy heuristic for embedding a VN request on a PN with $1+1$-protection. PAR maximizes the probability of accepting a VN request by first embedding the virtual nodes on physical nodes with higher residual node capacities. After node embedding, PAR embeds the virtual links using a modified version of Suurballe's algorithm [21]. In our case, we do not have node capacities. Therefore, we first implemented PAR by randomly mapping a virtual node $\bar{u} \in \bar{V}$ to a physical node within its location constraint set $L(\bar{u})$. However, such random mapping lead to infeasible solutions almost all the time. Then we used our proposed `MapVNodes` (Algorithm 1) procedure to map the virtual nodes. The link embedding was done exactly the same way as described in [11]. It is worth noting that even after the modification in the node embedding, PAR could only find solutions for $\approx 12\%$ test cases in our simulation setting.

We first compare how much extra resource is allocated by PAR and *FAST-DRONE* compared to *OPT-DRONE*. For this comparison, we compute the ratio of costs (cost is computed using (1)) between PAR and *OPT-DRONE*, and *FAST-DRONE* and *OPT-DRONE*. Fig. 5(a) shows the CDF of these cost ratios. This plot shows that in 90% cases, PAR allocates up to 40% additional resources compared to the optimal, whereas, *FAST-DRONE* allocates up to 23% additional resources. On average, the amount of extra resource allocated compared to the optimal is 25% and 14.3% for PAR and *FAST-DRONE*, respectively. We also compute the ratio of PAR's cost to that of *FAST-DRONE* and plot the CDF in Fig. 5(b) to see how much *FAST-DRONE* improves over PAR. This plot shows that PAR never performs better than *FAST-DRONE* and allocates up to 40% extra resources compared to *FAST-DRONE* at the 90-th percentile. On average, we found PAR to allocate 17.5% additional resources compared to *FAST-DRONE*.



(a) Comparison with *OPT-DRONE*



(b) Cost Ratio of PAR to *FAST-DRONE*

Fig. 5. Comparison between *FAST-DRONE* and PAR [11]

VI. Related Works

A. VNE with Dedicated Protection

A very recent work by Ye *et al.* addresses a similar problem of providing dedicated protection for VNE with different goals and a different input model [11]. They formulate the problem using a Quadratic Integer Program in contrast to our ILP formulation. However, the major difference between their approach and ours is the objective. [11] focused on increasing the VN request acceptance ratio over time, whereas we focus on minimizing the resource allocation cost for embedding virtual networks. Ye *et al.* proposed a greedy heuristic based on the node resource requirement, which is not suitable for our case since we do not consider any node capacity or node embedding cost. Lastly, [11] does not consider the location constraint, which is an important constraint in our case. Another closely related work is from Jiang *et al.* [22]. They propose a backup scheme where the backup virtual nodes are disjoint from the primary virtual nodes. However, the backup nodes can share a single physical node and therefore exhibit lesser survivability compared to $1+1$-*ProViNE*. [22] also does not provision full backup of the virtual links, rather provisions some backup paths that can be used to route between the virtual nodes during a single physical resource failure. There are also a number of other research works in the literature that address different aspects of SVNE [8], [11], [23], [24], [25], [26], [27], [28], [29]. However, none of the existing research except [11] addresses the issue of providing a dedicated $1+1$-protection.

B. Unsplittable Flow and Partitioning

The root of providing dedicated protection for virtual network embedding goes back to combinatorial optimization problems such as graph partitioning and multi commodity unsplittable flow problem [15], [17]. Relevant literature shows that they are computationally hard to solve. Finding a constant factor approximation algorithm for these problems for generic graphs is still open [18], [19], [30]. The best known approximation ratio for graph partitioning is not constant, rather it is a poly-logarithm function of the number of nodes [19]. On the other hand, the unsplittable flow problem with known sources and destinations has $(7+\epsilon)$ and $(8+\epsilon)$ approximation ratio for simple line graph and cycle graph, respectively. Without known sources for the commodity and the flow destinations, this problem is even harder to solve.

VII. Conclusion and Future Work

In this paper we have studied $1+1$ **Pro**tected **Vi**rtual **N**twork **E**mbedding ($1+1$-*ProViNE*) problem that embeds a VN on a PN while ensuring dedicated backup for each virtual node and link. We presented *DRONE*, a suite of solutions for $1+1$-*ProViNE*. We devised an ILP based optimal solution (*OPT-DRONE*) as well as a heuristic (*FAST-DRONE*) to tackle the computational complexity. Trace driven simulations show that *FAST-DRONE* can solve $1+1$-*ProViNE* in a reasonable time frame with only 14.3% extra resources on average compared to *OPT-DRONE*.

We believe that $1+1$-*ProViNE* will open up new avenues for future research. Among the possibilities we intend to investigate the problem of providing mixed backup scheme for the VNs. A mixed backup scheme consists of providing both shared and dedicated backup to the VN elements based on the SP's request. A mixed backup scheme can enable the SPs to have dedicated protection for critical network paths while shared protection for best effort network paths for instance.

Acknowledgement

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] "Amazon Virtual Private Cloud," http://aws.amazon.com/vpc/.

[2] "OpenFlow-enabled Transport SDN," https://www.opennetworking.org/images/stories/downloads/sdn-resources/solution-briefs/sb-of-enabled-transport-sdn.pdf.

[3] "Global Transport SDN Prototype Demonstration," https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/oif-p0105_031_18.pdf.

[4] M. Chowdhury and R. Boutaba, "A survey of network virtualization," *Computer Networks*, vol. 54, no. 5, pp. 862–876, 2010.

[5] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," in *Proc. of IEEE INFOCOM*, 2009, pp. 783–791.

[6] A. Razzaq and M. S. Rathore, "An approach towards resource efficient virtual network embedding," in *Evolving Internet (INTERNET), 2010 Second International Conference on*. IEEE, 2010, pp. 68–73.

[7] J. F. Botero, X. Hesselbach, M. Duelli, D. Schlosser, A. Fischer, and H. De Meer, "Energy efficient virtual network embedding," *Communications Letters, IEEE*, vol. 16, no. 5, pp. 756–759, 2012.

[8] M. R. Rahman and R. Boutaba, "SVNE: Survivable Virtual Network Embedding Algorithms for Network Virtualization," *Network and Service Management, IEEE Transactions on*, vol. 10, no. 2, pp. 105–118, 2013.

[9] S. Ramamurthy and B. Mukherjee, "Survivable WDM mesh networks. Part I-protection," in *INFOCOM'99.*, vol. 2. IEEE, 1999, pp. 744–751.

[10] P. A. Bonenfant, "Optical layer survivability: a comprehensive approach," in *Optical Fiber Communication Conference and Exhibit, 1998. OFC'98., Technical Digest*. IEEE, 1998, pp. 270–271.

[11] Z. Ye, A. N. Patel, P. N. Ji, and C. Qiao, "Survivable virtual infrastructure mapping with dedicated protection in transport software-defined networks [invited]," *Journal of Optical Communications and Networking*, vol. 7, no. 2, pp. A183–A189, 2015.

[12] P. Gill, N. Jain, and N. Nagappan, "Understanding network failures in data centers: measurement, analysis, and implications," in *ACM SIGCOMM CCR*, vol. 41, no. 4. ACM, 2011, pp. 350–361.

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

[14] R. Krauthgamer, J. S. Naor, and R. Schwartz, "Partitioning graphs into balanced components," in *Proc. of SODA*, 2009, pp. 942–949.

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

[16] M. Melo, J. Carapinha, S. Sargento, L. Torres, P. N. Tran, U. Killat, and A. Timm-Giel, "Virtual network mapping–an optimization problem," in *Mobile Networks and Management*. Springer, 2012, pp. 187–200.

[17] C. Chekuri, S. Khanna, and F. B. Shepherd, "The all-or-nothing multicommodity flow problem," in *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. ACM, 2004, pp. 156–165.

[18] P. Bonsma, J. Schulz, and A. Wiese, "A Constant-Factor Approximation Algorithm for Unsplittable Flow on Paths," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 767–799, 2014.

[19] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Engineering: Selected Results and Surveys, LNCS 9220*. Springer-Verlag, 2015 (in press).

[20] Y. Zhu and M. H. Ammar, "Algorithms for assigning substrate network resources to virtual network components." in *INFOCOM*, vol. 1200, no. 2006, 2006, pp. 1–12.

[21] J. W. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974.

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

[23] T. Guo, N. Wang, K. Moessner, and R. Tafazolli, "Shared backup network provision for virtual network embedding," in *Proc. of IEEE ICC*, 2011, pp. 1–5.

[24] 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 *Proc. of IEEE GLOBECOM*, 2010, pp. 1–6.

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

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

[27] M. G. Rabbani, M. F. Zhani, and R. Boutaba, "On achieving high survivability in virtualized data centers," *IEICE Transactions on Communications*, vol. 97, no. 1, pp. 10–18, 2014.

[28] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *Proc. of IEEE INFOCOM 2014, Toronto, Canada*. IEEE, 2014, pp. 289–297.

[29] M. M. A. Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "SiMPLE: Survivability in multi-path link embedding," in *ACM/IEEE/IFIP CNSM 2015, Barcelona, Spain, November 9-13, 2015*. IEEE Computer Society, 2015, pp. 210–218.

[30] M. Skutella, "Approximating the Single Source Unsplittable Min-cost Flow Problem," *Mathematical Programming*, vol. 91, no. 3, pp. 493–514, 2002.