Dedicated Protection for Survivable Virtual Network Embedding

Shihabur Rahman Chowdhury, *Student Member, IEEE*, Reaz Ahmed, Md Mashrur Alam Khan, Nashid Shahriar, Raouf Boutaba, *Fellow, IEEE*, Jeebak Mitra, and Feng Zeng

***Abstract*—Network virtualization is enabling infrastructure providers (InPs) to offer new services to 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 backup resource provisioning is to reserve a mutually exclusive 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* problem. *DRONE* includes an Integer Linear Programming (ILP) formulation for optimal solution (*OPT-DRONE*) and a heuristic (*FAST-DRONE*) to tackle the computational complexity of 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 200 – 1200 times faster. Simulation results also show that *FAST-DRONE* can accept 4 times more VN requests on average compared to the state-of-the-art solution for providing dedicated protection to VNs.**

***Index Terms*—Survivable Virtual Network Embedding, Network survivability and resilience, Optimization techniques**

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 slices of their networking infrastructure to 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 solutions.

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 number 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 co-existence of multiple customers with full-fledged VNs instead of traditional end-to-end connectivity, while each customer having full control over its virtual slice. These customer VNs carry high volumes of traffic at high speed, 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 InPs may need to provision dedicated backup resources for the entire VN topology, which can be used for immediate recovery from a physical failure [11]. 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 ob-

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.

Shihabur Rahman Chowdhury, Reaz Ahmed, Nashid Shahriar and Raouf Boutaba are with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada (email: sr2chowdhury@uwaterloo.ca, r5ahmed@uwaterloo.ca, nshahria@uwaterloo.ca, rboutaba@uwaterloo.ca)

Md Mashrur Alam Khan is with Cisco Systems, Ottawa ON K2K 3E8, Canada (email: maalamk3@cisco.com)

Jeebak Mitra is with Huawei Technologies Canada Research Center, Ottawa, ON K2K 3J1, Canada (email: jeebak.mitra@huawei.com)

Feng Zeng is with Huawei Technologies Co., Ltd., Chengdu, Sichuan Province, P.R. China (email: zengfeng137140@huawei.com)

jective of minimizing resource provisioning cost in the PN, while protecting each node and link in a VN request with dedicated backup resource in PN. The primary and backup embeddings need to be disjoint to ensure that a single physical node failure does not affect both the primary and the backup. If the primary embedding of a VN is affected by a physical node failure, the SP should not incur a significant service disruption typical when migrating the whole or part of the VN to the backup. Indeed, during a single physical node failure, the disjoint primary and backup embeddings both accessible to the SP enables the InP to instantly switch traffic to the backup embedding without requiring any re-embedding decision. This capability of instantly switching traffic to the backup facilitates fast recovery within tens of milliseconds, which is a typical SLA between an OTN provider and customer [9], [10].

A major challenge in solving $1+1$-*ProViNE* is to find the primary and backup embedding at the same time. Relevant literature [12] 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 [13], [14], 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 [12]. 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 [15], [16].

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 14.3% extra resources on average compared to *OPT-DRONE*, while executing 200 – 1200 times faster. Simulation results also show that *FAST-DRONE* outperforms state-of-the-art solution for providing dedicated protection to VNs [12] and accepts 4 times more VN requests on average.

This paper extends our initial work presented in [17] on several aspects. First, we provide a guideline on how to parallelize *FAST-DRONE* to leverage multiple CPU cores on a multi-core machine. Second, we perform more extensive simulations and present results on the steady state behavior of *FAST-DRONE* and compare to the state-of-the-art solution [12]. We also extend the micro-benchmarking performance analysis by demonstrating the impact of VN topology type on *OPT-DRONE* and *FAST-DRONE*'s performance, and including results on mean embedding path lengths obtained by *OPT-DRONE* and *FAST-DRONE*. Finally, we present in-depth discussion on the subtle differences of $1+1$-*ProViNE* to various related problems in SVNE literature.

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*. Then we discuss related works from the literature in Section VI. Finally, we conclude with some future research directions in Section VII.

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

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:

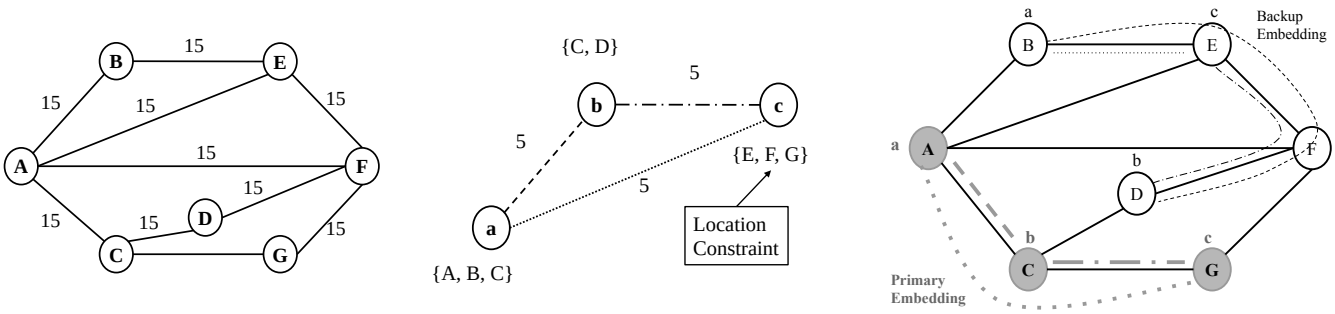
$$\ell_{\bar{u}u} = \begin{cases} 1 & \text{if } \bar{u} \in \bar{V} \text{ can be provisioned on } u \in V, \\ 0 & \text{otherwise.} \end{cases}$$

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 disjointedness principle applies for the primary embedding.
- The total cost of provisioning bandwidth in PN is minimum according to the following cost function:

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



(a) Physical Network (b) Virtual Network Request (c) Embedding by *DRONE*

Fig. 1. Example embedding with *DRONE*

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 (1). 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 glossary of key notations used in the ILP formulation is provided in Table I.

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.

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

TABLE I
Summary of Key Notations

$G = (V, E)$	Physical Network
b_{uv}	Residual Bandwidth capacity of physical link $(u, v) \in E$
C_{uv}	Cost of allocating unit bandwidth on physical link $(u, v) \in E$ for provisioning a virtual link
$\bar{G} = (\bar{V}, \bar{E})$	Virtual Network Request
$b_{\bar{u}\bar{v}}$	Bandwidth requirement of virtual link $(\bar{u}, \bar{v}) \in \bar{E}$
$\mathcal{L}(\bar{u})$	Location constraint set for virtual node $\bar{u} \in \bar{V}$
$\ell_{\bar{u}u} \in \{0, 1\}$	$\ell_{\bar{u}u} = 1$ if $u \in \mathcal{L}(\bar{u})$, $u \in V, \bar{u} \in \bar{V}$
$\tilde{G} = (\tilde{V}, \tilde{E})$	Replica of virtual network request $\bar{G}$
$\hat{G} = (\hat{V}, \hat{E})$	Transformed virtual network request, $\hat{G} = \bar{G} \cup \tilde{G}$
$x_{uv}^{\hat{u}\hat{v}} \in \{0, 1\}$	$x_{uv}^{\hat{u}\hat{v}} = 1$ if $(u, v) \in E$ is on the embedded physical path for $(\hat{u}, \hat{v}) \in \bar{E}$
$y_{\hat{u}u} \in \{0, 1\}$	$y_{\hat{u}u} = 1$ if $\hat{u} \in \hat{V}$ is mapped to $u \in V$

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 [18].

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

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

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

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* [16].

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*.

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 disjointedness 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).

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

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

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).

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

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

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.

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

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

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 disjointedness 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 $(2+\varepsilon)$ approximation algorithms for line and cycle graphs, respectively [19]. However, finding constant factor approximation algorithms for general graphs still remains open [20]. With the added mutual exclusion 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 [21] and does not have a constant factor approximation algorithm [15], [21]. This makes it challenging to devise an 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, *i.e.*, *FAST-DRONE*, for finding solutions within a 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 in detail (Section IV-B, Section IV-C, Section IV-D, and Section IV-E) to solve the restructured problem. We also analyze the running time of *FAST-DRONE* (Section IV-F) and provide a guideline on parallel implementation of *FAST-DRONE* on a multi-core machine (Section IV-G).

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 (1).

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 disjointedness 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 [21], 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 [16], 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 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

Algorithm 1 MapVNodes

```
1: function MapVNodes(G, Ḡ, location, seed)
2:     nmap_p(seed.node) ← seed.primary
3:     nmap_s(seed.node) ← seed.backup
4:     taken(seed.primary), taken(seed.backup) ← false
5:     P ← φ, Q ← φ
6:     // Sequence V̄ represents virtual nodes sorted in
7:     // decreasing order of location constraint set size
8:     for all ū ∈ V̄ do
9:         best ← NIL
10:        for all c ∈ location(ū) do
11:            if taken(c) = false then
12:                if BetterAssignment(G, P, Q, c, best) then
13:                    best ← c
14:                end if
15:            end if
16:        end for
17:        if best ≠ NIL then
18:            taken(best) ← true
19:            nmap_p(ū) ← best, P ← P ∪ {best}
20:        end if
21:        best ← NIL
22:        for all c ∈ location(ū) do
23:            if taken(c) = false then
24:                if BetterAssignment(G, Q, P, c, best) then
25:                    best ← c
26:                end if
27:            end if
28:        end for
29:        if best ≠ NIL then
30:            taken(best) ← true
31:            nmap_s(ū) ← best, Q ← Q ∪ {best}
32:        end if
33:    end for
34:    return {nmap_p, nmap_s}
35: end function
```

$best_u$ 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

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. `Mean-SP-Reduction` procedure computes the reduction in mean shortest path length within a partition if a candidate node is added to that partition.

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 14 – 16)? If the answer is *yes*, then u is added to $\mathcal{P}$, otherwise we evaluate the next criterion. `Components-Reduced` procedure computes the reduction in number of components if a candidate node is added to a partition.
- Does the candidate node $u \in V$ has more physical links going to $\mathcal{P}$ compared to $\mathcal{Q}$ (line 19 – 24)? If the answer is *yes* then u is added to $\mathcal{P}$, otherwise u is added to $\mathcal{Q}$. `Num-Cut-Edges` procedure computes the number of physical links from a candidate node to a partition.

Algorithm 2 PartitionGraph

```
1: function PARTITIONGRAPH(G, nmap_p, nmap_s)
2:     P ← ⋃_{∀n_p∈nmap_p} n_p, Q ← ⋃_{∀n_s∈nmap_s} n_s
3:     taken ← Array of size |V|, initialized with false
4:     for all v ∈ V do
5:         if taken(v) = false then
6:             if IsFeasiblePartition(G, P, Q, v) = false then
7:                 Q ← Q ∪ {v}
8:             else
9:                 x ← Mean-SP-Reduction(G, P, Q, u)
10:                y ← Mean-SP-Reduction(G, Q, P, u)
11:                if x > y then
12:                    P ← P ∪ {v}
13:                else if x = y then
14:                    δ_p ← Components-Reduced(G, P, v)
15:                    δ_s ← Components-Reduced(G, Q, v)
16:                    if δ_p > δ_s then
17:                        P ← P ∪ {v}
18:                    else if δ_p = δ_s then
19:                        cut_p ← Num-Cut-Edges(G, P, v)
20:                        cut_s ← Num-Cut-Edges(G, Q, v)
21:                        if cut_p > cut_s then
22:                            P ← P ∪ {v}
23:                        else if cut_p < cut_s then
24:                            Q ← Q ∪ {v}
25:                        else
26:                            Assign u to the smaller partition
27:                        end if
28:                    else
29:                        Q ← Q ∪ {v}
30:                    end if
31:                end if
32:            end if
33:        end if
34:    end for
35:    return {P, Q}
36: end function
```

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 [22]. 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 line 4,5 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: $O((n'\sigma+n)(n+m+\delta+n'^3)+m'm\log n)$.

G. Parallel Implementation of FAST-DRONE

The proposed heuristic, *i.e.*, *FAST-DRONE* can be implemented as a multi-threaded program to utilize multiple CPU cores. We observed in Algorithm 1 that embedding of the first virtual node to its primary and backup physical nodes has the most impact on the subsequent embeddings. To mitigate this impact, we can consider all possible initial primary/backup embedding combinations for the most constrained virtual node, *i.e.*, the virtual node with the smallest location constraint set and then run the rest of the heuristic. We can parallelize this process by executing each run of the heuristic with one combination of primary/backup embedding of the first virtual node on a separate thread running on one CPU core. After the parallel executions finish, we can choose the best embedding among all the parallel executions.

V. Performance Evaluation

We evaluate the proposed solutions for $1+1$-*ProViNE* through extensive simulations. We perform simulations using both randomly generated network topologies with various connectivity levels and a real ISP topology. Section V-A describes the simulation setup in detail and Section V-B defines the performance metrics. Then we present our evaluation results focusing on the following two aspects: first, we perform micro-benchmarking of our solutions and compare with PAR [12], a recent work on survivable virtual infrastructure embedding with dedicated resources. For the micro-benchmarking scenario, we consider each VN embedding request in isolation and assume that the VN can always be embedded on the PN. Under these assumptions, we measure the resource efficiency of our proposed solutions and compare our results with that of [12]. Second, we perform steady state analysis of *FAST-DRONE* and compare with that of PAR [12]. The steady state analysis considers VN arrivals and departures over a period of time and also considers the possibility of failing to embed a VN request on the PN. The steady state analysis provides valuable insight on the number of accepted VNs and the physical resource utilization in a longer time frame.

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 implemented the heuristic as a multi-threaded program by following the guidelines in Section IV-G. Both the heuristic and CPLEX solutions were run on a machine with hyper-threaded 8×10 core Intel Xeon E7-8870 CPU and 1TB of memory. Both the CPLEX and heuristic implementations spawn up to 160 threads to saturate all the processing cores during their executions. We have developed an in-house discrete event simulator that simulates the arrival and departure of VNs for the steady state scenario.

2) Physical Network Topology: We have generated random topologies for the micro-benchmarking scenario by varying the number of physical 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. For the steady state scenario, we have used the topology of a large ISP (AS-6461) from the Rocketfuel ISP topology dataset [23] containing 141 nodes and 374 links. We used random integers uniformly distributed between 35000Mbps and 40000Mbps as link bandwidth, since link bandwidth is not specified in [23].

3) Virtual Network Topology: We generated three types of VN topologies for the micro-benchmarking scenario: ring, star and randomly connected graphs with ≤ 16 nodes to study the impact of different types of VNs on *FAST-DRONE*'s performance. As stated earlier, the micro-benchmarking scenario evaluates the resource efficiency of the algorithms given that the physical network's capacity and the virtual nodes' location constraints yield a feasible embedding. In order to ensure that the VNs have at least one feasible embedding, we have iteratively grown the VNs from an input PN. We first start with an empty VN and add exactly one virtual node and some virtual links to the VN in each iteration. During an iteration, we first randomly select a physical node and its neighbor as the primary and backup embedding for the new virtual node being added. Then we find paths in the PN from these primary and backup embeddings to the primary and backup embeddings of the existing virtual nodes, respectively. These existing virtual nodes are selected according to the type of VN (*e.g.*, ring, star, random) that we are trying to grow. Each pair of these newly found primary and backup paths in the PN, correspond to a virtual link between the virtual node being added and the selected virtual nodes from the already grown VN. While computing the paths, we maintain the disjointedness invariant of $1+1$-*ProViNE* as described in Section II-C. This procedure ensures that the grown VN has at least one valid embedding on the PN. This process is continued until a VN of a desired size is found.

For the steady state scenario, we generated VNs with random connectivity. We set the VN connectivity level at 50% and vary the number of virtual nodes from 4 to 8. The virtual link bandwidth requirements are integers chosen uniformly between 12000Mbps and 15000Mbps. The VN arrival rate follows a Poisson distribution with a mean from 4 to 10 VNs per 100 time units. The VN life time is exponentially distributed with a mean of 1000 time units. These parameters have been chosen in accordance with the standards used in the related literature [5], [24]. For the location constraint of a virtual node, we randomly choose a physical node and all the nodes reachable within a 3-hop radius. We choose a larger location constraint set in this case to reduce the chances of VN embedding failure due to limited number of locations.

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*.

3) Mean embedding path length: For a given VN request, this is the mean of the physical path lengths corresponding to the virtual link embeddings.

4) Acceptance Ratio: Acceptance ratio is the fraction of VN requests that have been successfully embedded on the PN over all the VN requests.

5) Utilization: We compute the utilization of a physical link as the ratio of total bandwidth allocated to the embedded virtual links to that physical link's capacity.

C. Micro-benchmarking Results

Our micro-benchmarking evaluation scenario focuses on the following aspects: i) comparing the resource efficiency of the proposed heuristic (*FAST-DRONE*) with that of the optimal (*OPT-DRONE*) (Section V-C1), ii) analyzing the impact of VN topology type (Section V-C2), iii) analyzing the impact of physical network connectivity levels (Section V-C3), iv) demonstrating the scalability of *FAST-DRONE* (Section V-C4), and v) comparing *DRONE* with PAR [12] (Section V-C5).

1) 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 Cumulative Distribution Function (CDF) of cost ratio for different types of VN requests as well as the CDF for all types combined. A point (x, y) on this curve gives us the fraction y of total VN requests with cost ratio $\leq x$. 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 compared to *OPT-DRONE*. On average this extra resource provisioning is 14.3% over all VN request types.

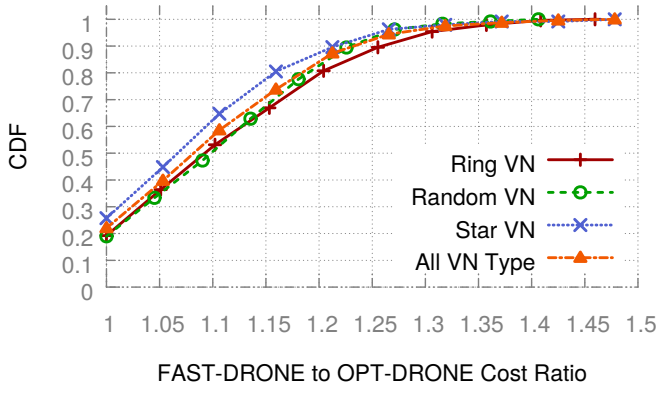
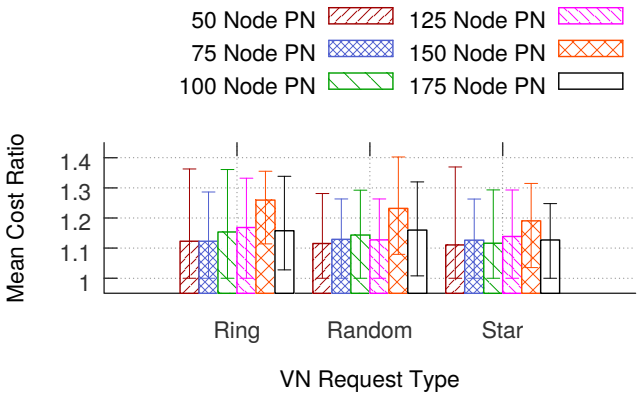
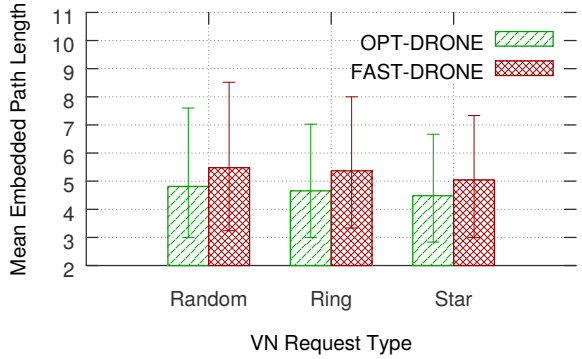


Fig. 2. Comparison between *OPT-DRONE* and *FAST-DRONE*

2) Impact of VN Request Type: Fig. 3(a) presents result for the cost ratio of different types of VN requests on different sizes of PNs. A take away from this result is that *FAST-DRONE* performs better for star VN topologies compared to ring and randomly connected VN topologies. The reason behind such



(a) Impact on Cost Ratio



(b) Impact on Mean Embedded Path Length

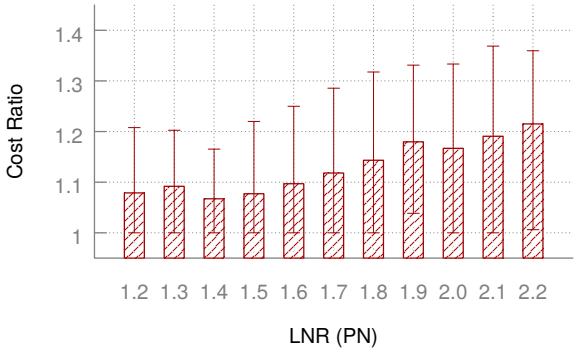
Fig. 3. Impact of VN request type

behavior is that only the center node in a star topology imposes high disjointedness requirement. On the other hand, all nodes in a ring or a randomly connected VN topology impose similar disjointedness requirement. This intensifies resource contention while allocating disjoint paths in the PN leading to longer paths, hence, the higher cost ratio.

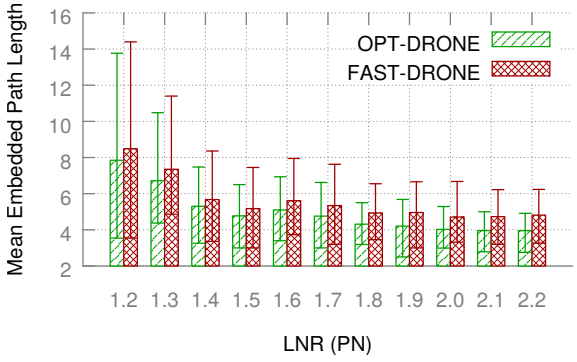
We also compute the mean physical path lengths for the embedded virtual links to validate this finding and present the result along with 5th and 95th percentile error bars in Fig. 3(b). The difference between mean embedded path length obtained by *FAST-DRONE* compared to *OPT-DRONE* is slightly higher for ring and randomly connected VN topologies (15% and 13%, respectively) compared to star VN topologies (12.5%). If we consider the 95th percentile of the spectrum, this difference is more significant, *i.e.*, 13.8%, 12%, and 10%, for ring, random, and star VN topologies, respectively.

3) 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. 4(a) shows the mean cost ratio with 5th and 95th percentile error bars against different LNRs. This plot gives us an idea about a good operating region for *FAST-DRONE*. As we can observe, *FAST-DRONE* allocates about 15% extra resources compared to the optimal solution for PNs having an LNR $\leq$ 1.8. For higher LNR, the increased path diversity may lead to more sub-optimal solution since the heuristic does not explore all the paths to keep the running time fast.

We also compute the mean of embedded physical path lengths corresponding to the virtual links and present the results in Fig. 4(b). The takeaway from this figure is that, a lower LNR in the PN, *i.e.*, sparse PNs cause both *OPT-DRONE* and *FAST-DRONE* to select longer paths for virtual link embedding. This is due to less path diversity in the PN. However, with increasing LNR, more paths become available in the PN and the mean path lengths for embedding virtual links become shorter. However, in line with the previous observation, *FAST-DRONE* explores a much smaller set of paths to keep the running time fast. As a result, the gap between mean embedded path lengths for *OPT-DRONE* and *FAST-DRONE* increases.



(a) Impact on Cost Ratio



(b) Impact on Mean Embedded Path Length

Fig. 4. Impact of PN Connectivity

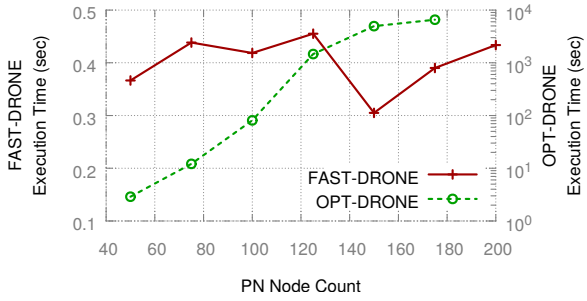


Fig. 5. Comparison of Execution time

4) Scalability of Heuristic: To demonstrate the scalability of *FAST-DRONE* we show the execution times of *FAST-DRONE* and *OPT-DRONE* on same problem instances in Fig. 5. 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 200 – 1200 times 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.

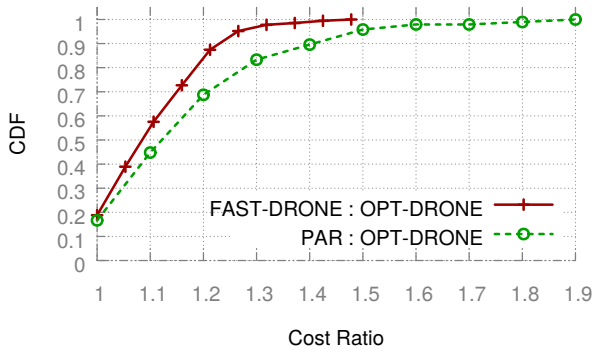
5) Comparison with PAR [12]: PAR [12] is a greedy heuristic for embedding a VN request on a PN with $1+1$-protection. PAR maximizes the probability of accepting a VN request by first embedding the virtual nodes on physical nodes with higher residual node capacities. After node embedding, PAR embeds the virtual links using a modified version of Suurballe's algorithm [25]. 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 [12]. It is worth noting that even after the modification in the node embedding, PAR could only find solutions for ≈12% test cases in our simulation setting.

We first compare how much extra resource is allocated by PAR and *FAST-DRONE* compared to the optimal solution (*OPT-DRONE*). For this comparison, we compute the ratio of costs[1] between PAR and *OPT-DRONE*, and *FAST-DRONE* and *OPT-DRONE*. Fig. 6(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. 6(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 90th percentile. On average, we found PAR to allocate 17.5% additional resources compared to *FAST-DRONE*.

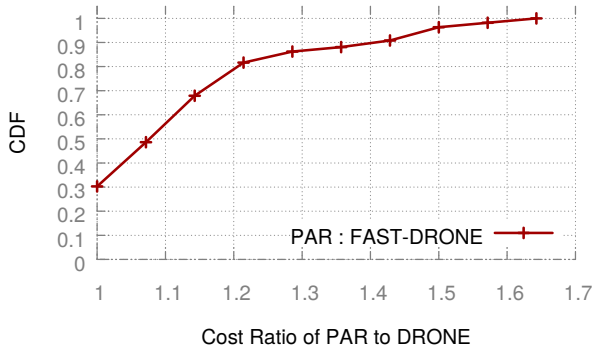
D. Steady State Analysis

Our steady state analysis focuses on the following aspects: (i) comparing the acceptance ratio obtained by *FAST-DRONE* with that of PAR [12] under different loads, *i.e.*, VN arrival rates (Section V-D1), (ii) compare the impact of *FAST-DRONE* on the load distribution on physical links with that of PAR (Section V-D2), and (iii) analyze topological properties of the solutions (Section V-D3). We perform the steady state analysis using the VN arrival rate and duration parameters described in Section V-A3 for a total of 10000 time units. The total number of VNs used in this simulation was varied between 400 to 960.

[1]cost is computed using (1)



(a) Comparison with *OPT-DRONE*



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

Fig. 6. Comparison between *FAST-DRONE* and PAR [12]

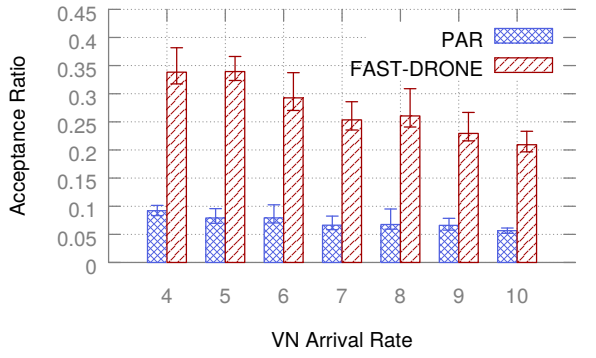
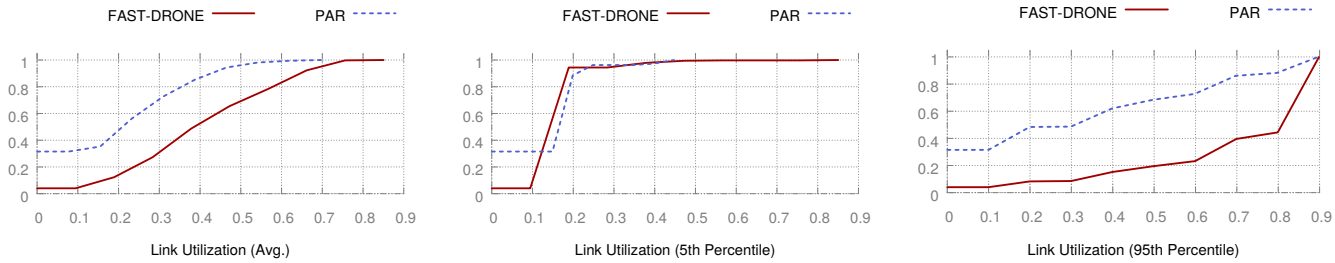


Fig. 7. VN Acceptance Ratio

1) Acceptance Ratio: In this section, we report our findings on the acceptance ratio obtained by *FAST-DRONE* and compare that with the acceptance ratio obtained by using PAR [12]. We consider the first 1000 time units of the simulation as the warm up period and discard the values from this duration. We take the mean of the acceptance ratio obtained during the rest of the simulation and report it along with 5th and 95th percentile values under varying VN arrival rates in Fig. 7. We can see from the results that *FAST-DRONE* outperforms PAR in all cases and accepts 4× more VNs on average over all VN arrival rates. There are two possible reasons contributing this



(a) Link Utilization (Avg.) (b) Link Utilization (5th Percentile) (c) Link Utilization (95th Percentile)

Fig. 9. Load Distribution on Physical Network

difference: (i) either the network is being quickly saturated by one algorithm leading to its lower acceptance ratio, or (ii) the one algorithm having lower acceptance ratio is not sufficiently exploring the search space to be able to utilize the PN resources, hence, leaving PN resources unused. In the next section, we analyze how *FAST-DRONE* and PAR distribute the load on the PN to gain more insight into the difference between their achieved acceptance ratios.

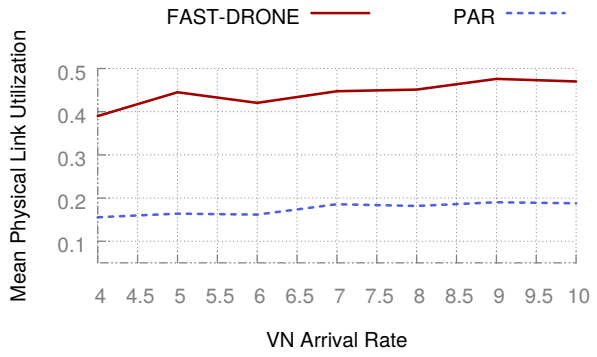
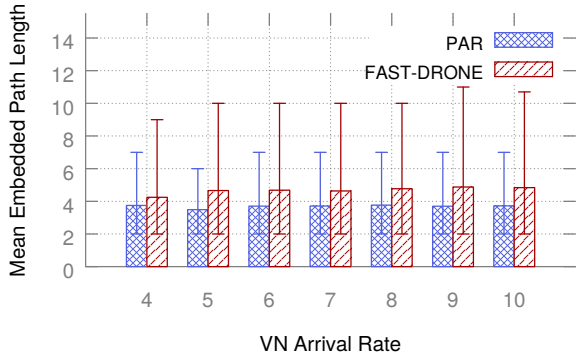


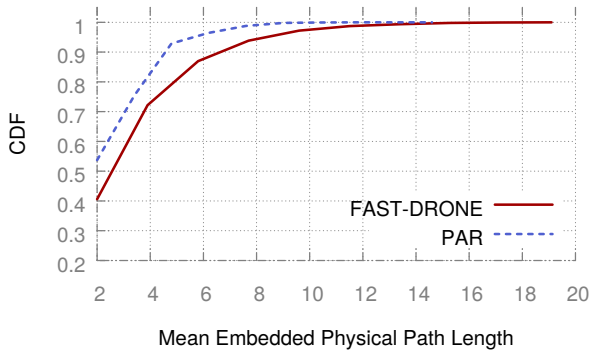
Fig. 8. Mean Physical Link Utilization with Varying Load

2) Load Distribution on PN: We measure the utilization of each physical link at each VN arrival and departure events, and compute the mean utilization for each physical link. We first present results showing the mean physical link utilization with varying VN arrival rates in Fig. 8. As we can see, there is a slight increase in the mean physical link utilization with increasing VN arrival rate. Also physical link utilization is on average 2.5× higher for *FAST-DRONE* compared to PAR. However, this plot, representing the mean utilization, fails to capture the variance in link utilizations and does not give us much insight into how the load is distributed across the PN.

In order to capture the essence of load distribution across the physical links, we compute the CDF of average, 5th and 95th percentile physical link utilization for each VN arrival rate. However, we found the CDFs for different VN arrival rates to follow similar trend, hence, we combined the CDFs for different VN arrival rates into one and present the results in Fig. 9. It is evident from Fig. 9 that a significant portion of the physical links (≈30%) remain unused by PAR throughout the simulation. In case of *FAST-DRONE*, the fraction of unused physical links is less than 5%. In addition, for any level of utilization x, the fraction of physical links having utilization $\geq x$ is larger for *FAST-DRONE* compared to PAR for all three cases, *i.e.*, average, 5th and 95th percentile. When load distribution is combined with acceptance ratio, we observe that despite having unused capacity in the PN, PAR yields lesser acceptance ratio compared to *FAST-DRONE*. This indicates that *FAST-DRONE* is exploring larger portion of the solution space compared to PAR, hence, the increased acceptance ratio.



(a) Mean Embedded Path Length vs. VN Arrival Rate



(b) CDF of Mean Embedded Path Length

Fig. 10. Topological Properties of Solutions

3) Topological Properties of the Solutions: We compute the mean embedding path lengths for the virtual links and present the results in Fig. 10. Fig. 10(a) shows the variation in mean path length with varying VN arrival rate, and Fig. 10(b) shows the CDF of mean path lengths over all VN arrival rates. The results show that for similar VN arrival rate and also for all VN arrival rates *FAST-DRONE* yields embeddings that have slightly longer mean embedding path length compared to PAR. According to our cost function (1), longer embedding

paths result into an increased cost. Therefore, results from these plots are counterintuitive when they are compared to that from Section V-C5, which indicated that *FAST-DRONE* yields more resource efficient embeddings compared to PAR. The reason behind this slightly longer paths during the steady state scenario is that, a higher acceptance ratio for *FAST-DRONE* pushes the network closer to saturation, hence, exhausting the shorter paths as more VNs are embedded. Therefore, *FAST-DRONE* is forced to choose the longer paths for the later VNs. In case of PAR, the lower acceptance ratio and the lower network utilization still leaves sufficient room in the shorter paths, resulting in shorter embedding paths on average.

VI. Related Works

In this section, we first discuss the research efforts that address different aspects of SVNE problem (Section VI-A). Then we focus our discussion on the works that are closely related to $1+1$-*ProViNE* (Section VI-B). Finally, we briefly discuss the known results regarding the hardness of unsplittable flow problem and graph partitioning problem (Section VI-C).

A. Survivable Virtual Network Embedding (SVNE)

Network survivability has been extensively studied in the context of mapping IP links over WDM optical networks [26], [27], [28], [29]. However, unlike VN embedding, IP link mapping over WDM networks assume that the endpoints are already mapped and addresses the issue of provisioning light paths for embedding IP links. In contrast, node embedding is as important as link embedding in VNE, and a coordination between node and link embedding have been shown to increase the acceptance ratio of VNs [5]. Ensuring survivability in the context of VNE was first addressed by Rahman *et al.* in [8]. They formulated the problem of ensuring survivability in VNs under single physical link failure as a Mixed Integer Program and proposed heuristics to obtain solutions in a reasonable time. However, unlike $1+1$-*ProViNE* their proposal does not protect VNs from node failures. A number of subsequent research works since then have addressed different aspects of SVNE such as considering node failures [30], [31], [32], [33], [34], optimizing backup resource allocation [35], [36], and ensuring certain level of availability of the virtual resources [37], [38], [39]. In the rest of this section, we briefly discuss some of the works that consider different aspects of SVNE and contrast them with our proposal.

[30], [31] are among some of the early works that consider node failure for SVNE. [30] addresses the issue of ensuring survivability under regional physical failures. A regional physical failure corresponds to a physical damage of a facility due to a disaster, leading to multiple physical node failures. [30] addresses the regional failure scenario by proactively provisioning backup nodes at different geographical locations. More recent research on ensuring survivability under regional failures, including [32] and [33], propose to take reactive measures after a regional failure and also allow VNs to operate with degraded QoS during regional failures for reducing backup provisioning overhead. On the other hand, [31] and [34] address the SVNE problem for a single physical node failure. [31] and [34] propose to enhance a VN with some additional virtual resources. During a physical node failure, virtual nodes and links are migrated to the additional resources to restore the VN. In contrast, we take a proactive approach and provide dedicated protection for each component of the VN to facilitate fast recovery.

Some SVNE approaches optimize the backup bandwidth provisioning by sharing the backup path of multiple virtual links [35] or by leveraging multi-path embedding of virtual links [36]. However, with a shared backup scheme such as [35], multiple virtual links sharing the same backup can suffer from degraded QoS during a single physical link failure. Khan *et al.* proposes to optimize the backup bandwidth provisioning by embedding a virtual link over multiple physical paths and provisioning a fraction of the virtual link bandwidth instead of the full bandwidth over each such path. The advantage of this approach is that it requires less backup bandwidth and can survive single link failures with full QoS. However, they assume virtual links can be splittable, which is not the case for $1+1$-*ProViNE*.

SVNE problem has also been addressed from the point of view of guaranteeing availability. [37] and [38] have addressed the issue of guaranteeing availability of virtual resources in the context of Virtual Data Centers (VDCs). A VDC is an extension to a VN to include compute, memory and storage resources. More recently, [39] has addressed the issue of ensuring VN availability on optical networks. These works determine the number of backups required for a virtual resource based on the historical reliability data on the physical resources. Subsequently, the embedding ensures that the primary and the backups provide a desired level of availability for that virtual resource. In contrast, the number of backup resources are already known for $1+1$-*ProViNE* and we need to find a resource efficient embedding.

B. Virtual Network Embedding with Dedicated Protection

The motivation for $1+1$-*ProViNE* comes from use cases in T-SDN virtualization, where customers are provided with full-fledged VNs instead of traditional end-to-end connectivity. A recent paper [11] identifies dedicated protection for an entire VN topology as one possible customer requirement for reliability among others. Therefore, it becomes important for the InP to embed a customer VN request with $1+1$-protection for the entire topology in a resource efficient way. In this context, the most related to our work is a recent work by Ye *et al.*, which addresses the problem of providing dedicated protection for VNE [12]. 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. [12] focused on increasing the VN request acceptance ratio over time, whereas we focus on minimizing the resource allocation cost for embedding VNs. 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, [12] does not consider the location constraint, which is an important constraint in our case. Another closely

related work is from Jiang *et al.* [40]. 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*. [40] 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.

C. Unsplittable Flow and Graph 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 [16]. Relevant literature shows that they are computationally hard to solve. Finding a constant factor approximation algorithm for these problems for general graphs is still open [20], [21], [41]. The best known approximation ratio for graph partitioning is not constant, rather it is a poly-logarithm function of the number of nodes [21]. On the other hand, the first constant factor approximation algorithm for the unsplittable flow problem with known sources and destinations had $(7+\epsilon)$ and $(8+\epsilon)$ approximation ratio for simple line graph and cycle graph, respectively [20]. Approximation ratio for the same types of graphs has recently been improved to $(2+\varepsilon)$ by Anagnostopoulos *et al.* [19]. Friggstad *et al.* has recently proposed a Linear Programming relaxation based algorithm for unsplittable flows on trees [42]. However, the approximation ratio for this LP relaxation based algorithm is not constant, rather it is a logarithmic function of the number of nodes. 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 formulated $1+1$ **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 using both real and synthetic topologies 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*. Simulation results also show that *FAST-DRONE* can accept 4 times more VN requests compared to the state-of-the-art solution [12].

We believe that the formulation of the $1+1$-*ProViNE* problem 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.

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 *Proc. of IEEE INTERNET*, 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," *Comm. Letters, IEEE*, vol. 16, no. 5, pp. 756–759, 2012.

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

[9] S. Ramamurthy and B. Mukherjee, "Survivable WDM mesh networks. Part I-protection," in *Proc. of IEEE INFOCOM'99.*, 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] W. Wang, Y. Lin, Y. Zhao, G. Zhang, J. Zhang, J. Han, H. Chen, B. Hou, Y. Ji, and L. Zong, "First demonstration of virtual transport network services with multi-layer protection schemes over flexi-grid optical networks," *Comm. Letters, IEEE*, vol. 20, no. 2, pp. 260–263, 2016.

[12] 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.

[13] 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.

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

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

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

[17] S. R. Chowdhury, R. Ahmed, M. M. A. Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng, "Protecting virtual networks with Drone," in *Proc. of IEEE/IFIP NOMS 2016*, Istanbul, Turkey, April 2016.

[18] 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.

[19] A. Anagnostopoulos, F. Grandoni, S. Leonardi, and A. Wiese, "A mazing 2+ ε approximation for unsplittable flow on a path," in *Proceedings of ACM-SIAM SODA 2014*. SIAM, 2014, pp. 26–41.

[20] P. Bonsma, J. Schulz, and A. Wiese, "A constant factor approximation algorithm for unsplittable flow on paths," in *Proceedings of IEEE FOCS 2011*, 2011, pp. 47–56.

[21] 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).

[22] 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.

[23] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with Rocketfuel," in *ACM SIGCOMM CCR*, vol. 32, no. 4. ACM, 2002, pp. 133–145.

[24] 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.

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

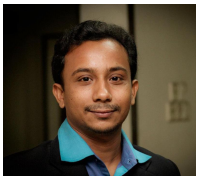
[26] M. Kurant and P. Thiran, "Survivable mapping algorithm by ring trimming (SMART) for large IP-over-WDM networks," in *Proc. of IEEE BroadNets*, 2004, pp. 44–53.

[27] Z. Zhou, T. Lin, K. Thulasiraman, G. Xue, and S. Sahni, "Novel survivable logical topology routing in ip-over-wdm networks by logical protecting spanning tree set," in *IEEE ICUMT*, 2012, pp. 650–656.
[28] C. Liu and L. Ruan, "A new survivable mapping problem in ip-over-wdm networks," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 3, pp. 25–34, 2007.
[29] M. Kurant and P. Thiran, "Survivable routing of mesh topologies in ip-over-wdm networks by recursive graph contraction," *Selected Areas in Communications, IEEE Journal on*, vol. 25, no. 5, pp. 922–933, 2007.
[30] 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.
[31] 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.
[32] X. Liu, Y. Wang, A. Xiao, X. Qiu, and W. Li, "Disaster-prediction based virtual network mapping against multiple regional failures," in *Proc. of IEEE/IFIP IM*, 2015, pp. 371–378.
[33] M. Pourvali, K. Liang, F. Gu, H. Bai, K. Shaban, S. Khan, and N. Ghani, "Progressive recovery for network virtualization after large-scale disasters," in *Proc. of IEEE ICNC*, 2016.
[34] B. Guo, C. Qiao, J. Wang, H. Yu, Y. Zuo, J. Li, Z. Chen, and Y. He, "Survivable virtual network design and embedding to survive a facility node failure," *Journal of Lightwave Technology*, vol. 32, no. 3, pp. 483–493, 2014.
[35] 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.
[36] M. M. A. Khan, N. Shahriar, R. Ahmed, and R. Boutaba, "SiMPLE: Survivability in multi-path link embedding," in *Proc. of ACM/IEEE/IFIP CNSM*, 2015, pp. 210–218.
[37] 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.
[38] Q. Zhang, M. F. Zhani, M. Jabri, and R. Boutaba, "Venice: Reliable virtual data center embedding in clouds," in *Proc. of IEEE INFOCOM*, 2014, pp. 289–297.
[39] H. Jiang, Y. Wang, L. Gong, and Z. Zhu, "Availability-aware survivable virtual network embedding in optical datacenter networks," *Optical Communications and Networking, IEEE/OSA Journal of*, vol. 7, no. 12, pp. 1160–1171, 2015.
[40] H. Jiang, L. Gong, and Z. Zuqing, "Efficient joint approaches for location-constrained survivable virtual network embedding," in *Proc. of IEEE GLOBECOM*, 2014, pp. 1810–1815.
[41] M. Skutella, "Approximating the Single Source Unsplittable Min-cost Flow Problem," *Mathematical Programming*, vol. 91, no. 3, pp. 493–514, 2002.
[42] Z. Friggstad and Z. Gao, "On linear programming relaxations for unsplittable flow in trees," in *LIPIcs-Leibniz International Proceedings in Informatics*, vol. 40. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

Shihabur Rahman Chowdhury is a PhD candidate at the David R. Cheriton School of Computer Science, University of Waterloo, Canada. He received his B.Sc. degree in computer science and engineering from Bangladesh University of Engineering and Technology (BUET) in 2009. His research interests include virtualization and softwarization of computer networks. He is a recipient of the Ontario Graduate Scholarship, Presidents Graduate Scholarship, and GoBell Scholarship at the University of Waterloo.

Reaz Ahmed received his Ph.D. degree in computer science from the University of Waterloo, in 2007. He received M.Sc. and BSc. degrees in computer science from BUET in 2002 and 2000, respectively. He received the IEEE Fred W. Ellersick award in 2008. His research interests include future Internet architectures, Information-Centric Networks, Network Virtualization and content sharing peer-to-peer networks with focus on search flexibility, efficiency and robustness.

Md Mashrur Alam Khan is currently employed as a Software Engineer in Cisco. He obtained MMath in Computer Science at the University of Waterloo, Waterloo, ON, Canada in 2015. He received his Bachelor of Science (B. Sc.) degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology in 2012. His research interests include network virtualization, virtual network embedding, fault tolerance, and Internet of things.



Nashid Shahriar is a Ph.D. candidate at University of Waterloo, Canada. He received MSc. and BSc. degrees in Computer Science and Engineering from Bangladesh University of Engineering and Technology in 2011 and 2009, respectively. His research interests include network virtualization, future Internet architectures, and network function management. He is a recipient of David R. Cheriton Graduate Scholarship at the University of Waterloo.

Raouf Boutaba received the M.Sc. and Ph.D. degrees in computer science from the University Pierre & Marie Curie, Paris, in 1990 and 1994, respectively. He is currently a professor of computer science at the University of Waterloo (Canada). His research interests include resource and service management in networks and distributed systems. He is the founding editor in chief of the IEEE Transactions on Network and Service Management (2007–2010) and on the editorial boards of other journals. He received several best paper awards and recognitions including the Premiers Research Excellence Award, the IEEE ComSoc Hal Sobol, Fred W. Ellersick, Joe LociCero, Dan Stokesbury, Salah Aidarous Awards, and the IEEE Canada McNaughton Gold Medal. He is a fellow of the IEEE, the Engineering Institute of Canada, and the Canadian Academy of Engineering.

Jeebak Mitra received M.A.Sc. and Ph.D degrees in Electrical Engineering from University of British Columbia (UBC), Canada in 2005 and 2010 respectively. He was the recipient of the Best Student Paper Award at the IEEE Canadian Conference in Electrical and Computer Engineering (CCECE) 2009. From May 2010 to Dec 2011 he worked as Senior DSP engineer with Wimatek Systems leading the system level design for an LTE baseband. From 2011 to 2012, he was team lead for the Wireless DSP team at BLINQ Networks, Ottawa, working on backhaul products for small cells. He is currently working as a Staff Engineer at Huawei Technologies Canada Research Center, Ottawa in the areas of algorithm design and implementation of high-speed ASICs for optical transceivers and flexible optical network design, since 2013. His research interests lie broadly in the area of physical layer design aspects for fiber-optic and wireless networks with an emphasis on high-performance communication systems. He regularly serves as a reviewer for several conferences and journals in related areas.

PLACE PHOTO HERE

Feng Zeng received the master degree from the University of Electronics Science and Technology of China, majoring in communication and information. He worked in high performance Router PDU of H3C. In 2008, he joined Huawei and mainly focused on network plan, evaluation and optimization algorithm.